Exploiting App Differences for Security Analysis of Multi-Geo Mobile Ecosystems

by

Renuka Kumar

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2023

Doctoral Committee:

Professor Roya Ensafi, Co-Chair
Professor Atul Prakash, Co-Chair
Professor Vineet Kamat
Professor Morley Mao

Renuka Kumar

renukak@umich.edu

ORCID iD: 0000-0002-0395-7268

© Renuka Kumar 2023

DEDICATION

Dedicated to Amma, Sri Mata Amritanandamayi Devi.

ACKNOWLEDGMENTS

I wrote this acknowledgment well before starting my thesis when I had much to be grateful for the few people in my life without whom I would not have seen the finish line.

First, I would like to thank my advisor extraordinaire, Professor Atul Prakash, for his immense faith in my abilities when I had none in myself and for teaching me to think strategically in professional and personal matters. Professor Prakash has left an indelible mark on me by showing me how to stay calm through adversity. His encouragement and support gave me the courage to pursue a Ph.D. at the University of Michigan. I thank my co-advisor, Professor Roya Ensafi, for sharing her passion and vision of the world being a better place and allowing me to participate in her fight to build equality through technology. Professor Ensafi's friendship, wit, and charm are something I will always cherish. The freedom and support my advisors have given me are what saw me through this process. I thank Professor Vineet Kamat and Professor Morley Mao, my wonderful doctoral committee, for their valuable insights and feedback during my thesis writing process. I also thank Professor Daniel Genkin for his friendship, mentorship, and support.

To my lab mates and peers— Reethika Ramesh, Ram Sundara Raman, Shruti Nagaraja, Nishil Talati, Apurva Virkud, Diwen Xue, Ryan Feng, Andrew Kong, Haizhong Zheng, Elisa Tsai, Hao Lu, and Marina Minkin . Thank you for walking with me every step of the way, for your willingness to help, and for all the party and board game nights.

To Amrita Vishwa Vidyapeetham (University) for giving me free rein to pursue projects close to my heart during my tenure there. The impact of my work there paved the way for my Ph.D. at the University of Michigan. I am grateful to Professor Bharat Jayaraman and the many faculty and staff there that supported my research.

My gratitude to a few extraordinary people who always rooted for me and whose presence was beyond solace. To Anvita Prakash (w/o Professor Atul Prakash) for her kindness, unconditional support, and simple words of reason. To Prabhakar Krishnan (PK), for his countless counsels and for reassuring me every time that I am turning the corner even when it seemed otherwise. To my friends Preeti Pillai, Morgan Russell, Sreesh Kishore, Radhesh Krishnan, and Rakesh K.T. for all the laughter and fun and for reminding me that I still got it in me. To Jayakumar Muthukumarasamy (JK) and Ullas Ramanadhan, my managers from my previous industry jobs, for their constant support in my career.

I cannot express in words the sacrifices my family has made as I chose to embark on this journey rather late in life. They have cheered me for as long as I can remember. My mother has toiled and sacrificed years of her life so I could pursue my passion. For her, this win is beyond personal. I am beyond grateful to my father for setting me free and supporting me as I carved my own path in life. To my sister Aruna Binuraj and her family for constantly listening to my woes and for being ever willing to support me. To my brother Krishna Kumar and his family for giving me stability, a home, and a footing to start fresh. To my uncle Satish Menon and aunt Padma Menon for sheltering me unconditionally in more ways than I can enumerate for over two decades; you are why I am still (partly) sane. To uncle Hari Pillai, for the $250 that he lent me to write the GRE way back in the year 2000—it sure did take me a long way. To Swami Purnamritananda Puri, who taught me to find grace and courage from within me as I faced failure. To Swami Pranavamritananda Puri for showing me that passion is all it takes for perfection. To Swami Amritatmananda Puri for sharing his life experiences as I sought purpose. To Swami Dayamritananda Puri for his friendship. To my family—this is as much yours as it is mine.

And to everyone else, whether you realize it or not, to those whose prayers I have asked for and so liberally gotten, thank you!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Billions of users worldwide access essential Internet services such as banking, education, and healthcare through mobile apps on their phones. This growth in mobile use is fueled by the rise of large platforms that provide a common backend infrastructure to provide free services to users. These platforms drive entire ecosystems by allowing competing app developers to integrate their apps with their shared backends for users worldwide to consume and carry out billions of dollars in transactions. However, security and privacy vulnerabilities in these widely deployed ecosystems compromise millions worldwide. This dissertation raises this question and asserts that there is, in fact, a practice gap in the security and privacy offerings of widely deployed mobile ecosystems.

This thesis presents and discusses the security analysis of two of the world's largest mobile ecosystems: (i) Google Play for app distribution and (ii) the Unified Payments Interface (UPI) for free bank-to-bank micropayments. We make significant contributions by demonstrating how security analysis and measurements of these black-box systems can be made feasible at scale even within the confines of a severely fragmented ecosystem, despite having no sophisticated tools or access to their backend infrastructure. We reverse-engineer these security-hardened ecosystems across nation-state boundaries from the point of view of an attacker (or user) having access to multiple vantage points, specifically, multiple versions of highly-rated apps integrated with these platforms.

This thesis exposes critical and foundational flaws in these mobile ecosystems that expose millions of users to significant security and privacy threats, even when using highly-rated apps from official app markets. In our study of the UPI ecosystem (which first emerged as a regional ecosystem in India for payments), we expose severe flaws in the design of UPI's multi-factor authentication protocol as well as the payment apps integrated with UPI which, when combined with region-specific vulnerabilities, can enable an attacker to remotely launch large-scale attacks even without any knowledge of its user. Our disclosures led to the Indian Government acknowledging and addressing the core vulnerabilities we found, releasing an upgraded 2.0 version of the payments infrastructure. We also obtained several CVEs for our vulnerability disclosures on payment apps.

Through our empirical investigation of thousands of highly-rated, essential apps on Google Play from vantage points in 26 countries, we show how users in some countries are at a higher risk of attack because developers selectively release apps with weaker security settings or privacy disclosures. We uncover a significant amount of geoblocking of essential apps on Google Play that disproportionately isolates some countries; we root cause the actor responsible for it. We open-source our code and dataset, the largest multi-country app dataset, to foster further research. The concerns raised by this research were acknowledged by the highest levels of Google's privacy teams and covered by over 25 news websites worldwide.

Thus, this thesis shows how complex black-box ecosystems can be analyzed end-to-end despite the barriers to measuring them. Our experience with app disclosures reveals that the vulnerabilities we uncovered may take years to resolve. We provide several actionable recommendations for platform owners and developers to address the issues we find, such as removing barriers for the security community to audit these ecosystems, vetting apps for compliance with MITRE's recommendations for app developers, and performing end-to-end testing of apps from multiple vantage points.

CHAPTER 1

Introduction

1.1 Motivation

Mobile devices have become an intrinsic part of people's everyday lives; they have become the preferred way and, for some, the only way to access Internet content and services. Over 83% of the world's population are smartphone users [35, 64], with several countries pushing for a mobile-first digital economy by democratizing mobile services such as banking. Users worldwide access essential services such as banking, healthcare, education, e-commerce, messaging, and social media via mobile apps on their devices. So much so the traffic from mobile devices has far surpassed traditional web traffic [225]. As Phil Nickinson, the editor of *AndroidCentral.com*, aptly said, "Your mobile device quickly has become the easiest portal into your digital self."

This massive shift towards a mobile-first paradigm has been fueled by the emergence of large mobile platforms that provide free services to users across nation-state boundaries, such as money transfer services for banking and payments or app distribution services for access to other mobile apps. These platforms drive the growth of large mobile ecosystems by providing a common backend infrastructure onto which thousands of competing app developers worldwide can integrate their apps and release them for public use. These platforms drive millions of users to their backend infrastructures and can carry out billions of dollars in transactions.

However, vulnerabilities in these ecosystems will impact millions of users. For instance, a security and privacy hole in any of the competing popular apps integrated with these platforms can expose millions of users of these apps to security and privacy risks. And security and privacy holes in these platforms themselves can impact the entire ecosystem as a whole. By virtue of a mobile device's proximity to a user, any compromise in the mobile ecosystem has significant stakes since attackers can leak fine-grained information about a user besides other implications, such as possible theft of money.

What we are curious to learn about is the security guarantees of these widely deployed mobile ecosystems in the presence of an attacker having access to multiple vantage points. For instance, what information could an attacker leverage if an attacker were to examine in-depth any one of the competing apps and their different versions that share the platform's common backend infrastructure within a region? How would the security guarantees of these ecosystems change if, in addition, an attacker could leverage a vantage point on a user's device that can, unbeknownst to the user, capture some part of a user's communication with the platform. Or, what maybe different if an attacker can access vantage points in different geolocations (countries) from where other versions of the same app maybe visible.

1.2 Thesis Objective

Thesis Statement. This dissertation raises this question and asserts that there is, in fact, a practice gap in security and privacy implementations of widely deployed mobile ecosystems by conducting end-to-end studies. Through principled data-driven techniques, we reverse-engineer complex black-box platforms and the apps integrated with them to evaluate the security posture of these hardened real-world mobile ecosystems from the perspective of an attacker having access to multiple vantage points, specifically, multiple versions of highly-rated popular apps from an official app market. We further aim to identify the actors and threat models that can potentially breach these mobile ecosystems. Based on the findings of our study, we then aim to close the research to practice gaps in these ecosystems by providing actionable operating recommendations to platform owners and developers to address the issues we find. By exposing these security and privacy gaps and bringing them to public knowledge, this thesis will help the security community, governments, and businesses to build a secure and open mobile ecosystem at scale by bringing transparency and accountability to how these critical ecosystems operate. To the best of our knowledge we are the first to conduct end-to-end studies of widely deployed mobile ecosystems.

To do this, we pick two of the world's largest mobile ecosystems: (i) Google Play which provides a platform for mobile app development, hosting, distribution, and billing to users worldwide; and (ii) Unified Payments Interface (UPI) which facilitates free bank-to-bank micropayments at scale from a mobile platform. These widely deployed ecosystems have several commonalities. For instance, they provide free services and a consistent user experience that drives millions of users in different geolocations to their common backend infrastructures. They have hundreds of apps integrated with their platforms that are distributed only after a thorough security vetting. They are well-known for being designed and deployed with security and privacy offerings such as multi-factor authentication and encrypted communication. And they carry out billions of dollars worth of transactions via their common backend infrastructures [26, 195]. However, despite being widely deployed, not much is known about the security guarantees these large-scale mobile ecosystems provide.

While studying practice gaps is common in medical and management domains [3,3,36,46,123], it is limited to a literature study in information security [72,72,150,250]. Going beyond literature study to analyze real-world deployments is seldom done owing to the challenges entailed in measuring complex black-box ecosystems and the need for more tools to do automated analysis across nation-state boundaries. Continuing real-world attacks on users show that even widely deployed systems are yet to factor in attack vectors introduced by the many stakeholders of these systems. Given the recent eminence of mobile ecosystems, much work needs to be done to understand the security guarantees these ecosystems provide in the face of continually evolving threats.

Though platform owners have made a concerted effort to develop systems following secure design principles, typically only their functional correctness is evaluated during real-world pilot deployments. Security assessments are carried out pre-deployment through a series of internal penetration tests, vulnerability assessments, and compliance checks that only look for pre-canned

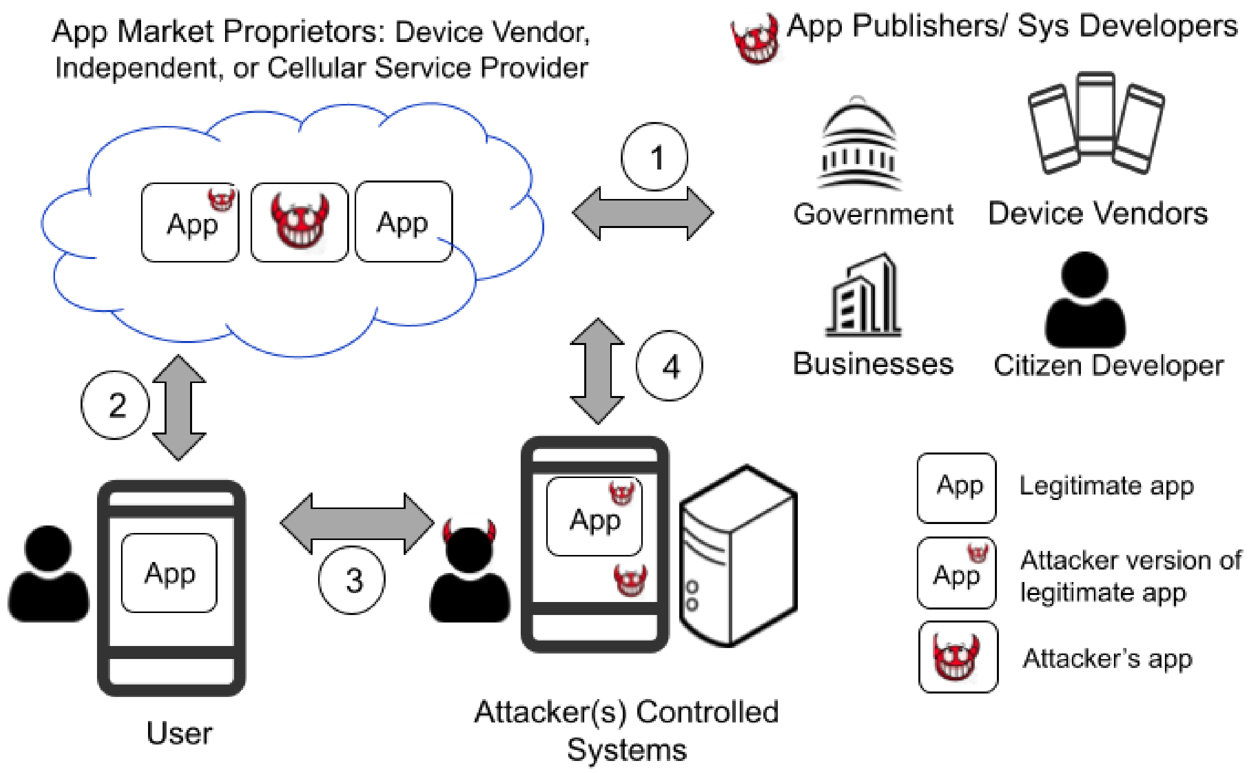

App Market Proprietors: Device Vendor, Independent, or Cellular Service Provider
App Publishers/ Sys Developers
App
App
App
1
Government
Device Vendors
Businesses
Citizen Developer
2
4
3
App
App
User
Attacker(s) Controlled Systems
App
Legitimate app
App
Attacker version of legitimate app
Attacker's app

Figure 1.1: **Stakeholders and User's view of Attacker Vantage Points.** The figure shows the various stakeholders of a mobile ecosystem and the attacker's vantage points from a user's perspective. Developers of a mobile system may be governments, businesses, device vendors (e.g., Google and Amazon), and netizens. From a user's perspective, any of these stakeholders may be malicious. (i) A mobile system developer builds the backend infrastructure and publishes their client app with one or more of the app markets for distribution. (ii) A user purchases a device from one of the many device vendors and may download apps from one or more app markets for personal use. These apps communicate with the mobile system's backend infrastructure (iii) A compromised user (app or device) may also communicate with attacker-controlled systems to leak data (iv) An attacker too can release a mobile system and trojanized apps for public use.

vulnerabilities or known attacks, that too within the confines of a pre-defined threat model. Users have no knowledge of these assumptions or the user action(s) that may compromise them.

1.2.1 Contributions

This thesis makes the following significant contributions:

1. We conduct a security analysis of two of the world's largest mobile ecosystems— Google Play and UPI. We perform this analysis, one in-depth, from within the confines of a region

where it was first released (UPI in India), and the other from the vantage point of 26 diverse countries (Google Play).

2. Through principled techniques and carefully crafted measurements, we show how these complex, black-box systems could be analyzed at scale even within the confines of a severely fragmented ecosystem, despite having no sophisticated tools or access to their backend infrastructure. We reverse-engineer these ecosystems across nation-state boundaries from the point of view of an attacker (or user) having access to multiple vantage points, specifically, multiple versions of highly-rated apps integrated with these platforms.

3. We conduct the first security analysis of a critical mobile ecosystem—that of UPI, the world's largest and the first-of-its-kind mobile payments ecosystem that utilizes a common backend infrastructure (Chapter 3). We conducted this study in the region-specific context of India, where UPI was first released. While mobile payment apps have been studied for vulnerabilities in the past, we are the first to conduct a security analysis of a payments system end-to-end from the vantage point of the most popular payment apps integrated with UPI.

    (a) We show how different weaknesses in the security and privacy implementations of these apps, can be used to leak sensitive protocol data. Protocol data leaked from an app with weaker security settings can then be leveraged to launch attack using a more secure app (Section 3.3.1).

    (b) We found several security and privacy flaws in the design of UPI's protocol, that allows an attacker to launch scaleable attacks against a user remotely, even without any knowledge of a user. Some of these flaws persisted even after the Indian government released a patch that fixed a core vulnerability we exploited (Section 3.3).

    (c) Through our study on UPI, we show how attackers can leverage region-specific vulnerabilities (e.g., access to low cost cell phones and that India is in the list of countries with most number of potentially harmful applications (PHAs) pre-installed on devices) along with the vulnerabilities that we discover to automate these attacks.

   (d) **Impact.** Disclosures of this work led to the Indian Government addressing the core vulnerabilities we found to release an upgraded version of the payments infrastructure. We obtained several CVEs for this work and mobile payment vendors like Samsung addressed our disclosures for which we received a bounty of $5000.

4. We build on our lessons from the study of UPI to conduct the first large-scale empirical investigation of the mobile app ecosystem by examining geodifferences in highly-rated essential apps on the Google Play store, the world's largest mobile app ecosystem, as experienced by users from 26 countries (vantage points) at the same time (Chapter 4). We aim to determine if users in diverse geographies have access to these essential apps and if so, whether or not their security and privacy offerings vary.

   (a) We designed and implemented a semi-automated measurement test bed that can download thousands of apps in parallel from Google Play (Section 4.3). We collect the largest multi-country app dataset and their privacy policies (117,233 apk files and 112,607 privacy policies for those apps). We release our measurement code and data for further research at `https://github.com/censoredplanet/geodiff-app`.

   (b) We uncover significant amounts of geoblocking on Google Play that disproportionately isolates users of some countries and root cause the actors responsible for it. While our data corroborates anecdotal evidence of takedowns. due to government requests, unlike common perception, we find that blocking by developers is significantly higher than takedowns in all our countries and has the most influence on geoblocking in the mobile app ecosystem (Section 4.5).

   (c) We show how users in some countries are at a higher risk of attack because developers selectively release apps with weaker security settings or privacy disclosures there (Section 4.5.3).

   (d) **Impact.** The concerns raised by our research was acknowledged by the highest-levels of Google's privacy teams with a write up of this work appearing on over 25 news sites worldwide.

5. Through our large scale studies, we break down the complexity of these large ecosystems, identify the variables that impact principled analysis of these ecosystems, and show how these black-box systems can be studied both in-depth and in breadth. We expose critical and fundamental weaknesses in these ecosystems that expose millions of users of these systems to significant security and privacy threats even when using highly-rated apps from an official app market. In reality, we show how these ecosystems fail to secure users even within the confines of the threat model they have defined. These vulnerabilities are thus beyond the scope of what users can address.

6. We show how slight but very realistic changes to the threat model can easily compromise these systems effectively, not just violating the CIA properties but also more broadly: (i) allowing an adversary to launch scalable attacks remotely that can take over a user's account (Section 3.3.3.2) (ii) deny access to essential apps, in some cases effectively isolating entire countries (Section 4.5.2) or (iii) provide apps with weak security and privacy protections based on a user's geolocation (Section 4.5.3).

7. Our experience with app disclosures reveals that many of these vulnerabilities result from design decisions made by these platform owners and are left behind knowingly (Section 5.1). As a result, many of the issues we find may take months, if not years, to resolve, leaving users unprotected for a prolonged period.

8. Our research shows how security and privacy is impossible to achieve when left to the decisions of platform owners alone though they are the best positioned to regulate these ecosystems at scale, by virtue of their unique position at the top of the ecosystem. While security and privacy policies exist, our research shows that the controls that enforce these policies are insufficient. We bring to light outline several reasons why platform owners must rethink their policies to help mitigate the issues we find (Section 5.2).

9. We provide several actionable recommendations to address the issues we find, such as removing barriers for researchers and third parties to audit these ecosystems, adding barriers for country-targeting of apps with different security and privacy implementations, additional controls to vet applications for compliance with MITRE's recommendation for app developers and perform end-to-end testing of applications from multiple vantage points (Section 5.3).

This thesis comes at a time when nation-states are increasingly adopting UPI's model for democratizing mobile banking for tech inclusion and regulating global app market proprietors like Google Play to foster alternate app markets. We provide several future research directions the community can take to find and resolve the issues persistent in the ecosystem proactively. We believe this thesis will help the security community, governments, and businesses wanting to build a secure and open mobile ecosystem at scale.

1.2.2 Challenges in Security Analysis

A mobile ecosystem consists of different stakeholders: (i) a user, (ii) an app or a system developer that releases mobile apps that act as client-side interfaces to a mobile system, (iii) mobile devices on which users install these apps that communicate with a system backend, and (iv) mobile app markets from where users download the apps they consume (See Figure 1.1).

Given the diversity and complexity of the mobile ecosystem, conducting large-scale studies is non-trivial and is beset with significant challenges. For one, our goal is to examine workflows of mobile systems as perceived by users without backend access to servers. We thus have a significant challenge of meticulously reverse-engineering complex black-box systems with limited documentation on their inner workings, uncovering their security offerings, and learning their client-server handshake remotely. Unlike the web ecosystem, there are no well-known, fully automated tools that can be used to study these mobile systems, be it mobile apps or networks. Besides, given the different stakeholders, the mobile ecosystem is severely fragmented, with numerous devices, app versions, and operating system versions in circulation today, making it impossible to develop fully automated tools for analysis.

Given how we study multi-geo mobile ecosystems, our research spans nation-state boundaries and may potentially take months, if not years, to complete. This means that for our work to be fruitful, we need stable and reliable vantage points in our countries of study. And given the number of unknowns (or variables) in conducting measurements remotely, we require techniques to clearly delineate transient errors from actual vulnerabilities or violations. Prior to conducting our analysis, we also have to conduct several preliminary experiments to understand these ecosystems and identify the variables that can impact our studies when conducted over a prolonged period. We also have to augment our observations with network measurements whenever required. Since studying such systems may span several months, our experimentation must be cognizant of potential server-side or app updates that can change the course of this study.

There may also be regional trends that facilitate or impact analysis. For instance, studies have shown how popular devices may exhibit different types of vulnerabilities when operating within a region [233]. Certain regions may also have specific versions of operating systems that are popular or may even be prone to certain attack vectors more than others. That apart, developers may also release different app versions optimized for different devices and countries; hence, reverse engineering these apps may require other techniques. Overall, our design choices must factor in or rule out the variables that can impact our long-running analyses.

1.3 Thesis Roadmap

In Chapter 2, we provide a background for this study that helps understand the stakeholders of the mobile ecosystem, potential attack vectors, and includes details on common hardening techniques we look for in apps as we try to reverse engineer them. We also contextualize our study in the context of other large-scale studies. In Chapter 3, we present our first exhaustive study of the world's largest mobile payments infrastructure building on the material presented in our 2020 publication titled S̈ecurity Analysis of the Unified Payments Interface and Payment Apps in India.̈ In Chapter 4, we will discuss the first empirical investigation into the geodifferences in Google Play as described in our 2022 publication “A Large-scale Investigation into Geodifferences in Mobile Apps”. In Chapter 5, we will discuss insights and recommendations for security researchers, governments, and businesses to build, secure, and FOCI the mobile ecosystem at scale (towards submission to FOCI). In Chapter 6, we will briefly discuss future directions for reseach, review the material presented, and conclude.

CHAPTER 2

Background & Related Work

2.1 Geodivide in Mobile Ecosystem

The term ’digital divide’ has been used to refer to the gap that exists across different countries with regards to ready access to information and communication technologies and the knowledge they offer. Prior studies have shown how a dramatic gap exists between the developed and the emerging economies in their access to the Internet both via the traditional web as well as mobile [62, 62, 95, 118, 214, 251–253]. These studies are either literature studies or surveys that ascertain factors that aggravate the digital divide such as those that deter physical access to Internet technologies or socio-economoic factors that deter technology adoption such as age group of users, gender, language restrictions, and income. These studies are limited in that, the countries or population explored is selected and judged according to political interests or subjective perspectives, with each study showing a perspective that is not based on empirical measurements [252]. In the last few decades there have been a significant reduction in the gap in physical access to technology in most parts of the world [252], but not much is known on the extent of geodivide that exists in the mobile ecosystem that is backed by real-world data.

In this thesis, we study differences in access, security, and privacy of the same app across diverse geographies (which we call geodifferences), a form of geodivide, in the context of inequities in mobile apps based on a user’s geolocation, as well as differences in security and privacy of a category of apps within a region (India) and its impact on the security of a large mobile ecosystem.

For our broad empirical study, like prior research, we consider the most developed countries like the USA, the UK, Germany, Australia, and Canada as the baseline for our studies. Given how mobile access has become the primary and for some income groups the only access to the Internet content in some emerging economies like India, there are government backed efforts to for tech inclusion and adoption. To explore region-specific differences in security and privacy, we chose essential mobile apps on a centralized backend payments infrastructure which the Indian government introduced as a means to facilitate financial and digital inclusion. India is one of the countries that experiences the most digital divide, despite having the world's second largest pool of Internet users [201].

2.2 Stakeholders of Mobile Ecosystems

Mobile ecosystems consists of several different stakeholders, which are entities, groups or processes that interact with each other and have much to gain or lose when a mobile system is compromised. Broadly, we identify the following stakeholders (also shown in Figure 1.1):

1. **Users**: Users are the most critical of all the stakeholders, as any compromise of a mobile system will directly impact them through a loss of their sensitive data or money. Widely deployed regional mobile payment systems like the Unified Payments Interface in India have over 300 million active users as of 2021 [34], while Google Play has over 1 billion users worldwide in 2022 [75]. Thus any attacks on these systems will also impact millions of users.

2. **Mobile App or System Developers**: Mobile apps are front-end interfaces that users use to access Internet content and services via a mobile device. Developers of mobile apps and their backend systems include governments (such as in the case of the UPI payments infrastructure), businesses (as in the case of Google apps), and any citizen developer. App developers have much to gain from a user installing their app as financial gains or benefits from profiling a user's sensitive data, which may then be sold or exploited to launch attacks. Thus developers profit from distributing their apps for maximum reach. Most apps are either available for free or have a freemium version with some essential features, with more sophisticated features

available through in-app purchases. These mobile apps belong to different app categories. There are about 49 different app categories on Google Play, for instance. These categories range from social media, dating, lifestyle health, news, and education, to name a few.

3. **Device Vendors**: Device vendors are suppliers that release various flavors of mobile devices with their diverse hardware and operating system. Users purchase their mobile devices from one of many device vendors (e.g., Google, Apple, and Samsung). Amongst mobile devices, Android devices have the largest market share in the world, primarily due to Google open-sourcing its stock OS. Global cell phone vendors release cheaper alternatives to gain market competition, especially in certain regions. For instance, Google released Android One, its cheaper version, only for India. Competing with these global device vendors are also regional device vendors that release low-cost alternatives for the less financially endowed population in emerging economies.

4. **App Market Proprietors**: Developers rely on app markets to distribute their apps to users. There are global centralized app markets like Google's Play Store and Apple's App Store, app markets hosted by independent publishers like ApkMonk and APKPure, and regional app markets such as CafeBazar in Iran. While app markets like Google Play have an app vetting process before they publish apps on their app market, many of these app market proprietors have little to no app vetting and thus provide a lower bar for developers to publish their apps. Some of these third-party app markets are more popular in certain regions, and users often sideload apps from these app markets in such areas.

In this thesis, we focus on mobile ecosystems powered by mobile platform owners. Platform owners are businesses that facilitate interactions between the different stakeholders of the mobile ecosystem to provide users with a specific service via their backend infrastructure [49]. Google Play is a platform that enable app developers to dissemination apps to users worldwide via their content delivery networks (CDNs). In contrast, the Unified Payments Interface released by the Indian government is a mobile payments platform that facilitate bank-to-bank transactions via their common backend interface via mobile payment apps integrated to their platform.

For platform owners, to design such large-scale mobile systems, typically involves the following four processes: identifying and defining stakeholders and their expectations, defining the technical requirements and threat models, developing system architecture by decomposing the requirements into system workflows, and finally, forming a design solution by (re-)iterating through a series of modeling and experimentation [79, 175]. While functional technical requirements are the system's actual use cases and are evaluated for correctness, the non-functional requirements are the optional yet desirable properties of the system that a user may expect. The CIA triads of confidentiality, integrity, and availability, are considered non-functional requirements of a software system [194]. Thus, while these are desirable features, a system may only provide some security guarantees, that largely depends on the stakeholders of the system, and the business requirements.

2.3 Potential Attack Vectors

To truly understand and security evaluate a mobile software ecosystem, it is imperative to understand the ecosystem as a whole, its vulnerabilities and the attack vectors that are available to an adversary. Compared to a traditional desktop ecosystem, mobile systems have a much larger attack surface. Below, we enumerate factors introduced by the various stakeholders that can impact security in the mobile ecosystem.

1. **Risks from app privileges**. Risks introduced by a mobile app varies by the privileges they acquire from a user when being installed on a mobile device or set by a developer when building an app. By virtue of these privileges a mobile app can access the many sensitive resources on a user's mobile device such as a device's keystore, camera, microphone, or GPS. These privileges also allow apps to interact with the other apps on a user's device thereby potentially allowing a user's data to be shared with the other apps. These privileges are set by the app developer, and may vary even amongst apps that provide similar functionality (e.g., chat apps). This is in stark contrast to how a desktop user typically accesses different

services mostly via a single client, i.e., a web browser. The permissions granted to a desktop web browser does not change with the service that is being accessed. Thus, depending on the privileges of an app, the attack surface will also vary.

2. **Long attack window.** Most the mobile apps are always logged into the service provider's backend system to enhance user experience by reducing the number of times a user has to login each time the app is opened. But this also means that these apps may have the ability to continuously collect and transmit a user's data under the hood. The conventional session timeouts that occur on websites are mostly not enforced when services are accessed via a mobile app. This gives a long attack window for an adversary to carry out an attack.

3. **Citizen developers.** The emergence of open app markets powered by Apple and Google has democratized app development—citizen developers can now release any number of mobile apps for consumer use. These citizen developers often lack knowledge of the best practices to develop apps and their services in a security savvy way. This also has led to an increase in malicious app developers.

4. **Competing app market proprietors.** Citizen developers can now distribute these through several app markets to bypass the costs incurred from global centralized app markets proprietors like Apple and Google. There are several local and third party app markets, for instance, those hosted by cellular network providers or independent publishers, that allow developers to distribute their apps with relative ease. That apart, this also allows malicious developers to distribute fake or trojanized versions of apps to users.

5. **Indequate app vetting.** Despite vetting apps for potentially malicious behaviors, malicious apps continue to plague global app markets like Google Play and Apple App Store, showing how citizen developers are able to bypass the security protections provided by these app markets. The bigger problem however is that many of the independent app markets may provide no vetting at all thus exposing users to a high level of fraud [43].

6. **Fragmented ecosystem.** The mobile hardware and software ecosystem as a whole is extremely fragmented with different versions of mobile devices, their operating systems, and app versions making it impossible to security vet against all combinations of software and hardware. Moreover, the firmware and software can now be updated over-the-air (OTA) by cell phone manufacturers and new app functionality side-loaded after installation on a mobile device. This opens an attack vector wherein, a malicious developer can alter an installed app, for instance, to contain malicious functionality.

7. **Vulnerabilities of cellular networks.** the mobile ecosystem grew on the top of existing cellular networks which by itself has been proven to have its own weaknesses making attacks feasible [88, 170, 239]. In traditional desktop systems, the cellular networks never comes into operation, and traffic passes over the IP network which has going through years of security vetting.

2.4 Understanding Android Apps for Reverse Engineering.

Android applications or *apps* are software programs that are packaged into an archive file called Android Package (APK) and runs on the Android platform. These software programs ends with the `.apk` extension. The contents of an APK file include - application code as .dex files, resources, assets and a manifest file. A DEX file consits of a set of instructions in Dalvik bytecode format. Android manifest is an XML file that describes permissions required by an app, package name of the application, Android components, etc. Resources of an app include icons, drawable files, strings, etc., and assets include files such as textures and gaming data that are compiled into the APK.

An Android app has four main components each with a specific purpose [99]. An *activity* is a single screen in an app that interfaces with the user. A *service* is a process that does not have a GUI and performs operations in the background. A *broadcast receiver* is a handler for broadcast messages from the system or other apps. A *content provider* is a repository used to manage access to data stored by itself or by other apps.

Unlike Java programs that have a *main* method as the entry point, an Android app has several entry points for execution. This makes reverse-engineering or statically analysing Android programs a significant challenge. Entry point of an application maybe: (i) the lifecycle methods of Android components are identified by *onCreate, onStartCommand* and *onReceive* respectively; (ii) event handler routines that handle registered application and system events (they are implicit entry points) (iii) Dangling methods that are possible call sites of dynamically dispatched methods or computed function invocations.

2.4.1 Reverse-engineering Barriers: Application Hardening Techniques

Hardening is a technique used to reduce the attack surface of an application. Typically, applications use several hardening techniques in tandem to provide a layered defense. This section summarizes app hardening techniques that are reverse-engineering barriers that we, from an adversarial standpoint, had to overcome or evade when studying mobile apps. Depending on the app, we had to overcome many of these barriers for our work. Most of these techniques are also known defenses in the literature [191], [77], [131], [236], [71], [192], though their implementations may vary based on the workflow of the application.

1. **Multi-factor User Authentication (MFA):** MFA is used to establish a users identity using something the user is, the user knows or the user has. Most apps that provide critical services such as payment apps, use atleast two factors of authentication. All the apps that we reverse engineered is secured at the very least using two factors, atleast at the time of initial registration. From an attacker perspective, an attacker has to compromise these factors of authentication to launch an attack.

2. **Root Detection:** Rooting is the process of attaining super-user privileges to the mobile operating system code, as well as apps on a device. Privileged applications on a rooted phone can access code and data of other apps installed on the device. The goal of root detection technique is to detect whether a phone is rooted phone or not. Certain apps fail to launch

itself or disable certain critical app features if it detects that the phone is rooted. Not all apps that we reverse engineered disabled itself on rooted phones, though most of them do notify the user that the device is rooted.

3. **Tamper Detection:** This is a hardening technique that developers employ to verify the run-time integrity of an application. One approach to check the integrity of an application is to verify whether the app's signature is valid. This approach is typically used to deter an attacker that has taken a legitimate app and modified it by inserting attacker code. Such an app is subsequently repackaged with an attacker's signature before it is distributed for public use. App's thus modified will have a signature different from the original. Verifying the integrity of an application at run-time can be used to detect if a user is using a repackaged version of an app or not.

4. **Use of Customized Keyboards:** Keyboard input sniffing is a common attack vector to steal user's sensitive information as they are entered by a user on an Android app. To prevent such an attack, developers sometimes use a custom keyboard (instead of Android's traditional keyboard) that has the ability to insert spurious inputs to obfuscate user's original data thus making it difficult to eavesdrop and interpret it. While this can be a security feature, custom keyboards also provide local language support.

5. **SSL Pinning:** Pinning is the process of embedding (or attaching) the server's SSL certificate into the app so that it overrides the certificates installed in the device's trusted store. This ensures that the app does not use any of the certificates on the device, which maybe a fraudulent certificate installed on the device by an adversary. Instead, the app relies only on the certificate attached to it. By passing this defense is rather tricky, for most the attacker may either have to be able to repackage an application after installing an attacker certificate to the app, or reverse engineer the SSL validation library if the developer is using their own SSL library, or use hooking libraries to intercept system calls to determine the handshake which requires a rooted phone and thus may not work on apps that are disabled in the presence

of a rooted phone. Finally, older versions of Android (version ¡6.0) allow apps to trust user-installed certificates. Thus in geographies where older versions of Android are still being used, SSL pinning can still be bypassed with relative ease. For this research, we have leveraged older Android devices to interept SSL communication.

6. **Disabling of Clipboard:** Android's copy and paste framework allows any application to read from or write to the device's clipboard. There are several apps such as password managers that use the clipboard to auto populate text fields. Copying clipboard requires no new permissions on an app, and hence an attacker, to eavesdrop, can copy the contents of a clipboard with relative ease. However, as a privacy enhancement Android 12 and later shows a toast notification when the contents of the clipboard are being copied [199].

7. **Restricting Screenshot:** Just like clipboards, taking a screenshop on an Android device required no additional permissions. To prevent screen capture attacks, the developer must treat open sensitive windows as secure using *FLAG_SECURE*. However, this can overriden by other apps on rooted phones.

8. **Session Timeout:** Unlike traditional web systems, a user that is signed into a mobile app, remains logged in indefinitely. While certain apps enforce session timeouts, most applications provide no such security to a user. This gives an adversary a large attack window and is a significant threat if a user's device is lost or stolen, since an attacker can gain access to all her open accounts on the device.

9. **Debugger Detection:** An attacker can attach a debugger to a running app to study the run-time behavior of an app and to inspect the values stored by an app's internal variables. To prevent such an attack, a developer may check for the presence of a debugger (by using the API call *isDebuggerconnected()*).

10. **Emulator Detection:** An Android Emulator simulates Android devices on your computer for a developer to be able to test their application on different devices and Android API

levels without having to own each physical device. However, an attacker can also leverage an emulator to reverse-engineer the run-time behavior of an application. Thus detecting the presence of an emulated environment is can help prevent the run-time analysis of an app and also establishes the fact that the user running the app is not a real user of the system. However, an attacker can modify the emulator to return valid device identifiers to fool an application.

11. **User Account Lockout:** To prevent brute force attacks on a user's account, the account must be locked out after a stipulated number of failed login attempts.

12. **No. of Active Users:** Certain apps allow more than one user to sign-in from the same device, or allow the same user to sign-in from different devices. While this is a useful feature, this allows an attacker to also attempt to compromise a user's account from his personal device. Some apps that we reverse engineered, provided a lower bar of authentication when an existing user installs the app on a new device. Alternatively, some apps also do not track a user's sign-in activity over time, to report a suspicious sign-in. These holes provide an opportunity to an attacker trying to compromise an existing user.

13. **Detection of Reverse Engineering Tools:** For an attacker having access to a rooted device, the attacker may also have the ability to modify system calls to mask the fact that the device is rooted. In such a case, attacker may leverage hooking tools like Frida or Xposed. To prevent such a case, a developer may also look for the presence of reverse-engineering tools and their common install locations on a device prior to loading the app.

14. **Code Obfuscation:** Developers often use static code obfuscation techniques such as lexical obfuscation and control flow obfuscation to hide program logic and to evade automated static analyzers. This is the de facto for commercial applications today and there are several tools like Proguard (which is part of Android SDK) that allows such obfuscation. While lexical obfuscators make the code unreadable, control flow obfuscation has the ability to deter code decompilation. This is a significant reverse-engineering barrier as now understanding the app's logical will involve reading native bytecode.

15. **Old App Versions/ Security Updates:** Developers may regularly provide security updates to apps as a means to secure the application and keep it upto date. There users may typically not update the application. But developers rarely enforce the users to update the application for usability reasons. Similarly, third-party app markets like APKPure provide several older versions of applications. An attacker can leverage either an old version of the application to either reverse-engineer or carry out an attack.

It is imperative to note here that for an application that can be repackaged, most of the client-side defenses can be disabled by an attacker. This approach has been leveraged to carry out some of the attacks in this research. The security mechanisms that are harder to bypass or reverse-engineer are those provided by the backend server of by a *Trusted Execution Environment (TEE)*, which is a computing environment that allows an application to execute and store sensitive data using hardware-backed attestation mechanisms.

2.5 Static Analyzer Tool

Motivation. Reverse-engineering binaries of apps is a non-trivial task. There are two ways to do this: (1) to decompile an app to get its high-level source code and then examine the app's converted source code or (ii) to use an existing static analyzer tool to facilitate automated analysis of app binaries. But practically, both approaches have limitations that deter analysis, and examining an app's binary involves combining a multitude of techniques, of which decompiling and static analysis is a significant conponent. Run-time examination of apps is infeasible in most cases, especially since most of the highly-rated apps we examine have defenses that deter runtime analysis. Also, these apps may only work in specific geolocations or their behavior may change when accessed from outside the purview of the intended geolocation.

Most off-the-shelf, open-source static analyzers have several limitations that prevent in-depth app binary examination:

1. The reports they generate furnish first-level information easily accessible after an app's

disassembly from its manifest file or config files.

2. They do not provide details about strings or constants passed as parameters or the calling hierarchy from the perspective of multiple entry points in the app.

3. They do not enumerate the classes and functions available in its split apps or third-party libraries, which are bundled to form the complete app binary.

The available decompilers also failed to convert the bytecode to high-level language when an app's binary has been obfuscated with complex control flow obfuscation techniques.

To offload some of the complexities that came with manual reverse-engineering of these apps, we developed a static analyzer that: (i) identified all the entry points of an application and constructed a call graph and a control flow graph starting with each of its entry points (ii) semantically identified a limited set of hardcoded constants in the app, and (iii) conducted a static data flow analysis to determine a set of constant values at specific program points, given an app's binary.

Types of Constants. The analyzer is designed to look for three types of hardcoded constants- a) *Simple constants* such as string literals or numbers a program uses directly b) *Derived constants* obtained by modifying other constant data types— e.g., an integer converted to a string, and c) *Dynamically composed* constants constructed by concatenating other constants—e.g., a string composed of several substrings. The analyzer prototype recognizes primitive data types such as `int`, `String`, `float` and `double`. It also models a limited set of objects that enable string manipulation operations such as `StringBuilder` or `StringBuffer`, arithmetic operations such as *Integer*, *Float* and *Double* and conversion from numeric to string data types (e.g, `Integer.parseInt`, `Integer.valueOf`).

Program Representation. The program is represented as a call graph, with the nodes of the graph being a function and the edges indicating the control flow across these functions. Each function is further represented as a control flow graph, with each node being a bytecode instruction and the edges being the control flow hierarchy between them. We do not compute basic blocks of a function to eliminate the overhead involved in doing this extra step. Our experience with tools such as Soot indicates that computing basic blocks, especially for control flow obfuscated programs, may result in the static analyzer going into an infinite loop. So we choose to represent the nodes of a control flow as an instruction instead of a basic block. The analyzer uses an app's call graph to perform interprocedural analysis and the call graph for intraprocedural analysis. The tool constructs a context-insensitive call graph per entry point. The drawback of a context-insensitive call graph is that it cannot accurately model the set of possible callees of dynamically dispatched methods and computed function invocations [116]. However, practically, a context-sensitive call graph is not scalable for large programs [227]. This is a design trade-off we make in the tool.

Given how Android apps are event-driven programs, an app will have multiple entry points. The analyzer looks for three types of entry points: (i) the lifecycle methods of Android's exported components (ii) event handler routines that handle registered application and system events (they are implicit entry points) (iii) dangling methods that are possible call sites of dynamically dispatched methods or computed function invocations. (Dynamic dispatch is a also commonly applied control flow obfuscation technique.)

Unlike conventional approaches where a program is represented as a single call graph, we model the program as multiple call graphs in keeping with the loosely coupled and asynchronous nature of Android apps. Since it is impossible to pre-determine the order of execution of Android components or events, there is no loss of information in representing them as independent call graphs analyzed in an arbitrary sequence. Thus the call graphs modeled by our tool are selectively flow-sensitive. Unlike [30] we choose to keep the call graphs independent (and not merged into a dummy main), in keeping with the asynchronous and loosely coupled nature of Android programs.

The control flow graph of a program includes all possible execution paths of the application. The analyzer may thus traverse spurious paths introduced by obfuscated code. Intraprocedural analysis is thus flow-sensitive since it considers the order of statements of the program and path-sensitive since it traverses every path guarded by a predicate to reason about the program state. The analyzer can trace into third-party libraries integrated with the application. This is particularly useful since

UPI-enabled apps integrate with UPI's backend infrastrucutre via the libraries released by NPCI.

To determine hardcoded constants, we examine a sequence of bytecode instructions that may indicate a constant *definition* followed by *use* of the constants. The general idea is that a variable's *def* is an instruction that causes a memory write (stack push in the JVM). In contrast, the *use* of a variable is an instruction that causes a memory read (stack pop in the JVM). For instance, the byte code instruction *ldc* and *sipush* are instructions that directly load a constant onto JVM's operand stack. Thus the presence of one of the *load direct* instructions indicate a hardcoded constant. Such instructions are only meaningful if it is coupled with an instruction that use it. For instance, a high-level languge statement such as `System.out.println("xxx.com")` in the bytecode appears as follows:

```
0: ldc #2 // String xxx.com
2: invokevirtual #4 //Method java/io/PrintStream.println:
                        ( Ljava/lang/String;)V
```

Here, `ldc` (a load instruction) immediately followed by an `invoke` statement indicates the use of a hardcoded variable as a function parameter. Whether one of the 5 types of function invocations *follows* or *immediately follows* depends on the arity of the function. For instance, `Socket localSock = new Socket(server,2010);` shows another use of the hardcoded string. The bytecode for this statement is as follows:

```
    100: new  \#2  // class java/net/Socket
    103: dup
    104: aload_1
    106: sipush  2010
    109: invokespecial \#4 // Method java/net/Socket."<init>":
                                  (Ljava/lang/String;I)V
```

The invoke instruction here, invokespecial is preceded by two load instructions that loads the parameters for the Socket API call.

Implementation. The static analyzer is a bytecode analysis framework written in Java using *Java ASM* to perform bytecode manipulation and analysis [238]. The input to the analyzer is an Android APK binary. Also input is a specifications file describing security-sensitive Android framework (API) calls that identify program points that have security implications. An Android APK is retargeted as a Jar file for analysis.

The static analyzer uses a two-pass algorithm. In the first pass, the analyzer identifies the app's entry points and constructs a call graph for each entry point. We construct a call graph using a class hierarchy analysis (CHA) approach that can conservatively handle dynamically dispatched methods [65, 245]. The initial set of entry points is the lifecycle methods (e.g.*onCreate*) of Android components obtained from *AndroidManifest.xml*. Once the algorithm has constructed a calling hierarchy, methods not belonging to any call graph are maintained in a separate list, as they may only be entered when an event is triggered.

In the second pass, the analyzer performs data flow analysis starting with a node containing any Android framework calls that interest us. The analyzer tries to infer the history of uses and defs of sensitive parameters of interest (e.g., domain in a socket call). The analyzer backtracks to its predecessor node if unable to determine the constant value of the parameter or establish that the parameter is not a constant. If it finds a constant value, it is forward propagated forward to the program point of interest.

The static analyzer mimics the JVMs stack-based execution model and internally uses a local variable table and operand stack to construct constants. The analysis output is a *JSON* file that provides a detailed report on the Android framework calls, their execution path, and the set of hardcoded constants bound to parameters of interest. For verifying strings present in the APK, the analyzer implements the equivalent of the Linux *strings* command. It also logs a list of all Android framework methods in each class and analyzes third-party libraries barring Android runtime libraries. We leverage this analyzer for all our analysis of apps.

2.6 Prior Large-scale Studies

Prior large scale studies on the mobile app ecosystem has extensively looked at apps on Google Play store and Tencent to understand app update behavior, frequency and timing delay, app auditing processes, transparency efforts, impact of app releases, download distribution of apps, app monetization schemes, behavior of app developers, third-party tracking threats and the incidence of malicious apps in these app markets by longitudinally studying apps from a region over a period of time [139, 162, 163, 180, 197, 235, 257, 261, 263, 271]. Lim et al. [151] conducted a large-scale survey on mobile app user behavior that affects app market downloads. These studies are confined to app markets within one region. There is also a geographically diverse study of popular mobile money apps that check for vulnerabilities in these applications. But the study in this case does not involve a central platform owner that is common to these apps, neither can these apps inter-operate for bank-to-bank transactions.

Studies on privacy policies of mobile apps have been carried in much smaller scope for the challenges entailed in collecting these policies A limited study of privacy policies of select Chinese apps as seen from China and in Canada has shown how policies of these apps in the overseas market differs from its policies in the Chinese market. Prior research has also looked at finding privacy policy violations by analyzing policy texts and apps [10, 11, 269], characterizing and querying privacy policies [119], and studying differences in the policies of free and paid apps [117]. Longitudinal studies of web privacy policies show that longer policies are slow to comply with recent legislation such as the GDPR [9, 66, 153].While these studies are all useful to determine developer (and app) behavior for a region, it does not give a clear

Prior research has also conducted large scale security analysis of web ecosystems to determine their overall security vulnerabilities and the defense mechanisms they adopt, analysis and detection of cross-site request forgery, shellshock vulnerability, update behavior of websites and its impact on security, comparison of security of desktop and mobile first websites, longitudinal trends of the adoption of client-side security mechanisms in european websites, [50, 53, 68, 69, 229, 254, 255]. A large scale study on desktop computers and laptops seeks to establish the prevalence of popular security practices and their relationship to security outcomes via a longitudinal empirical measurement of computer devices [67].

CHAPTER 3

Security Analysis of a Regional Mobile Payments Ecosystem

3.1 Introduction

Payment apps have become a mainstream payment instrument in India, with the Indian Government actively encouraging its citizens to use electronic payment methods after a demonetization of large currency notes in 2016 [138]. To facilitate digital micro-payments at scale, the National Payments Corporation of India (NPCI), a consortium of Indian banks, introduced the Unified Payment Interface (UPI) to enable free and instant money transfers between bank accounts of different users. As of July 2019, the value of UPI transactions has reached about \$21 billion [188]. UPI's open backend architecture that enables easy integration and interoperability of new payment apps is a significant enabler. Currently, there are about 88 UPI payment apps and over 140 banks that enable transactions with those apps via UPI [181, 182]. This paper focuses on vulnerabilities in the design of UPI and UPI's usage by payment apps.

We note that hackers are highly motivated when it comes to money, so uncovering any design vulnerabilities in payment systems and addressing them is crucial. For instance, a recent survey states a 37% increase in financial fraud and identity theft in 2019 in India [59]. Social engineering attacks to extract sensitive information such as one-time passcodes and bank account numbers are common [83, 126, 155, 244, 246].

Payment apps, including Indian payment apps, have been analyzed before, with vulnerabilities discovered [47, 209], and an Indian mobile banking service was found to have PIN recovery flaws [193]. However, in these studies, mobile apps did not share a common payment interface. As far as we are aware, an analysis of a common interface used by multiple payment apps has not been done before. Such an analysis is important because security flaws in them can impact customers of multiple banks and multiple apps, regardless of other stronger security features used. We focus on the security analysis of the unified payment interface used by many Indian payment apps and its design choices.

In this work, we use a principled approach to analyze UPI 1.0, overcoming significant challenges. A key challenge is that the protocol details are not available, though millions of users in India use it. We also did not have access to the UPI servers. We thus had to reverse-engineer the UPI protocol through the UPI apps that used it and had to bypass various security defenses of each app, including code obfuscation and anti-emulation techniques. Though we build on techniques used in the past for security analysis of apps [47, 91, 191, 209], our approach to extract the protocol details varies based on the defenses the apps use. We carefully examine each stage of the UPI protocol to uncover the credentials required to progress in each stage, find alternate workflows for authentication, and discover leakage of user-specific attributes that could be useful at a later stage.

We present results from the analysis of the UPI protocol, as seen by seven of the most popular (highly-rated) UPI apps in India listed in Table 1. Of the seven apps we analyze, four UPI apps—Google Pay (Tez), PhonePe, Paytm, and BHIM—have a combined market share of 88% [134] and are widely accepted at many shopping sites. From a total of 88 UPI apps, many are minor variations of BHIM, the flagship app released by NPCI (also the designers of UPI). Close to 48 banks today issue a bank-branded version of the BHIM app. Since Android owns over 90% of the Indian mobile market share [60], we focused on the Android versions of these apps.

Our threat model assumes that the user is careful to use an authorized payment app on a non-rooted Android phone, but has installed an attacker-controlled app with commonly used permissions. We also do not rely on the success of social engineering attacks, though they could simplify exploiting some of the vulnerabilities we uncovered. We uncovered several design choices in the

App Name	Launched	App Versions	Installs	Rating	UPI Version
BHIM	Dec, 2016	1.3, 1.4, 1.5	10M+	4.1	1.0
Ola Money	Nov, 2015	1.8.1, 1.8.2, 1.9.0	1M+	3.8	1.0
Phonepe	Dec, 2015	3.0.6, 3.3.23	100M+	4.5	1.0
SamsungPay	Aug, 2016	2.8.49, 2.9.3	50M+	4.7	1.0
Paytm	Aug, 2010	8.2.12	100M+	4.4	2.0
Google Pay (Tez)	Sept, 2017	39.0.001	100M+	4.4	2.0
Amazon Pay[1]	Feb 2019	18.15.2			2.0

[1]Amazon Pay is not available on Google Play store

Table 3.1: **Apps and their Google Play Ratings.** The table shows the list of apps analyzed in this work, their first launch dates on Google Play store, their popularity in terms of the number of installs and their app store rating, and the UPI protocol version at the time of analysis.

UPI 1.0 protocol that lead to the possibility of the following types of attacks:

- *Attack #1: Unauthorized registration, given a user's cell number:* This attack leaks private data such as the set of banks where a user has bank accounts and the bank account numbers.
- *Attack #2: Unauthorized transactions on bank accounts given a user's cell number and partial debit card number:* Purchases using a debit card in India, whether in-store or online, requires a user to authorize the payment by entering a secret PIN. In this attack, an attacker, by knowing a user's cell number and debit card information printed on the card (last six digits and expiry date, without the PIN), can do transactions on a bank account of a user who has never used a UPI app for payments.
- *Attack #3: Unauthorized transactions without debit card numbers:* This attack shows how an attacker that starts out with no knowledge of a user's authentication factors can learn all the factors to do unauthorized transactions on that user's bank account.

Our work started over two years ago when NPCI released UPI 1.0 and BHIM, which are the focus of our analysis. Given the potential risks with releasing our findings, we waited to publish until NPCI addressed a critical attack vector in the recently released UPI 2.0. Our key contributions are as follows.

- We conduct the first in-depth security analysis of the unpublished UPI 1.0 protocol that provides a common payment interface to many popular mobile payment apps in India and allows bank-to-bank transfers between users of different apps.

- We show how to systematically reverse-engineer this complex application layer protocol from the point-of-view of an adversary with no access to UPI servers. We use BHIM, the reference implementation for UPI apps released by the Indian government, for our initial analysis and then confirm our findings on other UPI apps.
- We found subtle design flaws in the UPI protocol, which can be exploited by an adversary using an attacker-controlled app that leverages known flaws in Android's design, to construct scalable remote attacks. We show how an adversary can carry out the attacks starting with no knowledge of a user.
- As responsible disclosure, we reported the flaws to app developers, CERT India, and CERT US, resulting in several CVEs. A key attack vector we reported to NPCI and CERT India was addressed in UPI 2.0.
- We present early findings from an ongoing analysis of UPI 2.0, using BHIM, Google Pay, Amazon Pay, and PayTM—four top-rated UPI 2.0 apps in India. Findings indicate that some vulnerabilities remain.
- We discuss lessons learned and potential mitigation strategies to consider when designing such protocols.

3.2 Background

Early mobile payment apps in India were wallet-only apps. They could withdraw money from a user's bank account by asking a user to enter a debit card number, but not deposit money back into the bank account. Post demonetization (in 2016), to encourage cashless transactions, a consortium of Indian banks called the National Payments Corporation of India (NPCI), backed by the Indian government, introduced the Unified Payments Interface (UPI) that allows NPCI-certified mobile

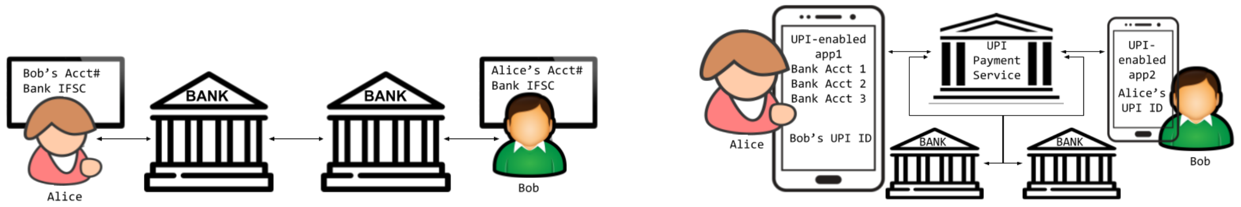


Figure 3.1: **Internet vs. UPI-based Money Transfer.** On the left is shown traditional Internet banking vs. UPI money transfer system on the right. UPI transactions requests from a mobile device is routed to a centralized UPI backend infrastructure, which then initiates a transaction between the two banks. In contrast, in the traditional Internet banking system, the transaction request is sent directly to the origin bank.

apps to do free instant money transfers between bank accounts of different users. UPI apps can inter-operate with each other since they all share the same payment interface. A user of BHIM, for instance, can transfer money instantly for a small purchase from her bank account to the bank account of a shopkeeper who uses Google Pay. Because of this, most stores in India accept mobile payments through UPI apps. Depending on the app, a user can do unlimited transactions up to $1500 per transaction. Figure 3.1 shows the UPI money transfer system (on the right) when compared with the traditional Internet banking system on the left.

3.2.1 User Registration on a UPI App

The UPI payment system requires Alice to register her primary cellphone (or cell) number with her bank account(s) out-of-band to send or receive money. UPI uses the cell number (i) as a proxy for a user's digital identity with the bank to look up a bank account given a cell number; (ii) as a factor in authentication via SMS one-time passcodes (OTP); and (iii) to alert users on transactions. The Government of India requires cellphone providers to get copies of government-issued IDs, manually verify the IDs, and do biometric verification before issuing a cell number [1].

To register for UPI services, Alice must set up her UPI user profile, add a bank account, and enable transactions on that bank account, as follows:

1. *Set up a UPI user profile:* Alice must first create a profile with UPI via a UPI app installed on

[1]A recent Indian Supreme Court ruling forbids Aadhar's biometric verification for issuing cell numbers. The impact of that ruling on UPI-based apps and banks is yet to be seen, as it may make it easier for an attacker to do an unauthorized transfer of a cell number and then take over an account. We do not discuss this attack vector in this paper.

her bank-registered cell phone. Alice must first give her cell number to UPI through the UPI app for verification. How UPI collects this information from a user may change with each app. For instance, some apps read the cell number from the device, while others ask the user to key-in the cell number. For instance, Figure 3.2, screenshot #3, shows how BHIM reads Alice's cell number(s) from her phone for Alice to choose from. The UPI app then sends Alice's cell number to the UPI server for verification. Once verified, the UPI server issues a UPI ID for Alice on that app. Figure 3.2, screenshot #4 shows how BHIM notifies Alice when she is verified. If Alice uses multiple apps, the UPI server issues a different UPI ID for each app. The app then prompts Alice to set a passcode. The nature of the passcode is again specific to the app. BHIM, for instance, asks the user to set a 4-digit passcode, as shown in Figure 3.2, screenshot #5.

2. *Add a bank account:* Once Alice's profile is set up, she must add the bank account that she wants to use for withdrawals and deposits. Alice is given a list of bank names that support UPI (Figure 3.2, screenshot #6), from which she can now choose her bank. Alice may repeat this step to add multiple bank accounts.

3. *Enable transactions:* For Alice to be able to transact on an added bank account, she has to set up a UPI PIN for that account before the first transaction. The UPI PIN is Alice's secret to authorize any future transactions. To set the UPI PIN, Alice must furnish information printed on the debit card— the last six digits of her bank's ATM or debit card number and expiration date. Alice must also enter an OTP she receives from the UPI server. The UPI PIN is a highly sensitive factor since the UPI server uses it to prevent unauthorized transactions on Alice's bank account.

To transfer money to Bob, Alice first logs into a UPI app using the passcode she set during user registration. Then, out-of-band, Alice requests Bob to provide his UPI ID, which is often Bob's cell number. Alice chooses one of the bank accounts she previously added to the app (Figure 3.2, screenshot #7), initiates the transaction to Bob, and authorizes it by providing her

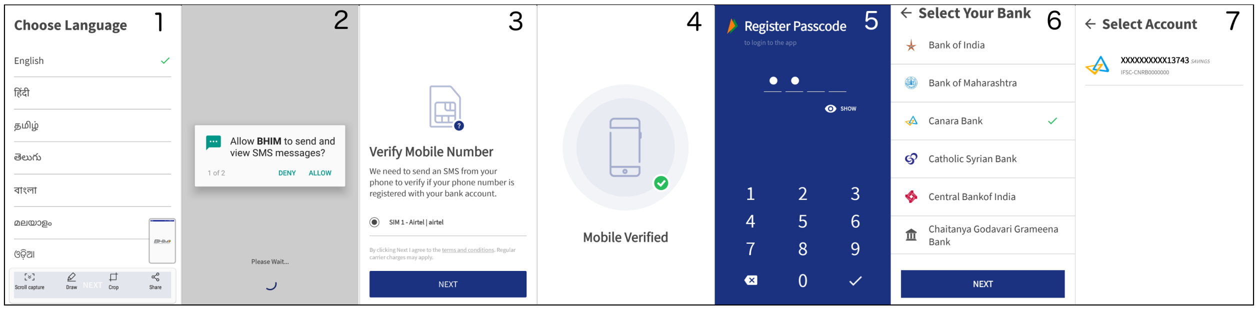


Figure 3.2: **BHIM User Registration Using 3FA.** The figure shows the user interface screens corresponding to the different steps in the user registration process as seen by a user of BHIM app. The registration process begins in (1) with the user choosing a language. The user is then presented with a consent screen (2) that requests permission to send and receive SMS permissions. In (3) the user is asked to verify her mobile number the confirmation for which is shown in (4). The user sets up a password (in 5), after which a list of banks is presented from where the user chooses her bank (6). The bank account of the user is identified in (7).

UPI PIN. Internally, the UPI payment interface directly transfers money from Alice's chosen bank account to Bob's bank account linked with his UPI ID.

3.2.2 UPI Specs for User Registration

The UPI specifications released by NPCI [187] provide ”broad guidelines” on the client-server handshake between a UPI app and the UPI server. We discuss the protocol details available to us from the specification.

1. *Set up a UPI user profile:* Once a UPI app gets a user's cell number, the app must send an outbound encrypted SMS from Alice's phone to the UPI server. This process is automated and does not involve the user in order to guarantee a strong association between a user's cell phone and her device. According to UPI, this is the “most critical security requirement” of the protocol since all money transactions from a user's device are first verified based on this association. UPI calls this association of a user's device (identified by parameters such as Device ID, App ID, and IMEI number) with her cell number as *device hard-binding*. The combined cell number and device information (that represents this binding) is called the *device fingerprint*, which per the UPI spec is the first factor of authentication.

Passcode. The UPI spec considers application passcode as optional and does not undertake responsibility for passcode authentication. UPI leaves it up to a UPI app vendor to authenticate the passcode. Thus, the responsibility to completely authenticate a user is shared between two servers— the UPI server (that verifies device fingerprint and UPI PIN), and a payment app server (that verifies an app passcode).

2. *Add a bank account:* A user's request to add a bank must be from the device registered with UPI. Internally, UPI fetches the chosen bank's account number and IFSC code based on a user's cell number for later transactions through the UPI app.

3. *Enable transactions:* UPI allows transactions to be done either using a cell number or an account number and IFSC code or any UPI ID. UPI spec mandates that all transactions must at least be 2FA using a cell phone (the device fingerprint) as one factor and the UPI PIN as the second. The spec considers a cell phone as a "what you have" factor, which allows UPI to provide "1-click 2-Factor Authentication" using the said two factors.

For apps that integrate with UPI, NPCI enforces application security via a code review and certification process. All communication with the UPI server is over a PKI-based encrypted connection. Currently, UPI has become the de facto standard for mobile transactions.

3.2.3 Threat Model

We assume a normal user, Alice, who installs payment apps from official sources such as Google Play; none of the payment apps contain extraneous malicious code. Alice has a properly configured phone with Internet facility and prevents physical access to it by untrusted parties.

On the other hand, the attacker, Eve, uses a rooted phone. Eve can use any tool at her disposal to reverse engineer the payment apps. We assume that Eve releases an apparently useful unprivileged app called *Mally* that requests the following two permissions—android.permission.INTERNET and android.permission.RECEIVE_SMS. Alice finds the app useful and installs it, granting it the necessary permissions.

The permissions requested for Mally are not unusual for Android. Recent versions of Android automatically grant the INTERNET permission without a user prompt [70]. SMS permissions have legitimate uses on Android, and about 15% of the Android apps request them [90]. RECEIVE_SMS permission only grants the permission to read incoming SMS messages, but not read previously received messages or send SMS messages. This permission is used by many popular social media apps such as Telegram and WhatsApp, SMS/call blocker apps, and also security apps such as Kaspersky Mobile Security and BitDefender.

We consider our threat model to be realistic for the following reasons. First, according to the Android security review for the last two years, India is among the top three countries with the highest rate of potentially harmful applications such as trojans and backdoors, sometimes pre-installed on Android devices [129, 130]. Google has also recently released a warning stating that 53% of the major attacks are because of malicious apps that come pre-installed on low-cost smartphones [86].

To simplify some attack descriptions, we describe Mally with the READ_PHONE_STATE or accessibility permissions. We do this to show the many ways an adversary can get a user's information, e.g., a user's cell number. However, in such cases, we also show other attack vectors that require neither of these two permissions.

3.3 Security Analysis

3.3.1 Methodology

In this section, we describe how we reverse-engineer UPI, a proprietary protocol, to learn its authentication handshake. Since we do not have access to UPI's servers, we choose to reverse engineer this application layer protocol through the payment apps that support it. We first reverse-engineer the apps to determine the hardening techniques used by these apps. Table 3.2 shows a few of the techniques we looked for in UPI 1.0 apps and our findings. Surprisingly, we found that these apps' defenses varied, which we exploited to reverse-engineer UPI. For instance, while PhonePe and SamsungPay could not be reverse-engineered by repackaging the application, we found that

BHIM and Ola could be repackaged, which we leveraged in our study. Similarly, on PhonePe, we leveraged the fact that the app did not use pinned SSL certificates and hence used a rooted phone to install our custom certificate and used that for traffic interception. Thus, we leaked UPI's proprietary protocol by exploiting these app differences and further confirmed our findings on other apps.

3.3.1.1 Protocol Analysis.

To reverse-engineer UPI, we first uncover each step of the client-server authentication handshake with the goal of (i) understanding how UPI does device fingerprinting; and (ii) establishing the credentials required by a user to set up an account and do transactions. Besides UPI's default authentication workflow, we also look for alternate workflows or paths that could be leveraged to minimize the credentials required by an attacker. Finally, we look for any leaked user-specific attributes during protocol interactions that could be leveraged later, if intercepted, by an adversary. We triage our findings from different workflows to find plausible attack vectors and to verify potential exploits.

The approach we use to extract protocol data varies based on the specifications of an app and the security defenses they use. Since UPI 1.0 specs only state broad security guidelines rather than protocol details, we examine multiple apps to know whether the protocol varies across different apps. We analyze BHIM, the flagship app published by the same government organization that maintains the UPI system and then confirm our findings by analyzing additional apps.

3.3.1.2 App Reversing-Engineering.

One approach to capture the protocol data sent and received by an app is to run it in a sandbox. Sandbox tools such as CuckooDroid [61] use an emulator for dynamic analysis. Hence, to test if the UPI apps can run in a sandbox, we manually run each app in Android SDK's built-in emulator on a Linux host. However, we find that these apps do not run without a physical SIM card, which is unavailable on an emulator. The apps also use anti-emulation techniques that prevent them from running in an emulator.

Hardening Techniques	Ola Money	Samsung Pay	BHIM	PhonePe
3FA	✓	✓	✓	✓
Root Detection	✗	✓	✓	✗
Tamper Detection	✗	✓	✗	✓
Session Timeout	✗	✗	✓	✗
Debugger Detection	✓	✓	✓	✓
Restricting Screenshot	✗	✓	✗	✗
Disabling Clipboard	✗	✓	✗	✗
SSL Pinning	✗	✓	✓	✗
Emulator Detection	✗	✓	✓	✓
No. of Active Users	✗	✓	✗	✗
Mandatory Security Updates	✗	✓	✓	✗
Use of Customize Keyboads	✗	✓	✓	✗
User Account Lockout	✓	✓	✓	✓
Detection of RE Tools	✗	✗	✗	✗
Obfuscation	✓	✓	✓	✓

Table 3.2: **Hardening Technique in UPI 1.0 Apps.** The table shows a few of the hardening techniques we found in our manual analysis of UPI1.0 that we analyzed. All apps used multi-factor authentication and used obfuscation techniques to protect their apps. However, these apps varied in the security hardening techniques they used. We exploited these differences to leak the workflow of UPI through these apps as the vantage point. A ✓ indicates that the app uses the security defense, and ✗ indicates that the app does not employ the defense.

Besides anti-emulation, we find that the payment apps also use several other defenses. For instance, all of them detect a rooted phone and deter a user from running the app on a rooted phone. Some apps also look for the presence of hooking libraries such as *Xposed* [136] that typically require root access to modify system files. That apart, all apps are obfuscated, use encrypted communication, enforce session timeout and account lockout, avoid storing or transmitting data in the clear, and avoid using hard-coded credentials or keys. The extent of security defenses used by these apps shows that app developers have designed the apps with security in mind. This is unlike findings by Reaves et al. [209] that found basic security flaws in Indian payment apps around 2015.

Our security assessments show that some apps, such as BHIM, allow repackaging. We leverage this to instrument an app's code statically to learn specifics of the authentication handshake, such as the name of the activity and method that generated network traffic. Because such specifics help with precise analysis, we first check whether the apps can be instrumented and repackaged. To instrument the app, we first disassemble it using *APKTool* [24], insert debug statements, and then

CWE	Name	Ola Money	Samsung Pay	BHIM	Phonepe
CWE-359	Exposure of Private Information	✗	✓	✓	✓
CWE-926	Improper Export of Android Application Components	✓	✓	✗	✗
CWE-295	Improper Certificate Validation	✓	✗	✗	✓
CWE-287	Improper Authentication	✓	✗	✓	✓
CWE-940	Improper Verification of a Source of a Communication Channel	✓	✗	✓	✓
CWE-282	Improper Ownership Management	✓	✗	✓	✗
CWE-288	Authentication Bypass Using an Alternate Path	✓	✗	✓	✗
CWE-841	Improper Enforcement of Behavioral Workflow	✓	✗	✓	✓

Table 3.3: **Summary Of Weaknesses.** This table shows the list of common weaknesses we found in the UPI1.0 apps we analyzed that facilitated leaking UPI handshake. For instance, the fact that some of these apps did not provide certificate validation, enabled us to capture traffic from these applicaitons. That apart, these apps allowed sensitive data to be leaked in some form or the other.

repackage it with our signature.

One question that arises is where to instrument in an app's code as this requires knowledge of the methods of the app we want to instrument. Since we do not know this *a priori*, we manually reverse-engineer the apps using the *JEB* [140] disassembler and decompiler. Some times, JEB fails to decompile certain classes that are control-flow obfuscated. In such cases, we use JDK's *javap* command to read bytecode. We augment our analysis with results from the static components of two hybrid analyzers *MobSF* [91] and *Drozer* [133].

We could not repackage certain apps such as Google Pay. In such cases, we intercept an app's network traffic using a TLS man-in-the-middle proxy called *mitmproxy* [167]. We install the OpenVPN app on our Android phone and an OpenVPN service on a Linux host and configure the host's firewall rules to route traffic to the *mitmproxy*. The setup also requires that we install *mitmproxy's* certificate on the phone. However, we find that starting Android Nougat, Android does not trust user-installed certificates, and setting up a system certificate requires root access, an impediment. Hence we conduct our analysis on Android Marshmallow and Lollipop devices.

Listing 3.1: BHIM code snippet

```
.class public Lin/org/npci/upiapp/utils/RestClient;
.super Ljava/lang/Object;
.source "RestClient.java"
# annotations
.annotation system Ldalvik/annotation/MemberClasses;
    value = {
        Lin/org/npci/upiapp/utils/RestClient$UnsuccessfulRestCall;
    }
.end annotation
# static fields
.field private static final a:Ljava/lang/String;
.field private static b:Lorg/apache/http/impl/client/DefaultHttpClient;
.field private static c:Lorg/apache/http/impl/client/DefaultHttpClient;
.method public static a(Landroid/content/Context;Ljava/lang/String;Ljava/util/ Map;)Lin/org/npci/upiapp/models/
    ↪ ApiResponse;)
    .locals 6
    .annotation system Ldalvik/annotation/Signature;
        value = {"(", "Landroid/content/Context;", "Ljava/lang/String;", "Ljava/util/Map", "<",
        "Ljava/lang/String;", "Ljava/lang/String;", ">;)", "Lin/org/npci/upiapp/models/ApiResponse;"
        }
    .end annotation
    .prologue
    const/16 v5, 0x130
    .line 404
    new−instance v2, Lorg/apache/http/client/methods/HttpGet;
    invoke−direct {v2}, Lorg/apache/http/client/methods/HttpGet;−><init>()V
    .line 405
    ...
    move−result−object v2
    const−string v3, "␣.␣Response␣Code:␣␣"
    invoke−virtual {v2, v3}, Ljava/lang/StringBuilder;−>append(Ljava/lang/String;)Ljava/lang/StringBuilder;
    move−result−object v2
    invoke−interface {v0}, Lorg/apache/http/HttpResponse;−>getStatusLine()Lorg/apache/http/StatusLine;
    move−result−object v0
    invoke−interface {v0}, Lorg/apache/http/StatusLine;−>getStatusCode()I
    move−result v0
    ...
.end method
```

3.3.1.3 App Instrumentation Example

We provide a brief discussion of one example instrumentation of BHIM's code with the goal of determining the workflow of the UPI protocol. BHIM version 1.3 consists of about 516K lines of obfuscated smali code. Some apps such as Paytm are even larger than BHIM, posing a significant reverse engineering challenge.

After searching through the BHIM code, we located the snippet below that belongs to the NPCI library and is integrated with the BHIM app. We found that NPCI had not obfuscated the name of the package as shown in line #1 *in/org/npci/upiapp/utils*. However, the method names are obfuscated as indicated by the method name at line #19 called *a*. The third-party libraries used by NPCI are not obfuscated as is seen by the class *org.apache.http.impl.client.DefaultHttpClient* at line #17.

We instrumented different portions of the BHIM app to determine the control-flow of the program. We found that when using automated tools such as Soot [221] to instrument the app, we got unexpected failures such as the app hanging indefinitely (we did get Soot to work for smaller test programs). We were unable to root-cause why BHIM's instrumentation with Soot did not work. Hence, we resorted to a careful smali code instrumentation of BHIM.

Listing 3.1 shows the method that performs HTTP GET. Since the methods are all *static* methods, by Android (and Java) convention, the first parameter is stored in the register *p0*, the second in register *p1* etc. The registers *v0*, *v1* etc. are registers local to a method body. Listing 3.2 contains code that prints the parameters to the GET request contained in the parameter *p1*. We inserted the code in Listing 3.2 after line #38, right at the beginning of the function (after the function prologue at line #35). The inserted code snippet prints the parameters using the *System.out.print* API call. The printed debug statements appear in Android *logcat* logs. We did a similar instrumentation for HTTP POST methods.

Some of the apps such as Paytm, that contain several DEX files (with each DEX file containing a maximum of 65536 methods), were even more challenging to instrument, as they obfuscate the calls to most of the third-party libraries they use. In such cases, further experimentation and analysis was required to discover the calls. That apart, the security defenses used by these apps may also change across app revisions. For instance, while older versions of Paytm could be repackaged, the latest version of the app resists repackaging.

3.3.2 Analysis of BHIM & UPI 1.0 Protocol

Bharat Interface for Money (BHIM) [37] is the Indian government's reference implementation of a payment app over UPI and was launched along with UPI 1.0. We discuss findings from our analysis of BHIM's user registration process for a user Alice whose UPI ID is her cell number. We instrument BHIM to see the protocol data it exchanges with the UPI server during registration. We show an example of how we instrument BHIM in the Appendix.

Listing 3.2: HTTP GET Instrumentation Code

```
sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
new-instance v1, Ljava/lang/StringBuilder;
invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()V
const-string/jumbo v2, "Log_debug_upi_str0: "
invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v1
invoke-virtual {v1, p1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v1
invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
move-result-object v1
invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
new-instance v1, Ljava/lang/StringBuilder;
invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()V
const-string/jumbo v2, "Log_debug_upi_restclient_map0: "
invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v1
invoke-virtual {p2}, Ljava/lang/Object;->toString()Ljava/lang/String;
move-result-object v2
invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v1
invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
move-result-object v1
invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
```

3.3.3 BHIM User Registration Protocol

Steps 1-10 on the left of Figure 3.3 are the steps of the client-server handshake between BHIM version *1.3* and the UPI 1.0 server, with minimal and relevant protocol data shown. The screen numbers (circled) on the left indicate the screenshot of the app in Figure 3.2 that generated the traffic. We describe the ten steps of UPI's *default workflow* below.

1. Step 1: When Alice starts BHIM, BHIM first requests Alice permission to send SMS messages (for later use) (Figure 3.2, #2). Once BHIM gets the permission, BHIM sends Alice's device details such as the device's Android version, device ID, make, manufacturer, and model to the UPI server as an HTTPS message.

2. Step 2: UPI server sends Alice a 13-digit registration token that identifies her device and waits to get the token back from Alice as an SMS message.

3. Step 3: BHIM app sends the registration token as an SMS message to the UPI server. BHIM waits for SMS delivery confirmation using the *sendTextMessage* API's *deliveryIntent*.

4. Step 4: When the UPI server receives the SMS, it (i) learns that Alice got the token; and (ii) gets her cell number from the message. The UPI server uses this information to hard-bind Alice's cell number to her device. UPI server also sends a confirmation to BHIM that it received the SMS.

5. Step 5: BHIM requests a status of its device's hard-binding from the UPI server by sending the registration token back to the server as an HTTPS message.

6. Step 6: The UPI server responds with a verification status that includes Alice's customer ID, a registration token, etc. back to Alice. By now, the UPI server has verified both Alice and her device (Figure 3.2, #4).

7. Step 7: BHIM asks Alice to set a passcode (Figure 3.2, #5). The app concatenates the SHA-256 hash of Alice's passcode with her cell number and sends it as an HTTPS POST request to the UPI server.

8. Step 8: The UPI server issues a login token to Alice (BHIM), which confirms that her profile is setup.

9. Step 9: BHIM then shows Alice a list of banks that support UPI (Figure 3.2, #6). When Alice chooses her bank from this list, BHIM sends a bank ID to the UPI server.

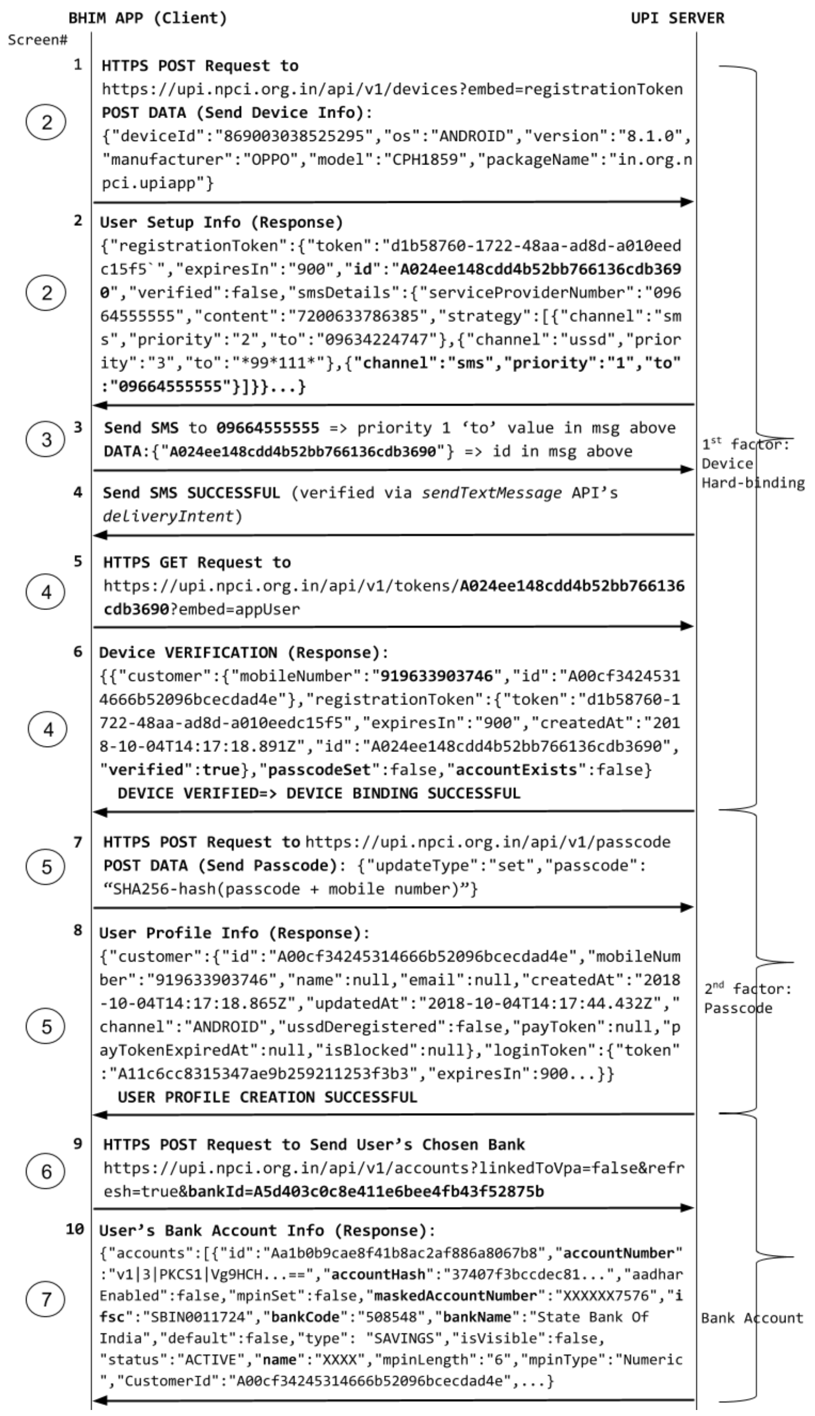
10. Step 10: The UPI server sends Alice's bank account details such as her masked account number, the hash of the account number, bank name, IFSC code, etc. back to BHIM (Figure 3.2, #7).

The protocol description until now has seen two factors—a) cell phone (and hence a device fingerprint) as required by the UPI spec; b) a secret passcode— both of which BHIM sends to the UPI server during the handshake. For BHIM, this means that the payment app server that authenticates a user's passcode and the UPI server that verifies a device's fingerprint is the same, a fact that is not surprising since the designers of UPI also wrote BHIM.

Finally, to enable transactions, Alice sets a UPI PIN on her bank account for which she needs her bank's debit card number and expiry date, as mentioned in Section 3.2.1.

Alternate Workflow1. In the default workflow described above, BHIM sends the device registration token to the UPI server as an SMS message for device hard-binding (Step 3). In case the UPI server does not receive the SMS, thus failing to hard-bind, BHIM provides an alternate workflow for hard-binding, as shown in Figure 3.4. BHIM prompts Alice to key-in her cell number; BHIM sends the keyed-in cell number along with the device registration token to the UPI server as an HTTPS message. The UPI server sends an OTP to Alice, which she must enter to complete device binding. The remainder of the protocol proceeds as before.

Alternate Workflow2. If Alice, an already registered user, changes her cell phone, then the UPI server has to re-bind her cell number with the new cell phone. At the time of device binding, the UPI server finds that an account for Alice already exists and notifies BHIM of the same (*accountExists* flag in Step 6). The UPI server prompts Alice for her passcode, and once Alice is verified (Step 7), the server sends back Alice's bank account information that she previously added to BHIM (Step 10). This workflow makes it convenient for Alice to transfer her bank accounts to another phone, without going through the hassle of adding all her bank accounts again.



```
1  HTTPS POST Request to
   https://upi.npci.org.in/api/v1/devices?embed=registrationToken
   POST DATA (Send Device Info):
   {"deviceId":"869003038525295","os":"ANDROID","version":"8.1.0",
   "manufacturer":"OPPO","model":"CPH1859","packageName":"in.org.n
   pci.upiapp"}

2  User Setup Info (Response)
   {"registrationToken":{"token":"d1b58760-1722-48aa-ad8d-a010eed
   c15f5`","expiresIn":"900","id":"A024ee148cdd4b52bb766136cdb369
   0","verified":false,"smsDetails":{"serviceProviderNumber":"096
   64555555","content":"7200633786385","strategy":[{"channel":"sm
   s","priority":"2","to":"09634224747"},{"channel":"ussd","prior
   ity":"3","to":"*99*111*"},{"channel":"sms","priority":"1","to"
   :"09664555555"}]}}...}

3  Send SMS to 09664555555 => priority 1 ‘to’ value in msg above
   DATA:{"A024ee148cdd4b52bb766136cdb3690"} => id in msg above

4  Send SMS SUCCESSFUL (verified via sendTextMessage API’s
   deliveryIntent)

5  HTTPS GET Request to
   https://upi.npci.org.in/api/v1/tokens/A024ee148cdd4b52bb766136
   cdb3690?embed=appUser

6  Device VERIFICATION (Response):
   {{"customer":{"mobileNumber":"919633903746","id":"A00cf3424531
   4666b52096bcecdad4e"},"registrationToken":{"token":"d1b58760-1
   722-48aa-ad8d-a010eedc15f5","expiresIn":"900","createdAt":"201
   8-10-04T14:17:18.891Z","id":"A024ee148cdd4b52bb766136cdb3690",
   "verified":true},"passcodeSet":false,"accountExists":false}
     DEVICE VERIFIED=> DEVICE BINDING SUCCESSFUL

7  HTTPS POST Request to https://upi.npci.org.in/api/v1/passcode
   POST DATA (Send Passcode): {"updateType":"set","passcode":
   “SHA256-hash(passcode + mobile number)”}

8  User Profile Info (Response):
   {"customer":{"id":"A00cf34245314666b52096bcecdad4e","mobileNum
   ber":"919633903746","name":null,"email":null,"createdAt":"2018
   -10-04T14:17:18.865Z","updatedAt":"2018-10-04T14:17:44.432Z","
   channel":"ANDROID","ussdDeregistered":false,"payToken":null,"p
   ayTokenExpiredAt":null,"isBlocked":null},"loginToken":{"token"
   :"A11c6cc8315347ae9b259211253f3b3","expiresIn":900...}}
     USER PROFILE CREATION SUCCESSFUL

9  HTTPS POST Request to Send User’s Chosen Bank
   https://upi.npci.org.in/api/v1/accounts?linkedToVpa=false&refr
   esh=true&bankId=A5d403c0c8e411e6bee4fb43f52875b

10 User’s Bank Account Info (Response):
   {"accounts":[{"id":"Aa1b0b9cae8f41b8ac2af886a8067b8","accountNumber"
   :"v1|3|PKCS1|Vg9HCH...==","accountHash":"37407f3bccdec81...","aadhar
   Enabled":false,"mpinSet":false,"maskedAccountNumber":"XXXXXX7576","i
   fsc":"SBIN0011724","bankCode":"508548","bankName":"State Bank Of
   India","default":false,"type": "SAVINGS","isVisible":false,
   "status":"ACTIVE","name":"XXXX","mpinLength":"6","mpinType":"Numeric
   ","CustomerId":"A00cf34245314666b52096bcecdad4e",...}
```

Figure 3.3: **BHIM User Registration Default.** The figure shows the client-server handshake between BHIM app on a user’s device and the UPI backend servers. The circled numbers on the left correspond to the GUI screen in Figure 3.2 where the request-response occurs. The smaller numbers on its right indicate the step number. BHIM masks the bank account number in step 10 of the handshake. The authors masked the other info to safe-guard privacy.

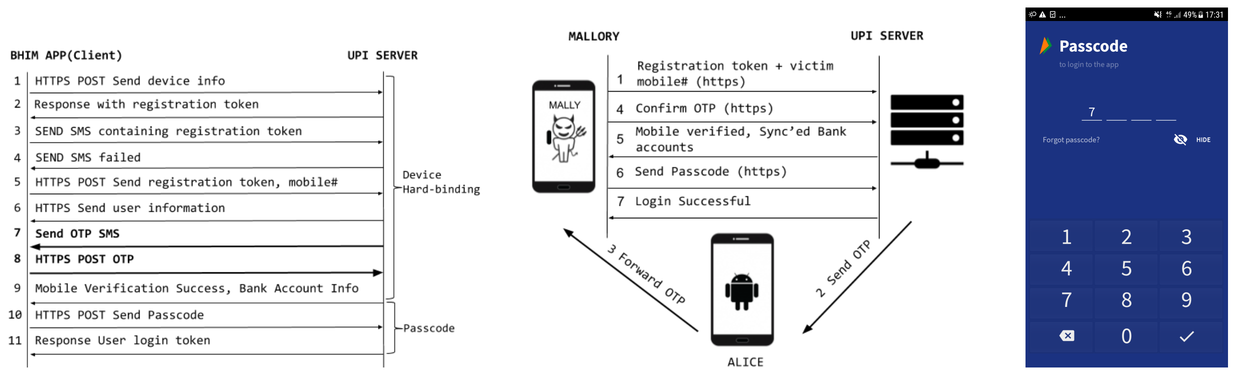

BHIM APP(Client)
UPI SERVER
1 HTTPS POST Send device info
2 Response with registration token
3 SEND SMS containing registration token
4 SEND SMS failed
5 HTTPS POST Send registration token, mobile#
6 HTTPS Send user information
7 Send OTP SMS
8 HTTPS POST OTP
9 Mobile Verification Success, Bank Account Info
10 HTTPS POST Send Passcode
11 Response User login token
Device Hard-binding
Passcode
MALLORY
UPI SERVER
MALLY
1 Registration token + victim mobile# (https)
4 Confirm OTP (https)
5 Mobile verified, Sync'ed Bank accounts
6 Send Passcode (https)
7 Login Successful
3 Forward OTP
2 Send OTP
ALICE
Passcode
to login to the app
7
Forgot passcode?
HIDE
1 2 3
4 5 6
7 8 9
0

Figure 3.4: **BHIM Alternate Handshake & Attack.** This figure shows BHIM app's alternate registration workflow when the default registration workflow fails (on the left). This alternate workflow can be exploited by an attacker controlled app to leak the OTP sent by the UPI server (as shown in the middle). On the right, is BHIM's passcode entry screen with an overlay on top that leaks the user's passcode.

3.3.3.1 Potential security holes—initial analysis

Before we describe the attacks on the UPI protocol, we first discuss three potential security holes that we observe:

1. *Potential Security Hole #1:* For an attacker Eve to take over Alice's account, one of the first barriers to overcome is UPI's device binding mechanism that binds Alice's cell number with her cell phone. For Eve to break the binding, Eve must able to bind her cell phone with Alice's cell number. Though the default workflow makes this hard, the *alternate workflow1* provides a potential fallback that allows Eve to send Alice's cell number as an HTTPS message from Eve's phone.

2. *Potential Security Hole #2:* The *alternate workflow1* uses OTP verification for device-binding. If Alice, say, enters a friend Bob's cell number on her phone, the UPI server will send the OTP to Bob's phone. If Bob shares that OTP with Alice, then Alice can confirm the OTP to the UPI server, which will hard-bind Alice's phone to Bob's cell number. As a result, Bob will receive all future SMS messages sent by the UPI server to Alice.

3. *Potential Security Hole #3:* In UPI's default workflow, Alice at no point provides a secret that she shares with her bank to confirm her identity. Nevertheless, the UPI server reveals an

existing user Alice's account details in the *alternate workflow2*.

None of the security holes by themselves are exploits as yet. Below we discuss the potential attacks as a result of these holes.

3.3.3.2 Attack #1: Unauthorized registration, given a victim's cell number

In this attack, we show how a remote attacker, Eve, can set up a UPI account, given a victim's cell number. For the attack to succeed, Eve requires only one thing: the victim's cell phone to have *Mally* app installed.

The attack setup is as follows. Eve on her phone has a repackaged version of BHIM that has client-side security checks disabled. Eve sets up a command and control (C&C) server, puts out Mally as a potentially useful app on various app stores, and waits for unsuspecting users to install Mally. As discussed in the Threat Model (Section 3.2.3), Mally has RECEIVE_SMS permission. An unsuspecting user Alice, uses a legitimate version of BHIM on a non-rooted phone, as is the best practice for Android.

For attacks to happen, Eve must have a way to discover a victim's cell number. To simplify the attack description, we assume that Mally also has READ_PHONE_STATE permission, which it uses to get the cell number from the victim's phone (almost 35% of the apps use this permission [259]). We show in Section 3.3.3.6 how Eve can discover a victim's cell number without the READ_PHONE_STATE permission.

Below we show how Eve can register with the UPI server as Alice, after Alice unwittingly installs Mally on her phone.

1. *Mally: I am installed!* Mally, once installed on Alice's phone, reports to Eve's C&C server over the Internet (Android automatically grants INTERNET permission). Mally reports Alice's cell number to Eve as a way for Eve (i) to discover Alice's cell number; and (ii) to associate the instance of Mally with Alice, which is essential for Eve to scale the attacks to many users.

2. *Eve: Use the cell number for hard binding:* Eve exploits *Potential Security Hole #1* in BHIM's workflow to bind her device to Alice's cell number as shown in Figure 3.4. Eve starts by putting her cell phone in airplane mode while remaining connected to the Internet through Wi-Fi. BHIM app on Eve's phone starts the handshake by sending Eve's device details. The UPI server responds with a device registration token for Eve. Ideally, Eve's BHIM must relay the token back to the UPI server via SMS. However, since Eve has turned off SMS messaging, the SMS containing the token fails to deliver. BHIM prompts Eve to key-in a cell number and Eve keys-in Alice's cell number. BHIM now sends Eve's device registration token and Alice's cell number to the UPI server as an HTTPS message for hard-binding. The UPI server then sends an OTP to Alice.

3. *Mally: Intercept the OTP.* On Alice's phone, Mally intercepts the incoming OTP message because its RECEIVE_SMS permission allows it. Mally then sends the OTP to the attacker's C&C server as an HTTPS message, along with Alice's cell number. (The cell number here is not strictly required. It merely allows the C&C server to associate each OTP with a victim and thus reduce some guesswork, in case it receives OTPs from other Mally installations.)

4. *Eve: Acknowledge the OTP.* The C&C server sends an SMS message containing the OTP to the attacker's phone. Note that the BHIM app normally checks the origin of the OTP message it receives and accepts the OTP only if it is from a known UPI server. However, Eve disabled this safeguard before the attack in the repackaged version of BHIM on her phone, thus exploiting *Potential Security Hole #2*

5. *Eve: New BHIM user? Create BHIM's Passcode:* BHIM on Eve's phone will ask for BHIM's 4-digit passcode. Now Eve does not know if Alice is a new user of BHIM or a registered user. However, Eve can determine this from Step 6 of the handshake where the UPI server sets a flag called *accountExists* to *false* for a new user. Eve can proceed to set a new passcode for a new user Alice. We discuss the workaround for the attack on an existing BHIM user in *Attack #1′*.

6. *Eve: Select the bank from the bank list.* Eve next selects each bank one-by-one on BHIM's bank selection screen until she finds one that the UPI server accepts. The UPI server will accept a bank if Alice has an account at that bank and has her cell number registered with that account.

   The UPI server does not appear to restrict brute-forcing—an error just brings the user back to the bank selection screen. In any case, brute-forcing is difficult to prevent since the list of banks is relatively short, and Eve can try out some of the larger banks where most people are likely to have an account with such as the State Bank of India or ICICI Bank.

Eve can repeat Attack #1 until she discovers all of Alice's bank accounts and registers with them.

3.3.3.3 Attack #1′: Eve: overcoming BHIM's passcode check for existing BHIM user

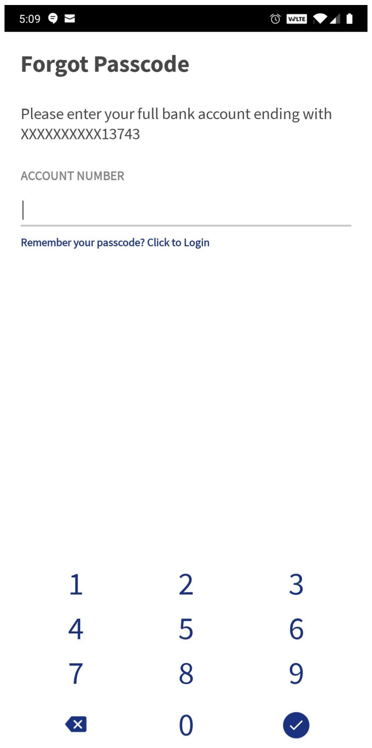
The first workaround is for Eve to wait for Mally to intercept and leak the new passcode. We found that Mally can do this as follows. *Mally* waits for Alice to launch BHIM. Mally detects BHIM's login activity to draw an overlay on it (see Figure 3.4, keys demarcated for clarity). To draw the overlay, *Mally* exploits a toast overlay vulnerability CVE-2017-0752 [177] that requires no additional permissions from the user. Once Mally intercepts the passcode, it forwards the passcode to the C&C server.

The second workaround is for Mally to request and use Android's accessibility permission, which enables Mally to observe user interactions and intercept the passcode.

An attacker may, at this point, choose to reset the user's passcode. We find that BHIM's passcode reset workflow requires a user's bank account number instead of the debit card number as shown in Figure 3.5. On the surface, it seems unlikely that Eve will know Alice's bank account number, and this, in isolation, may have been a reasonable passcode reset process. However, as described in *Potential Security Hole #3*, recall that the default UPI workflow reveals a user's bank account number. Eve can use the bank account number to reset Alice's BHIM passcode, courtesy of the UPI server.

Attack #1 on a registered user Alice stalls when BHIM prompts the attacker Eve for Alice's BHIM passcode. We present three solutions to overcome the passcode barrier.

Impact of Attack #1 and #1′. Eve cannot do transactions on the linked bank accounts after a successful registration. This attack, however, leaks private data such as the set of banks where Alice has bank accounts as well as Alice's bank account numbers. We also noticed that the UPI server sends a device registration token, a customer identifier, a login token, a hash of the account number, and the bank's account number back to BHIM (client) during the protocol handshake (see Figure 3.3). BHIM masks the bank account number but, nevertheless, the UPI server sends it, and Eve can get to it using the repackaged BHIM on Eve's phone. The Attack #1 is also a precursor to Attack #2 or Attack #3, which are more devastating. Note that the use of accessibility is only helpful in simplifying the attack; we do not require it for Attack #1.


5:09

Forgot Passcode

Please enter your full bank account ending with XXXXXXXXXX13743

ACCOUNT NUMBER

Remember your passcode? Click to Login

Figure 3.5: **Forgot Passcode.** This figure shows how BHIM's passcode reset workflow requires a user's bank account number instead of the debit card number that is used to secure transactions. Bank account number is revealed during BHIM's client-server handhsake to an attacker.

3.3.3.4 Attack #2: Unauthorized transactions on bank accounts given cell number and partial debit card number

In this attack, which follows Attack #1, Eve extends the previous attack to enable transactions on a bank account of a user Alice that does not use any UPI apps. For the attack to succeed, Eve requires additional knowledge about Alice: the last six digits of Alice's debit card number and expiry date. Debit cards are carelessly given to unknown people in stores and restaurants in India at the time of checkout (often with cell numbers, as cashiers routinely collect cell numbers to send discount offers or give reward points). The majority of debit cards in India also carry the bank name. Using a debit card for purchases in stores or online in India requires the user to key-in a secret PIN. In this attack, even without the debit card PIN from Alice, with access to the debit card information alone, Eve can set a UPI PIN to enable transactions on the associated bank account.

Impact. Losing or sharing one's debit card information along with the cell number (not the actual card, the actual cell, or the debit card PIN) can enable an attacker to set a UPI PIN and do transactions on one's bank account. Eve does not need bank account numbers or any of Alice's passcodes. The attack appears to be less scalable than Attack #1, however, since Eve needs to harvest debit card numbers along with associated cell numbers. For users who lose the two pieces of data to Eve and also install Mally, the impact is devastating. Eve could empty their account, with money transferred to any user in India. The attack does not even require a victim to have ever used a UPI app previously. To reset the UPI PIN, Eve requires the last six digits of the debit card number, expiry date, and an OTP, all of which she has.

3.3.3.5 Attack #3: Unauthorized transactions without debit card numbers

This attack follows from Attack #1′ for an existing user Alice. Such a user would have previously set up a passcode to log in to BHIM and UPI PIN to authorize transactions. Unfortunately, Mally can intercept the UPI PIN using either toast overlays or by requesting accessibility permission. As an alternative to intercepting UPI PIN, Eve can attempt to reset the UPI PIN (recall that Eve has already registered with the bank account in Attack #1′). As we described in the previous attack, resetting the UPI PIN requires debit card information, which reduces this attack to Attack #2. In short, either Mally intercepting UPI PIN or Eve possessing Alice's debit card information appears to be required. Eve now has all the factors to do transactions from her phone as Alice.

Impact: Eve can transfer money out to arbitrary UPI-based accounts in India. Note that for an attack on an existing user, Eve does not require any knowledge about Alice except for two things that Mally intercepts— an SMS message and the UPI PIN.

3.3.3.6 Eliminating the need for READ_PHONE_STATE permissions

The attacks we described so far relied on Mally knowing the victim's cell number and sending it to the C&C server, as a precursor to all the attacks. Now, we describe how Eve can associate a victim's cell number with an instance of Mally without Mally needing the READ_PHONE_STATE permission.

Given a set C of all targeted cell numbers (which is any list of cell numbers — valid or invalid), the following steps precede Attack #1:

1. For each cell number in C, send an SMS to that number with the following content: [receiver's cell number, "SMS TEST"] (or any such message).

2. Consider a subset SC of phones C that have Mally installed. Mally looks for the string "SMS TEST" and saves the cell number in the SMS as the victim's cell number.

   All instances of Mally that receive such an SMS message can thus learn their victim's cell number and report back to the C&C server to initiate the user registration protocol.

3.3.3.7 Whose problem: Android or UPI?

There is a potential question as to whether the attacks we discovered are primarily due to limitations of Android's permission model or due to flaws in the UPI design (and who should fix them). We think there are problems with both. We note that no bank-related credentials are required for an adversary to get a user's bank account number, given the user's cell number (in any of the handshakes– default or alternate). Attack #2 uses the last six digits of a debit card number and expiry date, a weaker threshold than for online and in-store purchases using debit cards where the entire number and the PIN is typically required in India. Alternate workflows in the UPI protocol contribute significantly to enabling our attacks. We, of course, leverage Android's security limitations as well, just as any good attacker would be expected to. We further discuss this issue in Section 3.5.

3.3.4 Other UPI 1.0 Apps

We now discuss whether the attacks on BHIM apply to the users of other UPI 1.0 apps. Our findings from testing three apps popular at the time of the study— PhonePe, Ola Wallet, and Samsung Pay—suggest yes. As shown in Figure 3.6, at the time of UPI 1.0, BHIM and PhonePe were the most popular UPI apps. PhonePe is also one of India's oldest payment apps. We did not include Google Pay (called Tez then) since it was not widely used, and Paytm was popular more for its wallet features. Below we discuss the attacks and its nuances under the same threat model.

First, these apps differ from BHIM because they are "third-parties" that integrate with UPI. Each third-party app uses its own factors for user profile setup. Hence, as discussed in the UPI specs Section 3.2.2, for third-party apps, their payment app server does the passcode-based authentication of a user while the UPI server verifies the device fingerprint and UPI PIN.

NPCI requires third-party apps to use NPCI's interface (libraries) for device fingerprinting and entering UPI PIN. We confirm that these apps internally use a common NPCI library to interface with the UPI server at the time of manual inspection. The UPI interface is accessible to a third-party app only after the user authenticates with the third-party payment app server. Thus, device binding and UPI PIN set up is done with the UPI server only after the user's passcode is set up with the payment app server.

Attack #1, unauthorized registration of a new user, can now be done by an adversary by setting up a user profile with the third-party app server and then exploiting the potential security holes of Section 3.3.3.1. Third-party apps make it easy for an attacker Eve to set up a profile. Eve can do it in two ways— Eve can either create a profile from her phone using her cell number (which is straight-forward) or create a profile from her phone using Alice's cell number. As an example of the latter, PhonePe provides an option to key-in a cell number at the time of user profile setup. Eve can use this option to key-in Alice's cell number in the app. For Eve to set a passcode on behalf of Alice, Eve needs an OTP the PhonePe server sends Alice. However, Eve can get the OTP through Mally on Alice's phone, given Mally's RECEIVE_SMS permission. The rest of Attack #1 continues as before, and Attack #2 follows from Attack #1.

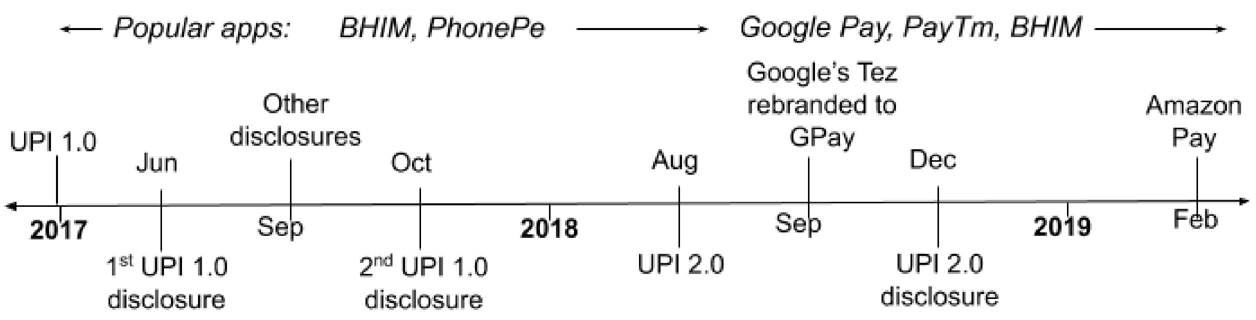


Figure 3.6: **Popular UPI apps and disclosure timelines.** This figure shows the general timeline of UPI 1.0 and UPI 2.0 general availability release along with an overall indication of the most popular payment apps during each timeframe . Also shown (below the timeline bar) are the timelines of when we disclosed the vulnerabilities we found with CERT-IN and UPI.

For Attack #1′ on an existing user, an adversary can exploit any authentication workflow flaws on the third-party app or app server. Once logged in, Eve can exploit the potential security holes (Section 3.3.3.1). For Eve to log in as an existing user, Eve either has to get Alice's password or has to reset Alice's password. To get Alice's password, Mally can either use the toast overlay attacks or the accessibility permission. A straightforward approach, however, is to exploit the app's passcode reset mechanism. On PhonePe, for instance, the passcode reset relies only on an OTP. On Ola Money, passcode reset requires a secret that is set up at the time the user creates a profile (which we could intercept). We note that once Eve logs in as Alice on Eve's phone, PhonePe logs Alice out from her phone. In Ola Money, however, Alice will not receive any notification since the app by design permits login from many devices. The rest of Attack #1′ continues as before, and Attack #3 follows from Attack #1′.

Samsung Pay (SPay) is slightly different in that its security measures make use of a Trusted Execution Environment (TEE) [217] implementation called KNOX. To use SPay, a user must have a Samsung account configured at the time of setting up the phone and additionally configure her fingerprint or a SPay PIN. SPay does not integrate with UPI; instead, it integrates with two UPI apps—Paytm and MobiKwik. Hence a user can choose one of the two apps that come with SamsungPay (they are also available for download separately on Google Play). Since both Paytm and MobiKwik app servers do not integrate with KNOX, they cannot use KNOX's hardware-based security features for device hard-binding at the time of user registration. The user's fingerprint or SPay PIN is used to authenticate a user with the device; neither the payment app servers nor the UPI server uses it for user registration. We test SamsungPay using MobiKwik. Mobikiwk's workflow is the same as Ola Money except that its passcode reset workflow uses a passcode and OTP, both of which we can intercept. This makes SPay prone to attacks that result from integrating with third-party UPI apps.

3.3.5 UPI 1.0 Responsible Disclosures

We reported the vulnerabilities of BHIM to NPCI, CERT-IN, and CERT-US, with the initial disclosure to CERT-IN in June 2017. We followed up with our disclosures again in Oct 2017 (timelines in Figure 3.6). Subsequently, we reported the vulnerabilities to CERT-US and got the following CVEs: CVE-2017-9818, CVE-2017-9819, CVE-2017-9820, CVE-2017-9821 for BHIM. We also got CVEs for our disclosures to other app vendors from CERT-US (CVE-2018-15660, CVE-2018-15661, CVE-2018-17400, CVE-2018-17401, CVE-2018- 17402, CVE-2018-17403) and a $5k bounty from Samsung (CVE-2018-17083) for a sensitive data leak. The original CVEs disclosed relied on accessibility permission, though we later determined that the attacks can be carried out without it.

3.3.6 Preliminary Analysis of UPI 2.0 Protocol

In August 2018, UPI made the first update to the UPI specification, UPI 2.0, over a year after we first reported the vulnerabilities to them. Based on our disclosures, UPI 2.0 does prevent our attacks in the current form. We present our preliminary findings; a detailed analysis of UPI 2.0 is currently ongoing. We follow the same approach we employed for UPI 1.0 and reverse-engineered the UPI 2.0 protocol using UPI 2.0 versions of four popular apps— BHIM, Google Pay, Paytm, and Amazon Pay. Google Pay (GPay) and Paytm are the leaders in the market, each with a 36% market share.

Some of our findings are as follows. We evaluate the UPI 2.0 version of BHIM (which is also used by many banks as their official UPI app under their own brand, e.g., BHIM SBI Pay and BHIM

PNB). We found that NPCI now forces an update on BHIM to its latest version. In UPI 2.0, in addition to the device information we saw in UPI 1.0, BHIM also sends the device's IMEI number, SIM number, network type, etc., to the UPI server for device hard-binding. In BHIM's latest update, NPCI removed the *alternate workflow1*, and hence the *Potential Security Hole #1* that we exploited for our attacks, a positive change. However, the other vulnerabilities persist as detailed below.

On GPay, we can set up a user's profile similar to how we did for Attack #1 and Attack #1$'$ in third-party UPI 1.0 apps. From GPay's traffic, we find that GPay authenticates with Gmail servers using OAuth2. Thus an adversary Eve can set up a GPay account as follows. Eve can use her own Gmail ID on her phone and can key-in Alice's cell number at the time of login to GPay. Google sends an OTP to Alice's cell number, which Mally can intercept (given Mally's RECEIVE_SMS permission). For Eve to proceed, GPay must send an SMS message containing Alice's device registration token back to the UPI server from Alice's phone.

In the absence of the *alternate workflow1* that previously enabled the attacks, we explored SMS spoofing as a means for Eve to send an SMS message to the UPI server. For the attack to work, the UPI server must get the spoofed SMS message from Alice's cell number. For proof-of-concept, we tested SMS spoofing with several services that claim to provide non-anonymous SMS spoofing. However, it did not work for a test number we own in India. While we can send SMS messages either anonymously or using a default number provided by the SMS spoofing service, we are unable to control the sender number of the SMS message, a must for the attacks to work. We are currently exploring this and other SMS related attack vectors noted in prior research [210]. Alternatively, Mally can request SEND_SMS permission and send the SMS message from Alice's phone.

On Paytm, we studied the handshake by instrumenting the app with debug statements at the bytecode level. Below is a snippet of the bank account information that Paytm receives during the handshake. The authors mask all the details below for privacy. We note that just as before, UPI sends back the bank account details without requiring a user to provide any credentials shared with the bank. We confirm the same on Amazon Pay as well. Amazon Pay uses Amazon credentials and the default cell number set in a user's Amazon account. To create a profile, an adversary Eve can set Alice's cell number in her Amazon credentials.

```
"defaultCredit":{"bank":"State Bank Of India",
"ifsc":"SBIN0008626",
"account":"000000379085XXXXX",
"accountType":"SAVINGS",
"name":"BXXXXXX TXXXX",
"branchAddress":"AMXXXXXXXXX"
```

Thus, we have confirmed that sensitive information leaks (similar to those in Attack #1) still exist. An open question remains on the possibility of other attacks, such as performing unauthorized transactions.

3.4 Lessons Learned

Below, we summarize the problems in the design of the UPI 1.0 protocol that enabled potential attacks.

1. The UPI protocol reveals bank account details of a user in any handshake (default or alternate), given the user's cell number and no bank-related credentials.
2. Device hard-binding, the first factor, relies on data that is easily harvested from a device. UPI does not use any secrets for this step.
3. A weak device binding mechanism allows a user (or an adversary) to bind her cell phone with a cell number registered to the bank account of another user.
4. Setting the UPI PIN, the second factor, requires partial debit card information printed on the card, which is not a secret. The debit card PIN, a secret a user shares with the bank, is never used. This is a lower bar as online, and in-store purchases require the entire card number and the debit card PIN.

5. When transferring an existing user's UPI account to a new phone, UPI does not require the user to provide any bank-related credentials or the printed debit card information to authorize transactions from the new phone. The UPI protocol relies on the UPI PIN alone.

6. On third-party apps, the passcode, the third factor, is managed by the third-party app server and hence easy to bypass. An attacker can bypass the passcode requirement by setting up an attacker-controlled profile (using attacker credentials) with the app. In this case, UPI effectively relies only on two factors— device binding and UPI PIN.

7. The bank account number leaked from the default workflow of any of the third-party apps is enough to reset a user's passcode on another app (such as BHIM).

We note that though UPI 2.0 closes the weak device binding mechanism #3 above, the other issues persist. The overall weakness in UPI is that user registration requires only the knowledge of a cell number and the ability to receive one SMS message from that number.

Attacks only require Mally to do two things: provide the OTP during registration and, for attacks on existing users of UPI, steal their UPI PIN. Need for Mally can be circumvented in two ways— unauthorized transfer of a user's cell number to the attacker or by social engineering attacks. Both are feasible, and social engineering attacks are scalable in India, given the cheap labor cost. For non-users of UPI, getting them to reveal an OTP during registration is sufficient.

There are significant risks associated with relying on cell numbers as the only means of user identification. Banks in India accept any cell number that the user registers with their accounts— there is no cross-check to verify if the cell number given actually belongs to the user. It is not uncommon, for example, for members of a family to provide the same cell number to the bank for their individual bank accounts. Thus, a person with access to family members' debit card numbers can add all their bank accounts to the same app for transactions. One may view this either as a convenience or a security and privacy risk, depending on one's perspective.

Finally, we would like to clarify that our claim is not that all the high-level lessons learned are new; most security principles are well-known by now. Nevertheless, we want to contextualize the lessons learned from the perspective of a widely adopted financial protocol. We note that both the designers of Android and UPI contribute to the flaws we discovered, which made getting app vendors to do fixes difficult. App vendors often blame it on Android design or users, who should not be granting dangerous permissions to apps. At the same time, UPI protocol designers could have factored in the current state of Android and security-awareness among users in India and made the protocol more secure.

It is well-known by now that security by obscurity does not help. We think the risks could have been better addressed had UPI published the protocol details once it was internally vetted, thus allowing the research community to analyze it further. We show how protocol analysis from the point-of-view of an adversary trying to uncover unpublished workflows and secrets, though important, is often overlooked for application-level protocols.

Limitations of our study: A limitation of our study is that we only studied seven UPI apps to analyze the security of the UPI protocol. Automated analysis techniques could not be used given the number of security defenses these apps use. Prior research by Reaves et al. [209] also reverse-engineered seven apps that resisted automated analysis. However, we consider seven to be a reasonable number for our work since our focus was on uncovering flaws with the UPI protocol that is common across the apps. Also, the apps we analyzed have 88% of the market share combined, and of the 88 UPI apps, a majority of them are minor variations of BHIM, which we analyzed. Nevertheless, a larger study could provide additional insights into the security of the payment ecosystem in India and will also be useful to other countries that decide to use a common payment interface.

3.5 Mitigation

We discuss possible mitigation strategies against the attacks and their pros and cons below.

UPI mitigations. We discuss steps the government can take to address some of the issues we have raised.

Minimizing protocol data: Our attacks show how protocol data revealed during the default workflow was used to exploit an alternate workflow. This was possible because the UPI server sent more data than the client needed to see. For instance, while the masked bank account number is useful to display on the screen, bank-specific details such as the bank name, account number and IFSC code, sent in the clear can be excluded from the handshake.

Secure alternate workflows: We leveraged two alternate workflows in our attacks, as summarized in Section 3.4. Though UPI 2.0 closes one of the flows, the other alternate flows are either unsecured or secured using weak credentials. For instance, an alternate workflow allowed a user to bind her cell phone with a cell number registered to the bank account of another user, even without providing any secrets pertaining to the other user.

Mandate opt-in into UPI apps: Currently, as we are aware, UPI services are by default available to users of a bank that is integrated with UPI; the UPI guidelines do not require users to opt-in with their bank. An opt-in requirement would increase risk awareness as well as cut down security risks for non-UPI users such as credit card users, cash users, or users of wallet apps. Alternatively, a user could be required to do an in-person verification with their local bank branch to register for UPI services on their cell phone. This can prevent unauthorized registrations of a user, which automatically eliminates the other attacks.

Provide opt-out option: As a follow-up on the previous mitigation, non-users and users wanting to discontinue UPI services must be allowed to opt-out for security and privacy reasons. The downside of making UPI optional is the negative impact it may have on UPI adoption.

Use debit card number + something user knows: Debit cards in India are Chip+PIN cards, and doing transactions with them always requires entering a PIN. In contrast, doing transactions via the UPI apps requires neither—only the information that is printed on the card—resulting in a weaker authentication path. Fixes to this are unfortunately difficult if Mally is powerful enough to intercept PIN entry. However, assuming user interactions can be secured on Android (e.g., see [84]), UPI guidelines requiring the user to enter a secret shared with the bank to enable transactions will be useful.

Require strong device binding: The UPI specification could require payment apps to do a stronger device-to-cell number binding. Since binding is one of the most critical steps of the protocol, the bank may issue a one-time secret to the user out-of-band, say, when the user visits the bank for UPI activation. The user has to enter this secret the first time she uses the UPI app on her phone. Additionally, the UPI server must verify that the UPI app it is communicating with is an official app running on a non-rooted phone. If the UPI server can somehow establish that, then an attacker may not be able to use a repackaged version of a UPI app to register an account. Unfortunately, this is tricky to enforce.

Android mitigations. In the attacks we describe, the attack starts when Mally on a user's phone gets the user's cell number as an SMS from the attacker. A possible defense would be for Android to have a policy that prevents SMS permissions from being requested by apps. Google is already moving in that direction. As of January 2019, Google announced that apps could not request SMS permissions unless they are the default SMS handler and get explicit approval from Google. How effective this policy is, remains to be studied. We note that this does not make the attack impossible. It would merely require Mally not just to be installed but also accepted as the default SMS handler (or get approved as an exception by Google). Also, the policy is specific to the Google Play store—apps from other stores could still introduce risks. Many popular carriers in India support alternate app stores such as Aircel and Airtel that allow SMS-triggered downloads [185].

User mitigations. Since Eve requires a user's cell number to initiate the attack, using a private cell number for bank accounts may slow down an attack. Unfortunately, it does not entirely prevent it. If the user has installed Mally, Mally suffices to detect the user's cell phone number (Section 3.3.3.6. Thus, users would also need to be careful to never install apps with read or receive SMS permissions on phones they use for banking.

3.6 Related Work

Panjwani et al. did one of the first studies on an Indian payment system called EKO, a mobile service provider [193]. They show PIN recovery attacks that could result in a user impersonation attack. Reaves et al. [209] first analyzed 47 mobile apps from 28 countries for SSL vulnerabilities and then manually reverse-engineer seven branchless banking apps, including three Indian payment apps (Airtel Money, Oxigen Wallet, and MobileOnMoney). They discover that an attacker can bypass authentication because of the use of an insecure channel, the use of weak crypto, or the use of weak passwords. A follow-up work by Castle et al [47] studies 197 payment apps, including some from Southern Asia (the apps they study is not listed). Castle et al. point out that payment apps have sufficient safeguards to prevent attacks, and the vulnerabilities pointed out by Reaves et al. are either because of regulatory constraints or from using old Android phones. They corroborate their findings with developer interviews with participants from well-established organizations.

Payment apps have been studied in other countries, as well. Yang et al. [266] notes implementation weaknesses in the third-party SDKs included by Chinese financial apps that can result in integrity attacks on financial transactions. Jung et al. [145] studies repackaging attacks on seven different banking apps in Korea. Their attacks could bypass integrity checks and anti-virus checks of banking apps. Yacouba et al. [148] launched a DDoS attack on a banking server through a repackaged banking app. Roland et al. demonstrates an NFC relay attack on the Google Wallet payment system [215].

Research has pointed out several vulnerabilities in financial applications. Taylor et al. [234] did a static analysis of financial apps on Google Play. They discover weaknesses such as the creation of world-readable and writable files, the use of unsecured content providers, and the use of weak random number generators. Bojjagani et al. [40] perform static and dynamic analysis on banking apps to discover 356 exploitable vulnerabilities, details unknown, from an unknown set of samples. AlJudaibi et al. [8] discuss 11 significant threats faced by mobile devices such as insecure data storage, weak server-side control in third-party apps, use of a rooted device, and lack of security in software and kernel. Chothia et al. [52], Stone et al. [228] and Bojjagani et al. [39] analyze both Android and iPhone apps for lack of hostname verification when an SSL certificate is pinned. Their results show how popular banking apps with these vulnerabilities are prone to phishing and man-in-the-middle attacks.

Protocol flaws that result in attacks on payment cards that use chip and PIN (EMV) [41, 160, 173, 216] and 3 Domain Secure 2.0 [6], an authentication protocol for web-based payments, are also studied before. Many issues concerning financial inclusion for developing countries such as Brazil and Africa have been extensively studied [120, 183, 265]. Weaknesses in financial systems as a result of excessive reliance on OTPs [51, 171, 210, 248] and its implication on Internet-based services are also well-known [1, 74, 147, 171, 220, 267].

Prior studies on Indian payments apps were done before the Indian government launched the Unified Payment Interface, a first of its kind. To the best of our knowledge, we are the first to conduct a study on UPI.

3.7 Chapter Conclusion

In this paper, we used a principled approach to analyze the UPI 1.0 protocol and uncovered core design weaknesses in its unpublished multi-factor authentication workflow that can severely impact a user. We showed attacks that have devastating implications and only require victims to have installed an attacker-controlled app, regardless of whether they use a UPI app or not. All the vulnerabilities identified were responsibly disclosed. A subsequent software update to UPI 2.0 prevents the discussed attack vectors for an exploit. Unfortunately, several underlying security flaws remain that suggest a need for further vetting and security analysis of UPI 2.0, given the protocol's importance for mobile payments in India. We discussed the lessons learned and potential mitigation strategies. Finally, we expect our findings to be useful to other countries that look to implement a common backend infrastructure for financial apps.

CHAPTER 4

Measurement Study on Geodifferences in Mobile App Ecosystem

4.1 Introduction

We often view the Internet as a worldwide medium for communication without regard for geographic location [137]. However, studies have shown differences in Internet equity, for instance, in access to Internet content based on a user's geolocation [125,154,205,264]. While censorship is a well-known enabler for such regional differences [4,29,189,231,272], there are emerging trends of geoblocking, a phenomenon where service providers or developers deny access to users in certain countries or regions. Recent studies on the web ecosystem show how service providers, given an option, lean towards indiscriminate blocking, effectively isolating certain countries (e.g., Cuba) and essential services (e.g., banking) [2, 164, 249]. Recognizing the gravity of this problem, in 2018, the EU passed regulations that ban unjustified geoblocking [81].

Mobile users worldwide access the Internet through apps downloaded from app markets like Google Play. Thus, any interference in app markets can result in different *app equity*, i.e., different access to apps or security and privacy offerings based on a user's geolocation. For example, Figure 4.1 shows different views of the LinkedIn app's homepage on Google Play for users in three countries. The app is available for install in the US and unavailable in Iran and Russia, though users perceive unavailability differently in the latter two countries. Although there are over 8.9 million apps and 3.5 billion smartphone users worldwide [184, 213], geodifferences in the mobile app ecosystem have received only limited attention.

In this paper, we present the first large-scale investigation into geodifferences in the mobile app ecosystem. Broadly, we are curious to know: (i) if we can download an app from, say, the US, Iran, and Russia at the same time; and (ii) whether the app, if available, has geodifferences. Given our goals for a wide geographic study, we select 5,684 globally popular apps from Google Play. Google Play is the largest and the most accessible app market, with over 2 billion active devices and 2 million apps that reach over 190 countries [18, 45], making it an obvious choice for our geographic study. We collect our measurements from 26 countries carefully chosen to have reliable direct vantage points while ensuring ample diversity in terms of geography, gross domestic product, and Internet freedom scores by Freedom House [93].

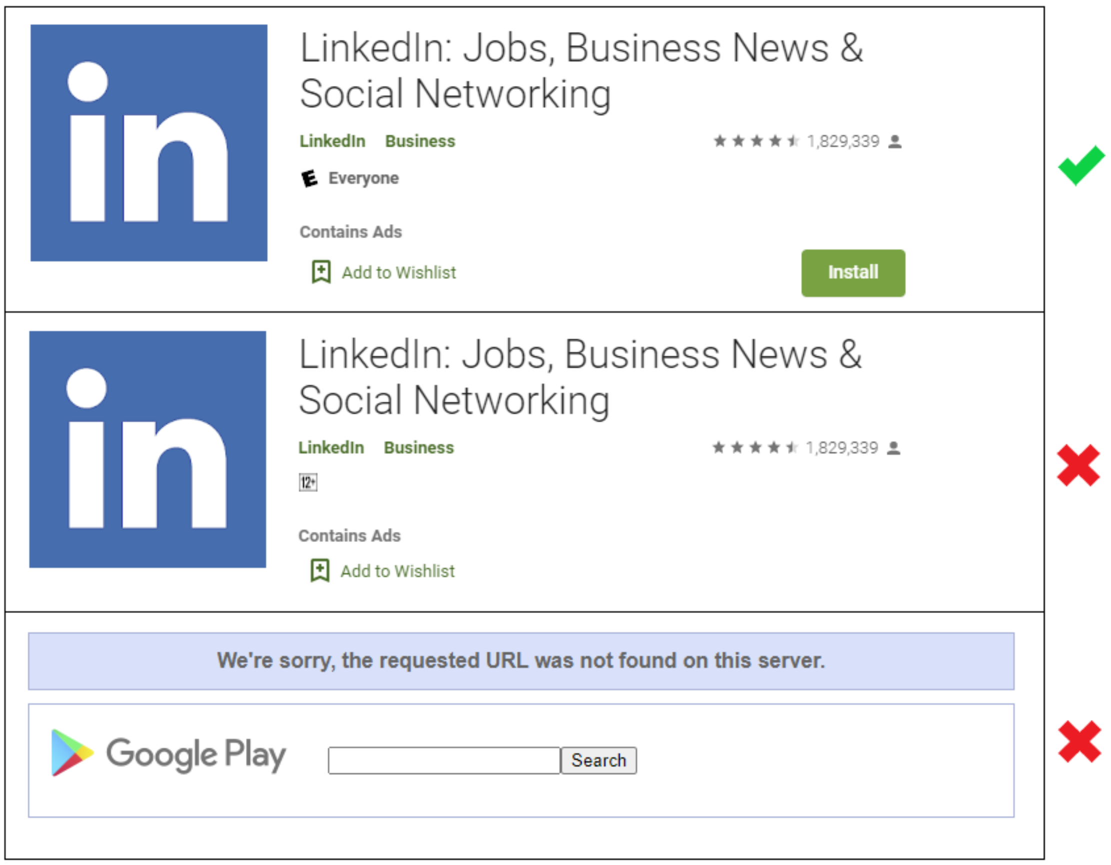
There are *significant* challenges in conducting large-scale measurements for thousands of apps from different locations. For instance, Google Play is a volatile app market with millions of apps, governed by opaque regulations [159, 200, 235]. Unlike web measurements, there are no known fully automated tools for collecting mobile measurement data. While web geoblocking has some clear signals (e.g., `403 Forbidden` HTTP response), the indicators for blocking in the mobile app ecosystem are unknown. Additionally, to measure geodifferences in apps, we need to download app binaries from many countries.

This paper makes several contributions that enable large-scale measurement studies of complex mobile app ecosystems. First, we identify the variables that impact measurements from Google Play through a set of preliminary experiments. Second, we design a semi-automated measurement technique that captures a reasonable snapshot of Google Play as seen by users in 26 countries. Using a parallel test-bed, we collect 117,233 app binaries and 112,607 privacy policies, which to the best of our knowledge, is the largest multi-country app dataset in the research community. Third, using control experiments, we extract the signals for geoblocking from Google's opaque server-side responses and deduce who is responsible for the blocking. Fourth, for apps that are not geoblocked, we investigate if users in certain regions are exposed to higher security and privacy risks from geodifferences in app features. Finally, we provide recommendations for app market proprietors like Google Play to address the issues we find.

Our results show high amounts of geoblocking with 3,672 globally popular apps geoblocked in at least one of 26 countries. We find that Iran and Tunisia have the highest geoblocking rates of 2,256 and 2,681 apps, respectively. In contrast, previous work on the web found up to 71 geoblocked domains from the Alexa top 10K list in the most affected countries [164]. While our data corroborates anecdotal evidence of takedowns due to government requests (e.g., recent ban of Chinese apps in India) [241], unlike common perception, we find that blocking by developers is significantly higher than takedowns in all our countries and app categories, and has the most influence on geoblocking in the mobile app ecosystem. Amongst the countries, Iran is the most blocked by developers and is the top outlier country in every app category. We believe this is because developers have unmoderated access to country targeting features on Google Play, which, as research has shown [164], could disproportionately isolate some regions.

While most developers release the same apps geographically, we find 596 apps with geodifferences, with confirmed instances of developers targeting different app versions to different countries, thus exposing users in certain countries to higher security and privacy risks. For instance, we find instances of the same apps requesting different permissions, using additional ad trackers, or selectively using unencrypted communication in different regions. While our data shows the positive influence of data protection laws (e.g., GDPR) in privacy policies in regions where such laws are enforced, we also find the same apps using outdated policies in countries with older legislation. Privacy policies of some apps in certain countries like Iran and Turkey could not be downloaded due to geoblocking of the websites hosting them.

We suggest several steps that Google and other app market proprietors could take to address some of the issues we find. For instance, app market proprietors could moderate their country targeting features, push for transparency from developers on their need for geodifferences in apps, and redress the blocking of privacy policies in certain countries by hosting the app's policy themselves to ensure its availability. We shared our work with Google and submitted a full disclosure on all the apps for which we observed geodifferences in security and privacy features. Google's privacy team has



We're sorry, the requested URL was not found on this server.

Figure 4.1: **User's View of Geoblocking:** *LinkedIn's* homepage on Google Play from the US, Iran, and Russia from top to bottom. Users in the US can download the app via the Install button, users in Iran see the homepage without the Install button, and users in Russia see a *URL not found* error on accessing the same page.

acknowledged our disclosures, and they are aware of the concerns raised through our research. To encourage further studies on the various aspects brought out by this work, we make our measurement data and code available at `https://github.com/censoredplanet/geodiff-app`.

4.2 Background

Internet fragmentation (or "balkanization") is emerging as a growing concern given the wide disparity in business and government practices worldwide [128]. Web geoblocking studies have shown that developers on a content delivery network (CDN, e.g., Cloudflare), given an option, indiscriminately geoblock their content, furthering the fragmentation. However, not much is known about geoblocking in the mobile app ecosystem, where app market proprietors or developers may block access to mobile apps that users use to access content from their mobile devices. Furthermore, it is poorly understood to what extent developers distribute different versions of apps in different countries and whether those versions differ on security and privacy protections. This study focuses on geographical differences (geodifferences) from geoblocking or differences in app releases.

4.2.1 Google Play

Google Play is the largest and the most accessible app market, with 2 billion active devices and over 2 million apps reaching over 190 countries [18]. Global app markets like Google Play are not the only means for developers to distribute apps. There are also local app markets, for instance, such as those hosted by cellular network providers (e.g., Docomo in Japan [76]) or independent publishers (e.g., CafeBazaar in Iran [85]). However, given Google Play's reach, any geodifferences here will have the most impact on users and makes it an obvious choice for our geographic study.

Google Play allows developers to restrict app availability to users, for instance, based on their country (country targeting) [100]. We call such geoblocking due to developers country targeting their apps as *developer-blocking*. Microsoft's Skype Lite, which is available only in India [166], and Hulu, which is available only in the US [127], are examples of developer-blocked apps. In a special case of country targeting, restrictions are imposed on access to paid apps or in-app purchases, for instance, due to embargo rules or Google's policy in a country (e.g., Iran) [20, 105]. Other forms of targeting are device targeting, where apps are released for specific devices, or carrier targeting, where apps are released for specific cellular service providers [14, 108].

At a high level, Google's transparency report shows content removal because of government requests and violations of Google's policies [114]. Real-world reports corroborate app removal requests (*takedowns*) specifically from Google Play for the same reasons. For instance, Google taking down LinkedIn in Russia [89] and the Indian government's ban of several Chinese apps [241] are examples of takedowns from government requests. On the other hand, Google's removal of Fortnite [23] is an example of a takedown due to a violation of Google policy. Broadly, we call the former *government-requested takedown* and the latter a *non-compliance takedown* and aim to distinguish the two in our work. While Google's non-compliance takedowns tend to be global, takedowns from government requests are regional.

Given how apps on Google Play are subject to developer-blocking and takedowns, a user's view of an app may vary across regions. Figure 4.1 shows an example of how users in the US, Iran, and Russia see the homepage of the LinkedIn app (as of June 2020). A user can download the app using the ”Install” button on its homepage in the US. A user in Iran, in contrast, sees the same homepage without the Install button, indicating that they cannot download the app. Finally, in Russia, accessing the app's URL returns an error. Given how a user's snapshot of Google Play may vary geographically, our goal is to characterize who is responsible for the differences we observe.

A user can access an app on Google Play directly via a URL with a fixed structure that points to its homepage. For the LinkedIn app, the URL to its homepage is `https://play.google.com/st ore/apps/details?id= com.linkedin.android`, where `com.linkedin.android` is the app's unique ID. The app's homepage contains its *metadata* such as the app's category, the number of installs, the app version, and the URL to download the `apk` upon clicking the Install button.

4.2.2 Android Security and Privacy

Android *apps* are programs written to operate on Android devices. An Android app is packaged into an installable archive called the Android Package (`apk`). An `apk` contains compiled classes, resources (e.g., images), assets (e.g., media files), configuration files, and a manifest file (`AndroidManifest.xml`). The Android manifest contains app settings used by both Google Play and the Android device, such as an app's unique app ID, its user-facing version name, internal version code for developers, permissions, and third-party libraries [15, 21]. This work studies app features that may have security and privacy risks to users.

Android has three notable security features that protect users—app permissions, signatures, and settings for encrypted communication. App permissions control access to system features (e.g., GPS) and is classified into one of four protection levels [12]: (i) *Normal*, which carries the least risk; (ii) *Dangerous*, which is high-risk and requires explicit user consent; (iii) *SignatureOrSystem*, which is privileged and used by system apps (e.g., apps by device vendors); and (iii) *Signature*, which is used to share data between apps.

Android apps are signed using a digital signature that serves as a bridge of trust between a user, developer, and Google Play. Android provides three signing algorithms: versions V1 (for Android versions ¡ 7.0), V2 (for Android 7.0+), and V3 (for Android 9+). Ideally, a developer has to sign with *all* three schemes for maximum security [16]. Note that there is also a V4 signing algorithm that was released September 2020 for Android 11 [38]. However, we exclude the V4 signature algorithm from this work since our `apks` predate Android 11.

Android allows developers to set their communication preferences via a *Network Security Policy* file (`network_security_config.xml`) [19] or an app's manifest file [22]. In the absence of this config file, prior to Android 9, an app's communications with servers by default use the unencrypted HTTP protocol unless disabled in the app's manifest file. However, with Android 9, the default is encrypted (HTTPS) communication. Given this nuance, not setting communication preferences is potentially dangerous since the system may default to unencrypted communication based on Android version.

Android developers may use third-party libraries for convenience or to provide services such as in-app advertisements or billing [33]. However, prior research has shown that third-party libraries may compromise a user's privacy by leaking their sensitive data [33, 121, 156]. To protect a user's privacy, Google requires developers to disclose collection of, access to, or use of sensitive data (e.g., personally identifiable information) [107], via a privacy policy that developers must host on an active URL. The policy must be listed on the app's homepage on Google Play [102] and must follow data protection laws when appropriate, including the US-based California Consumer Privacy Act (CCPA) [224] and the European Union's General Data Protection Regulation (GDPR) [132].

Country	ISO Code	Region	FHIF	Country	ISO Code	Region	FHIF
Canada	CA	NA	F(87)	Ukraine	UA	Europe	PF(56)
Germany	DE	Europe	F(80)	India	IN	Asia	PF(55)
USA	US	NA	F(77)	Zimbabwe	ZW	Africa	PF(42)
UK	UK	Europe	F(77)	Turkey	TR	Europe	NF(37)
Australia	AU	Oceania	F(77)	Russia	RU	Europe	NF(31)
Japan	JP	Asia	F(73)	Venezuela	VE	SA	NF(30)
Hungary	HU	Europe	F(72)	Bahrain	BH	Asia	NF(29)
Kenya	KE	Africa	PF(68)	UAE	AE	Asia	NF(28)
Colombia	CO	SA	PF(67)	Egypt	EG	Africa	NF(26)
South Korea	KR	Asia	PF(64)	Iran	IR	Asia	NF(15)
Tunisia	TN	Africa	PF(64)	Hong Kong	HK	Asia	-
Mexico	MX	NA	PF(60)	Ireland	IE	Europe	-
Singapore	SG	Asia	PF(56)	Israel	IL	Asia	-

Table 4.1: **Country List.** 26 countries with their ISO Code, region, and Freedom House Internet Freedom score, sorted by the freedom score (unavailable for three countries). Abbrv: NA= North America, SA = South America, F= *Free*, PF= *Partly Free*, NF= *Not Free*.

4.3 Measurement Design & Data Collection

Given that our goal is to study geodifferences, we want to capture a snapshot of apps from Google Play as seen by users in many countries. Measurements of such kind pose several challenges as noted in prior research on the web ecosystem [4, 164, 231], such as finding a vantage point closest to a user. However, in the mobile app ecosystem, we have the additional burden of downloading thousands of apps and their metadata from Google Play in different countries. These downloads may take weeks to complete, with app updates in between that complicate comparative analysis. We thus have to find vantage points suitable for long-running downloads, account for app updates, and factor in network errors and latency.

Country Selection. For this study, we carefully choose an initial list of 30 countries with ample diversity in geography, gross domestic product (GDP), and Internet freedom scores as measured by Freedom House [93]. We first considered choosing countries based on their GDP alone as in prior work on Internet geoblocking [164]; however, countries like India that are recently seeing a *crisis in expression* [28] have higher GDP than countries like Canada that are relatively free. Hence, we rely on *Freedom House's Internet Freedom* (FHIF) score, which numerous studies have used for country selection and analysis [4, 31, 196, 206], as an approximate indicator of Internet freedom. Since the study could potentially span several months, acquiring cost-effective and reliable vantage points was a significant challenge; we thus limit ourselves to 30 countries initially.

Testing Vantage Points. Prior research on Internet measurements has shown that data collected from different vantage points may have different properties [207]. Hence, we first conduct a preliminary study to confirm that a user's view of the app market from a non-residential vantage point is the same as a residential vantage point. To confirm this, we collect app metadata by proxying HTTPS requests to Google Play's home pages for a manually curated set of low-risk apps (e.g., Google Chrome, Netflix) from different vantage points and compare the data collected. We use Luminati, a commercial platform that sells access to residential proxy servers [157] and purchased numerous VPS/VPN servers for non-residential vantage points. We manually verified that app metadata collected using these vantage points was the same. We further confirmed this by collecting metadata from Google Play in select countries via a test Android app we developed. Given that these vantage points provided the same view of app metadata, we chose reliable non-residential vantage points whose advertised geolocation matched Google's detected location.

Final Country List. Table 4.1 shows our final list of 26 countries, their geographic region, and FHIF scores. We dropped four of our initial 30 countries (Brazil, Ethiopia, Cuba, and Saudi Arabia) because of unreliable vantage points. We exclude China because Google Play is unavailable there.

Choosing App Categories. Prior research has shown that Google is a "superstar" market dominated by a few most downloaded apps [271]. Given this and the wide scale of our comparative study, we select only the top apps in the most popular categories. Google Play has 32 top-level categories, and for Games, 17 sub-categories [104]. We first compute a category-wise breakdown of the top charts (i.e., top 200 apps) on Google Play in the top five FHIF *Free* countries in our list (Australia, Canada, Germany, UK, and USA). Then, we pick the top 20 categories and two others—MED, DATE—which contain apps we expect to collect sensitive information. In all, we study 22 categories as shown in Figure 4.2.

App Selection. To curate our app list, we collect the metadata of the top 200 apps in each of our 22 categories. Combining the apps thus collected for the same five FHIF *Free* countries, we

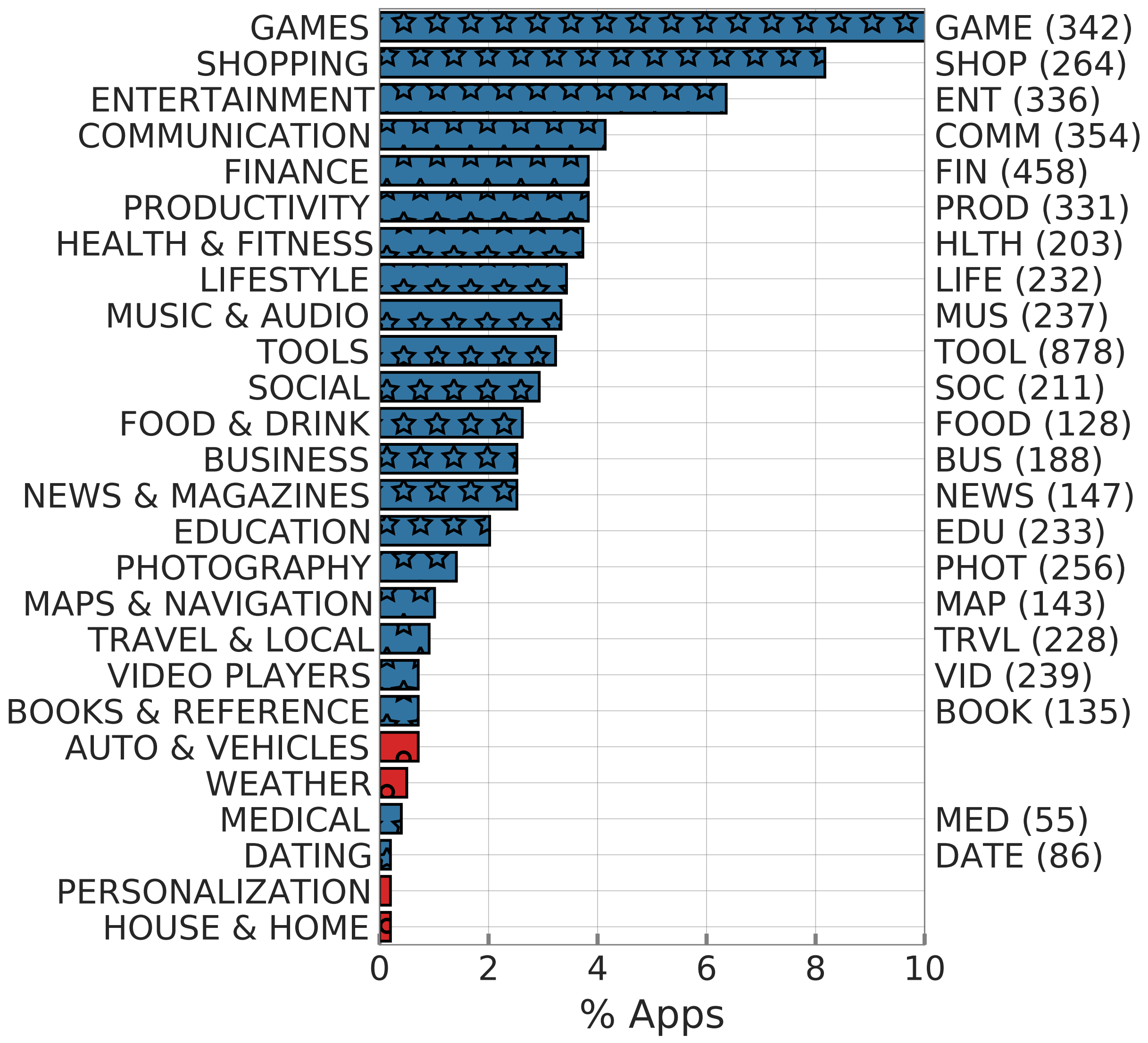


Figure 4.2: **Category Breakdown.** We compute the category breakdown of *top charts* of Google Play in 5 free countries. We study the top 20 categories and two additional categories MED and DATE (colored blue, full name on the left and abbreviation on the right with the numbers of apps per category).

get 17,351 unique apps. From this, we choose popular apps based on their number of installs, and use 1 million installs as the popularity bar for all categories except Games. For Games, we use 50 million installs as the popularity bar, as 47% of the apps with at least 1 million installs are Games. This trimmed our app list down to 4,465 apps. Finally, based on researcher interest in security and privacy, we add 1,219 apps from a keyword search for *security*, *privacy*, *vpn*, *ad blocker*, and *crypto*

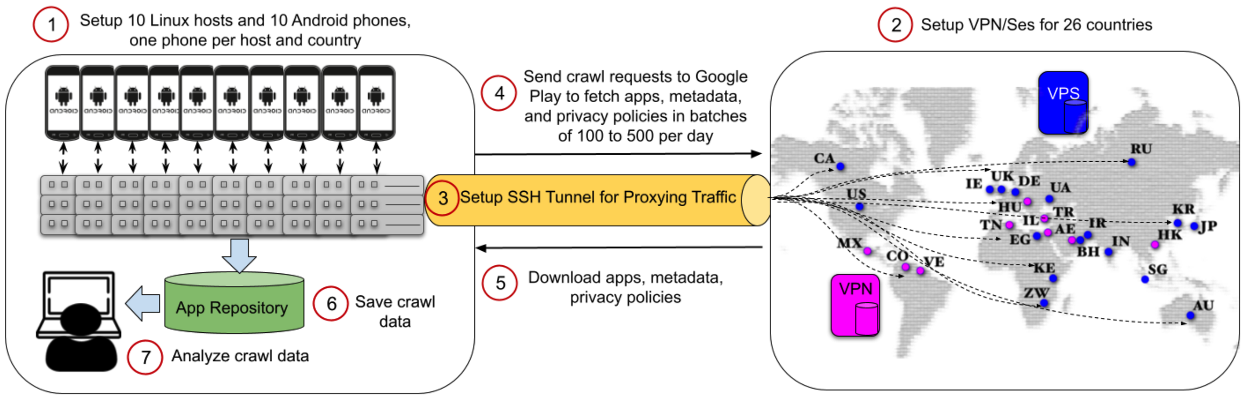

1 Setup 10 Linux hosts and 10 Android phones, one phone per host and country
2 Setup VPN/Ses for 26 countries
3 Setup SSH Tunnel for Proxying Traffic
4 Send crawl requests to Google Play to fetch apps, metadata, and privacy policies in batches of 100 to 500 per day
5 Download apps, metadata, privacy policies
6 Save crawl data
7 Analyze crawl data
App Repository
VPS
VPN
CA
US
MX
CO
VE
IE
UK
DE
UA
RU
HU
TR
TN
IL
EG
AE
IR
BH
IN
KR
JP
HK
SG
KE
ZW
AU

Figure 4.3: **Measurement Design and Setup.** Steps: (1) Set up ten Linux hosts and ten Google Pixel 2 phones, with one Google account for each country. (2) Set up VPN/Ses and SSH tunnels for downloads via VPSes. (3) Send crawl requests to Google Play to download in batches of 100–500 apps from all 26 countries, with each batch downloaded on the same day. (4) Save data to centralized repository for analysis.

wallet that belong to our selected 22 categories. Our final app list has 5,684 apps.

Measurement Design. Given the scale of our study, we want our measurement design to enable parallel downloads of apps from multiple countries. To eliminate inconsistencies from app updates during downloads, we conduct a preliminary longitudinal daily crawl of app metadata in 20 randomly chosen countries over 31 days and compute daily app updates. We find that only 1.1% of the apps have either a major or minor update per day on average and approximate the distribution of the percentage of apps updated on a given day to be N(1.1, 0.5) by the central limit theorem [176]. The median number of apps that have an update per day is 0. Based on this observation, we divide the 5,684 apps into batches of 100-500 apps and download each batch from every country within a 24-hour window. Doing so gets us a reasonable snapshot of an app within a batch in all 26 countries.

Data Collection. We download from ten countries simultaneously. For parallel downloads, we set up ten Linux measurement machines and ten Google Pixel 2 phones running Android 10, as shown in Figure 4.3. We set up these phones with one Google account per country. The Linux servers are set up with a *multipass* virtual machine [172], one per country, based on the vantage point's bandwidth and constraints of the host. To proxy download traffic, we set up an SSH tunnel to each VPS and use *proxychains4* as a SOCKS proxy [203]).

Error Code	Error Message	Error Reason
Err1	Item not found	**Takedown**
Err2	Google Play purchases are not supported in your country. Unfortunately you will not be able to complete purchases	**Developer-blocking**
Err3	Your device is not compatible with this item	Device-targeting
Err4	This item is not available on your service provider	Not registered with the service provider
Err5	The Play Store application on your device is outdated and does not support this purchase	Missing payment and billing info
Err6	The item you were attempting to purchase could not be found	Possibly incorrect device location or billing setup error by developer
Err7	Caused by SSLError (bad handshake)	Possibly network interference

Table 4.2: **Download Errors.** This table shows the error messages observed from failed app download requests to Google Play and the error codes we assign. Error reasons are our findings that clarify why these errors occur, specifically whether it is a takedown or developer-blocking.

We download 5,684 apps and their metadata in batches varying from 100–500 apps depending on the network conditions from all 26 countries on the same day. On average, we download a batch within a 9.5-hour window between the earliest and latest country. We download all batches in 15 days (June 2020).

Checks & Precautions. We deployed several mechanisms to mitigate transient download errors. We randomize app downloads and use a pre-calibrated rate-limit for each country. We retry every failed download until two consecutive retry attempts for the same app return the same error. Using a separate Google account for each country eliminates the possibility of fetching cached content. We also clear the Play Store cache on the phones between downloads. For countries that use a VPN vantage point, we also check for run-time changes to geolocation every 50 app downloads. We manually verify download error logs each day. For network errors that are not transient, we additionally re-crawl a random set of apps each day to confirm failures. We discard the day's download for all countries if a country is unreachable on a given day.

Scraper Implementation. We customize two scrapers to download an apk and its metadata from Google Play. The Google Play scrapers are Python modules—*PlaystoreDownloader* and *google-play-scraper*—that send HTTPS requests to a URL containing an app's ID (as described in Section 4.2) [115, 198]. The `apk` scraper requires the device's ID and Google account credentials.

To scrape privacy policy, we first get an app's privacy policy link from its metadata and use a Selenium crawler to download the policies [218].

Data Extraction. Considering our goal is a large-scale geographic study of apps, we only examine an app's static features. We use *aapt* and *aapt2* [17] to extract an app's permissions, version, and encrypted communication settings. In cases where *aapt* fails, we use *apktool* [24] to get decoded data from the `apk`. For app signature, we combine results from *keytool* [146] and *apksigner* [17]. For extracting third-party libraries, we use two tools used in prior research, `libradar` and `ExodusPrivacy` [7,82,117,158]. `libradar` returns a mixed list of third-party libraries present in an app. Given there is no clear way to filter ad libraries from that list, we use `ExodusPrivacy` specifically to collect ad and analytics libraries. To extract policy text from a privacy policy web page, we use *ReadabiliPy* and *Beautiful Soup* [208, 212].

Ethics. All the data used for this study is collected using our own mobile devices with traffic proxied through non-residential VPN/Ses we purchased. For all purchases, we used our real identities and agreed to comply with the terms and conditions of the hosting provider. In our preliminary testing to validate vantage points, we developed and shared a mobile app (including its source code) with our collaborators (all security researchers) to fetch app metadata from Google Play. The app collected no personal or device data. Our collaborators gave us informed consent and the collected data was discarded after successful tests. We reported all our findings to Google.

4.4 Data Characterization & Exploration

Using our measurement testbed, we successfully downloaded 5,385 of our initial list of 5,684 apps from at least one country. In all, we collected 117,233 app instances and 112,607 privacy policy instances from 26 countries. These apps are from 22 different categories, with 5,667 free and 17 paid apps, 2,365 apps that support in-app purchases, and 1,120 apps that provide content rating descriptions. The majority of these apps (86%) were last updated in 2020, indicating that most apps are actively maintained. We leverage this large-scale and diverse app data to investigate geodifferences. To the best of our knowledge, ours is the most extensive multi-country app collection in the research community.

While we successfully downloaded a large portion of the app instances from our vantage points, surprisingly, we discovered many apps that failed to download, with 299 apps that failed to download from any country. The download error messages varied for these apps. Given that there is limited knowledge of why these errors occur, we perform in-depth exploratory research to characterize these failures.

4.4.1 Characterizing Download Errors

While downloading the apps, we observed various error messages that range from "Item not found" or "purchase not supported in your country" to "device not compatible". Table 4.2 shows all the error messages and our assigned error codes. These error messages from the failed download requests to Google Play are opaque and do not clearly communicate why the download fails. As noted in Section 4.2.1, we are specifically interested in discovering whether the errors result from a non-compliance takedown, a government-requested takedown, or developer-blocking. Therefore, we perform control experiments using apps that are publicly known to be unavailable for the above reasons.

We find that our control apps that are known to be taken down by Google fail with Err1. The error is the same for both government-requested takedowns (e.g., LinkedIn in Russia), and non-compliance takedowns (e.g., Luna VPN - `com.luna.vpn` [222])[1]. However, for government-requested takedowns, Google removes the app only from the country issuing the order. In contrast, Google's non-compliance takedowns are applied to all 26 countries, confirming that such takedowns are global. Of course, if the developer decides to unpublish (remove) an app from everywhere, that also returns Err1 (global developer takedown), though we believe such occurrences are rare given

[1]Our characterization for takedowns rely on Google's transparency report. Google uses "government requests" as an umbrella term for content removal requests from the following requesters: judicial, executive, suppression orders, consumer protection authority, information and communication authority, court order directed at Google, government officials, and others.

our data consists of globally popular apps. We verify this by unpublishing our test app.

For the control apps that are known to be developer-blocked (e.g., SkypeLite in India [166]), we find Err2. We verify this by publishing our test app on Google Play and confirming Err2 when downloading from countries where we manually chose to block the app. We observe Err3 when some of our control apps fail to download because of developers device targeting their apps (e.g., Samsung's Secure Folder). Our apps that are carrier targeted (e.g., Sprint Music) fail to download with Err4. All paid apps fail with Err5 because Google Play requires a user's payment and billing information, which we did not set up. Finally, a few apps fail with Err6 in select countries. Though Google's official documentation provides no insights, public reports (e.g., Google discussion groups) suggest that this error is due to incorrect billing set up by the developer or conflict in the device's country settings with a user's actual location [113, 223].

Error Characterization for 299 Unavailable Apps. Considering that we seed our initial list from app metadata in five countries with high FHIF scores, it was surprising that 299 apps (5.3%) failed to download from all of our 26 countries for the following reasons. Using the error characterization from our control experiment above, we find that 143 apps failed to download with Err1, suggesting either Google's non-compliance takedowns or global developer takedowns. 126 apps fail with Err3 because of incompatibility with our measurement device. All 17 paid apps fail to download with Err5, and four apps specific to a service provider fail with Err4. One app, `it.mirko.transcriber`, is the only app that is developer-blocked (Err2) and is flagged "early access" by the developer indicating its limited release [106]. The remaining eight apps fail due to takedowns in some countries and developer-blocking or "not found" in others. Subsequent download checks show that they were removed from Google Play, indicating that Google or the developer may have been in the process of removing these apps entirely. We exclude all 299 apps for the remainder of this study.

SSL Errors in TN. Tunisia (TN) was unique among our countries in that a large number of apps (1,819), but not all, consistently failed to download with SSL bad handshake error (ERR7), even on repeated attempts. To root cause this error, we first look for client or server misconfigurations and

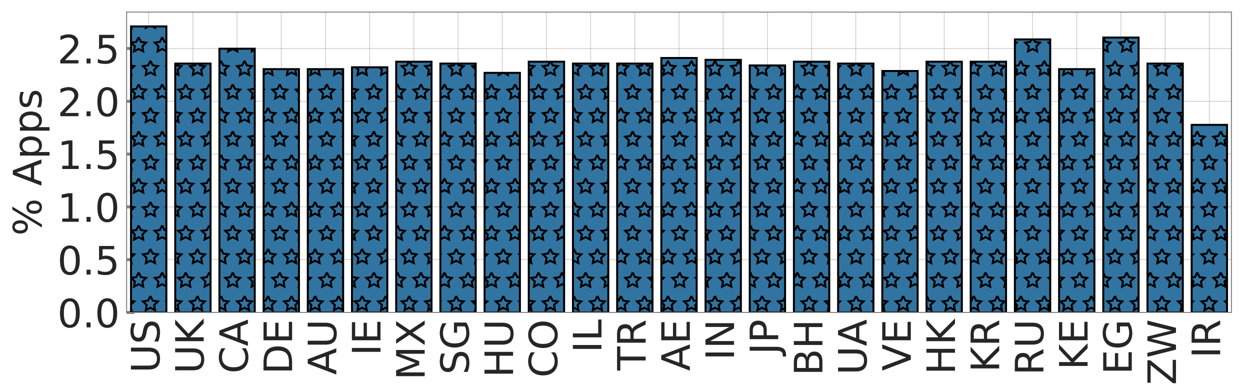


Figure 4.4: **Availability Discrepancy.** The percentage of apps with different availability based on app metadata versus `apk` download. Metadata, in aggregate, always over-reports availability everywhere. We exclude TN from the graph, which has 34% difference because of SSL errors preventing 32% of app downloads. We exclude TN (34%) from the graph, which has the highest percentage difference because of SSL errors preventing 32% of downloads though there are specific instances where the app is downloadable but metadata suggests non-availability

ruled that out as follows. We compared the packet traces of the client's communication with Google servers in TN and other African countries for select apps whose downloads succeeded and failed. In all countries, including TN, the SSL/TLS handshake completed successfully. We also ran the Qualys SSL Server Test [204] on TN's endpoint. We observed no anomalies, thus confirming that there was no client or server misconfiguration.

On further investigation, we found that the SSL error for TN only occurred after a successful SSL handshake and during the data transfer associated with the `apk` download. We found that the client (Linux host) received duplicate or replayed ACK packets during an app download, causing a failure. To further confirm a download error, we also manually attempted to install several apps via a VPN in Tunisia on our phone. For apps that failed, the install remained in a wait state indefinitely. Thus, our analysis of these failed apps in Tunisia suggests some form of network interference strategy at work that prevented the apps from being installed. Prior research has shown that censors may do deep packet inspection on SSL traffic and interfere with packets containing ACKs, including replaying them, to slow down or halt the data transfer [98]. We treat these apps as unavailable in Tunisia for this study.

Characterizing Availability Inconsistency. From a user's perspective, an app is available if: (1) its Google Play homepage (or metadata) has a download link; and (2) the actual `apk` is downloadable. Our data shows that there are apps that have a download link yet are not downloadable. For instance, in Tunisia, 1,948 apps that have a download link in metadata could not be downloaded. The difference is less in the remaining countries, with between 101 apps in Iran and 154 apps in the US not downloaded, despite having a download link. We flag these apps as unavailable in our study. Conversely, we also observe 13 apps (e.g., Google Pay, YouTube) that have no download link in Iran, yet the `apk` is downloadable by going to its URL directly. We consider these apps as unavailable in Iran as a user cannot access them. These observations point to a bigger issue when reporting on app availability—we find that measurements that only use app metadata consistently overestimate availability. A previous study [256] on app censorship measured availability based on app metadata alone, which we show is insufficient. For our study, we consider both conditions—a user's access to an app's download link and the `apk`.

4.4.2 Characterizing Security and Privacy

Security Features. In this section, we outline the security choices made by app developers in three of Android's notable security features—permissions, settings for encrypted communication, and app signature. We extract permissions from our collected apps and classify them into one of the four Android protection levels based on Android 10's source code and developer documentation [12]. Collectively, our apps request 4,816 unique permissions, of which only 229 are core Android permissions. Of the 229 core permissions, 32 are *Dangerous*, 67 are *Normal*, 53 are *Signature*, and 77 are *SignatureOrSystem* permissions. The remaining 4,587 permissions are custom and vendor-specific permissions, which we exclude as there is no clear way to assign a protection level to them.

Regarding encrypted communication, 2,825 apps (52%) explicitly disable encrypted communication in at least one country through the app's network security configuration file. The remaining apps either explicitly enable encrypted communication or use the system's default settings, which may be set to unencrypted communication depending on the Android version. Regarding app signatures, only two apps are signed using multiple signature schemes, as recommended by Google. While more than 70% of apps use the V2 signature algorithm, about 12% (663) of apps use V1, which is known to be vulnerable [260, 268]. Surprisingly, two apps also use Android's Debug certificate, which is not accepted by most app stores [16]. Apps also use deprecated hashing methods and weak key lengths [178, 179, 260, 268]; e.g., 2,358 apps use SHA1 with RSA, and 1,735 use 1024 bit keys with RSA.

Third-party Libs. We examine the third-party libraries in our apps, focusing on ad trackers. From the list of 400 trackers that `Exodus Privacy` looks for, we find a total of 274 unique trackers that are globally popular. We find 407 apps with no ad trackers (e.g., `org.torproject.android`). Consistent with existing reports [25, 78, 165], the top ad and analytics libraries in our apps are Google's—Firebase Analytics, AdMob, CrashLytics, Analytics—and Facebook's—Login, Share, Analytics, Ads, Places. Excluding ad trackers, the top five third-party libraries in our apps are `com/google/android/gms` (Google Mobile Services), `android/support/v4` (Android support), `com/google/gson` (JSON conversion), `okhttp3` (HTTP client), and `com/google/zxing` (Barcode processing).

Privacy Policies. We use a semi-automated approach to characterize the 112,608 privacy policies collected from 26 countries. A manual perusal of these policies shows that many downloaded pages are, in fact, error pages. To disambiguate an error (block) page from a valid policy page, we follow the approach taken by prior web censorship research [143, 164]. Using word count as a measure of page length, we study the distribution of word counts of all pages and find a threshold of 200 words reasonable to separate an error page from a policy web page. We further look for privacy-specific keywords in these pages, as we expect error pages to omit them. After excluding identical policies of an app, we apply the above heuristics on the remaining 17,025 policies and get 4,234 error pages. From the remaining non-error pages, we extract 8,737 English policy pages using Python's `langdetect`. We manually verify the disambiguated error and policy pages.

Besides finding error pages instead of privacy policies even where the app downloads, we observe other violations. For instance, 76 apps have no privacy policy link on their Google Play homepage, and another 56 apps have broken links or expired domains, all of which are explicit violations of Google's policy [102]. About half of these apps request *Dangerous* permissions, suggesting no disclosures on an app's access to sensitive information.

4.5 Results on Geodifferences

After characterizing and cleaning up our data, we investigate the prevalence of geodifferences in our 5,385 apps that are popular on Google Play. First, we discuss geoblocking, a type of geodifference in which an app is blocked to users from a particular country or region. Then, we describe another form of geodifference where an app's security and privacy offerings may vary based on the region.

4.5.1 Geoblocking

Even though Google Play is the most accessible app market, our data still suggests geoblocking in all 26 countries in our study. Of the 5,385 apps, 3,672 apps are geoblocked in at least one country. Iran (IR) and Tunisia (TN) have the highest blocking by a wide margin, with 2,256 and 2,681 apps geoblocked. Surprisingly, geoblocking varies in the other countries, with 300 apps blocked in the US to 800 apps blocked in Zimbabwe (ZW), indicating high variability based on the region. Compared to prior geoblocking investigation on the web [164], the blocking we observe is significantly high, even though our data consists of apps with millions of users globally. On the web, the maximum observed geoblocking was 71 domains from the Alexa top 10K sites in the most affected countries [164].

We next investigate the availability of each app across countries for all apps to detect potential blocking trends. We first compute the similarity between two countries C_i and C_j, as:

$$S(C_i, C_j) = \frac{|G_{C_i} \cap G_{C_j}| + | \sim G_{C_i} \cap \sim G_{C_j}|}{\bigcup_i \sim G_{C_i}} \tag{4.1}$$

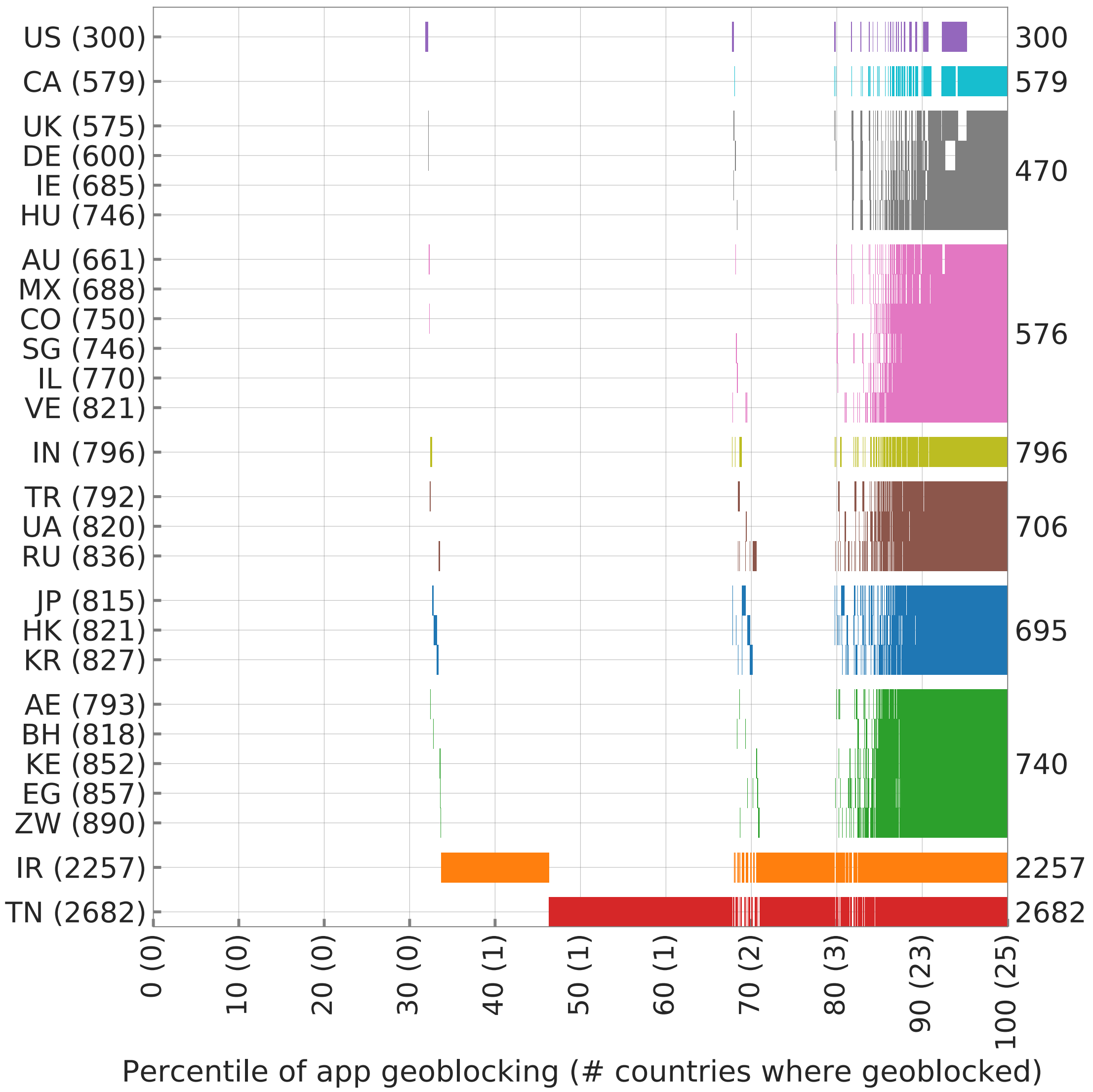


Figure 4.5: **Geoblocking per Cluster**: The countries are grouped by cluster, with geoblocking per country on the left axis and the intersection of geoblocked apps per cluster on the right axis. Each app is a data point on the x-axis (percentile when sorted by the number of geoblocked countries) shown in solid colors when geoblocked.

where, for country C_i, G_{C_i} is the set of apps that are geoblocked and $\sim G_{C_i}$ is the set of apps not geoblocked, and $\bigcup_i \sim G_{C_i}$ is the number of apps not geoblocked in at least one country (here, 5,385). We then use Ward's minimum variance method [144] to perform agglomerative clustering with the Hamming distance between countries computed as $S(C_i, C_j)$.

Figure 4.5 shows geoblocking in the 26 countries grouped by cluster, with the blocking per country on the left axis and the intersection of geoblocked apps within a cluster on the right axis.

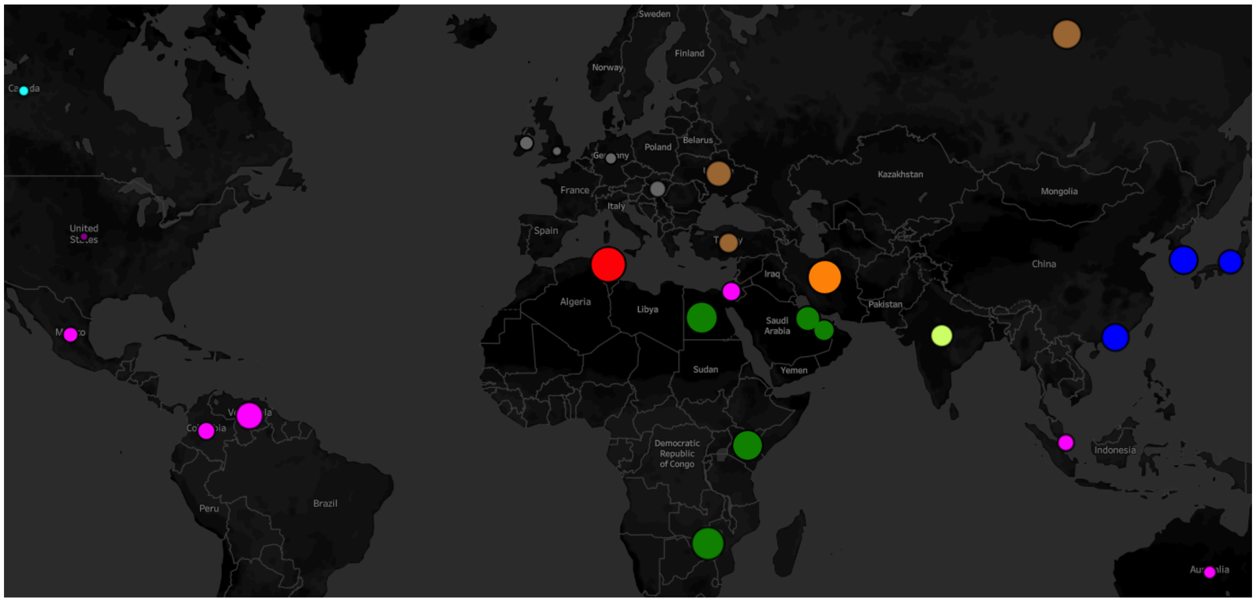


Figure 4.6: **World Map of Clusters**: This figure shows a world map of the availability (geoblocking) and is a complement to the cluster plot for app unavailability. Countries in the same geographic region have comparable app availability.

We find that the African countries experience the most geoblocking. Countries within the same region (e.g., the four EU countries) tend to have the same apps geoblocked, though minor variations exist. IR, TN, US, Canada (CA), and India (IN) are outliers and not clustered with any region. A world map of countries and their clusters based on app availability is shown also in Figure 4.6.

We investigate whether there exists a correlation between a country's app availability with its FHIF score and GDP using Spearman's rank correlation coefficient metric [96]. We find a moderate negative correlation (ρ=-0.58, p-value = 0.002) between a country's app availability and GDP. On the other hand, there is a moderate positive correlation (ρ= 0.64, p-value=0.001) between app availability and FHIF score. Despite the FHIF score's positive correlation with availability, we observe a few exceptions. For instance, CA, which has a higher FHIF score, has more geoblocking than the US and UK. Turkey (TR) and UAE (AE), which have very low FHIF scores, have higher availability than Japan (JP) and South Korea (KR). These exceptions could be because Freedom House, as far as we know, focuses only on government blocking of social media and communication apps [94].

We further examine whether there exists a correlation between a country's GDP and its FHIF score using Spearman's rank correlation coefficient metric. We find that there is a low, positive correlation (0.42) between Gross Domestic Product (GDP) and Freedom House's Internet Freedom (FHIF) Score, as shown in Figure 4.7. However, we observe countries like IN, which has relatively high GDP and lower FHIF score, and HU, which has lower GDP and a free FHIF score. We also observe countries with comparable GDP and different FHIF classifications (e.g. free CA and not free RU).

4.5.2 Who is Blocking?

While the statistics on geoblocking are valuable, we also want to know who is behind the blocking and the plausible reasons for such blocking. In this section, we are specifically interested in government-requested takedowns and developer-blocking of apps that are available in at least one country.

Government-requested takedowns. We observe that 61 unique apps are subject to government-requested takedowns (e.g., `com.truecaller` in TR, `mobi.jackd.android` in KR, and `com.mi.globalbrowser.mini` in the US). KR, specifically, has the most government-requested takedowns with 36 apps removed. The US has more than the median number of takedowns, while many countries with lower availability, particularly ZW, Bahrain (BH), Kenya (KE), and Egypt (EG), have the lowest of such takedowns.

We find a case of a government-requested takedown due to a violation of regional content regulation. For instance, in KR, 36 apps are taken down, of which 17 are gambling and game apps. This finding is consistent with reports of the Korean government's aggressive policy to block the distribution of apps with adult content, violence, or gambling [103]. Interestingly, KR is also an outlier in the categories DATE, ENT, LIFE, NEWS, and SOC, though to a lesser extent.

Certain app categories see more takedowns even when no government regulations appear to be violated. For instance, COMM apps encounter more takedowns in TR, and PHOT apps in IN. While

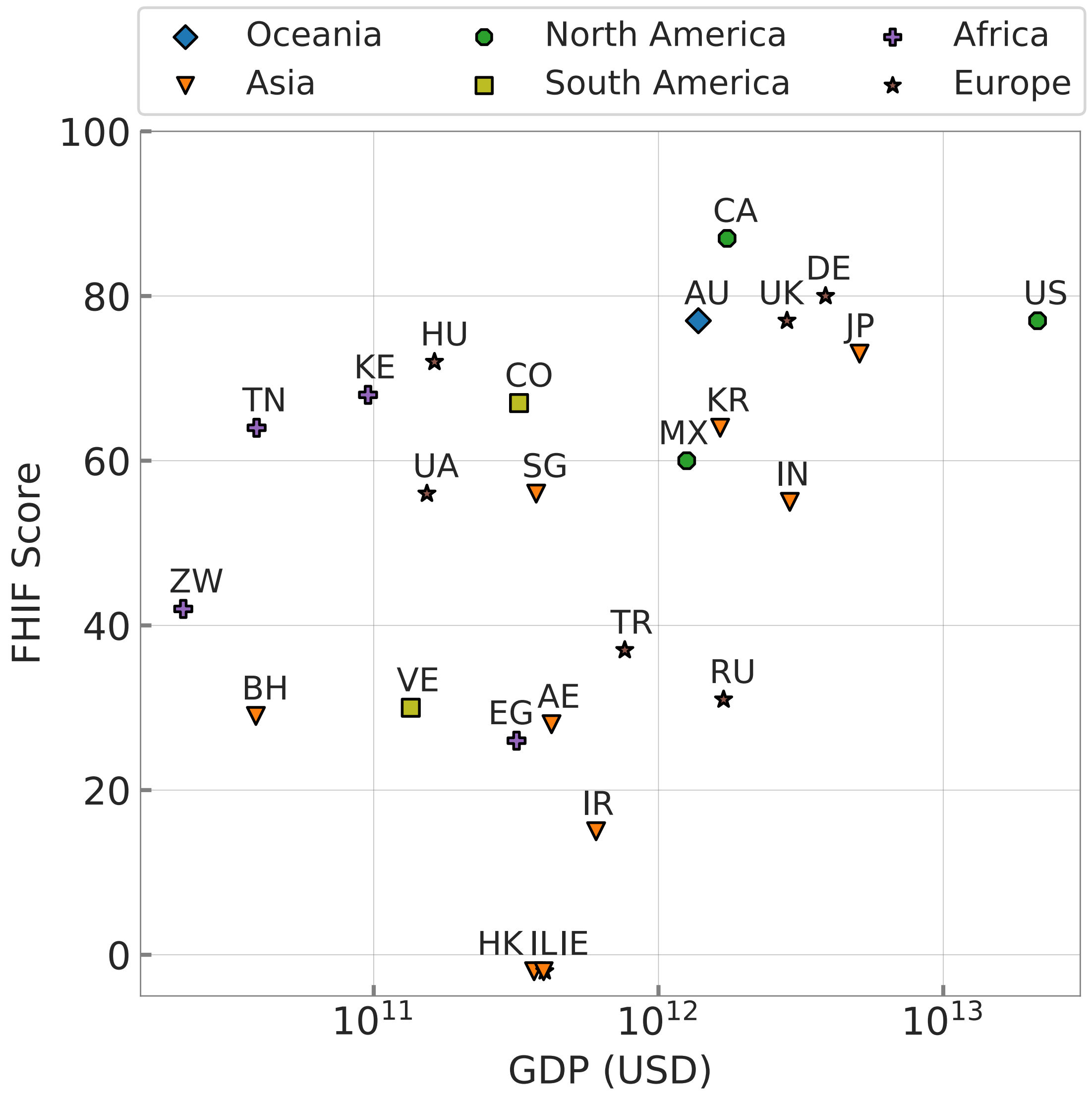


Figure 4.7: **GDP vs. FHIF Score**: The x axis is log-scaled. While there is a low, positive correlation between GDP and FHIF score, there are countries with comparable GDP with varying FHIF classifications (e.g. CA and RU). Note that HK, IL, and IE do not have a FHIF score.

countries with higher than the median geoblocking such as IR, RU, and TN are outliers in certain categories such as DATE, GAME, LIFE, surprisingly, countries such as the US, UK, and DE with lower geoblocking rates also are outliers in others such as BUS, TOOL, SHOP. Figure 4.8 (left) shows the distribution of government-requested takedowns by category and the outliers in each.

Developer-blocking. Compared to government-requested takedowns, we find that the proportion of apps that are developer-blocked is significantly higher in all countries and app categories, and

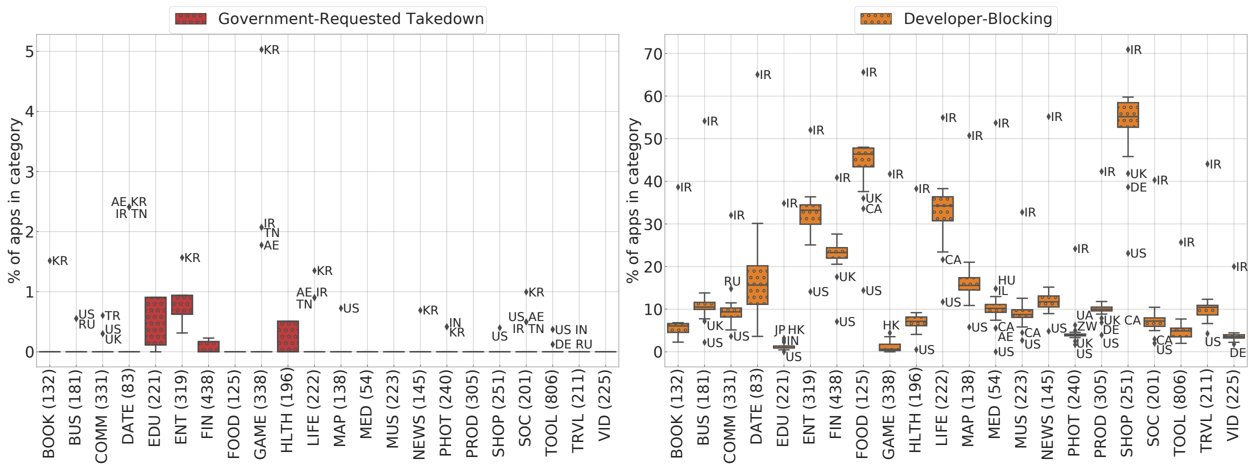


Figure 4.8: **Developer blocking vs. takedown category–wise (%).** Each figure shows the country distributions and outliers in each category (with their total apps) for takedowns on the left, and developer blocking on the right. KR is a frequent outlier in takedowns, while IR is the most developer-blocked in all categories. Categories like FOOD, SHOP, and LIFE tend to be naturally region-specific and thus blocked more.

has the most influence on geoblocking in the mobile domain. 2,419 (44.9%) unique apps are developer-blocked in at least one country (e.g., `free.vpn.unblock.proxy.turbovpn` in Hong Kong (HK), `com.google.android.apps.books` in Israel (IL), `com.twitter.android.lite` in the US). Consistent with prior web geoblocking study [164], IR is the most developer-blocked country with more than 50% blocking in eight categories and is the top outlier in every category.

While overall, countries within the same region (and hence, cluster) tend to have the same apps geoblocked, a closer look within a region shows different amounts of developer-blocking. For instance, the EU countries, UK, Germany (DE), Ireland (IE), and Hungary (HU) have 559, 580, 666, and 725 apps blocked, respectively. Examples of apps that are blocked differently in the EU countries are Paypal's `com.izettle.android`, which is blocked only in IE and HU, `com.lego.catalogue.global` that is blocked in the UK, IE, and DE, and `com.google.android.apps.walletnfcrel` and `com.yahoo.mobile.client.android.search`, both of which are blocked only in HU. Such region-specific differences in blocking may be a result of either local laws or consumer market segmentation for business.

We find 8 apps that are developer-blocked only in our four EU countries, possibly due to GDPR legislation. For instance, the news app `com.usatoday.android.news`, the largest gay social networking app, `com.blued.international`, and `com.ebates` are developer-blocked only in all four EU countries. While Facebook's Messenger for Kids is known to be geoblocked in our EU countries due to GDPR compliance issues [242], we find that it is blocked in other countries like Ukraine (UA), Israel (IL), HK, RU, and KR as well.

Certain countries have more developer-blocking in certain categories. For instance, the African countries in our study— KE, ZW, TN, and EG— appear in seven out of the top 10 developer-blocked categories. JP has the most developer-blocking in VID and EDU, RU in COMM, and HU in NEWS apps. Consistent with prior research on the web [164], SHOP and FOOD apps are the most developer-blocked everywhere, possibly because these apps are often region-specific and involve financial transactions in local currency. However, contrary to that work, we find that EDU and MED are the least developer-blocked. Figure 4.8 (right) shows the distribution of developer-blocked categories and the outliers in each.

Countries also vary in the developer-blocking of the special-interest apps (security, privacy, ad blocker, crypto, and VPN) we study. The top three countries that have the most developer-blocking of these apps are IR, ZW, and HK with 298, 79, and 78 apps blocked, respectively. While prior research has noted high blocking of VPN apps in RU [256], we find HK and TR (25 and 17 apps) have higher blocking of VPN apps than RU (15 apps). The higher blocking in HK and RU of VPN apps is consistent with the recent upsurge of surveillance laws there [124, 247].

Although our data consists of free apps, some support in-app purchases that may cause US sanctions to play a role in geoblocking in embargoed countries like IR, ZW, and Venezuela (VE). Amongst the embargoed countries, IR has much higher blocking when compared to ZW and VE. While Google prevents in-app purchases in IR, they are supported in VE and ZW [105]. Despite this, we find that developers disable in-app purchases in all apps in VE and ZW.

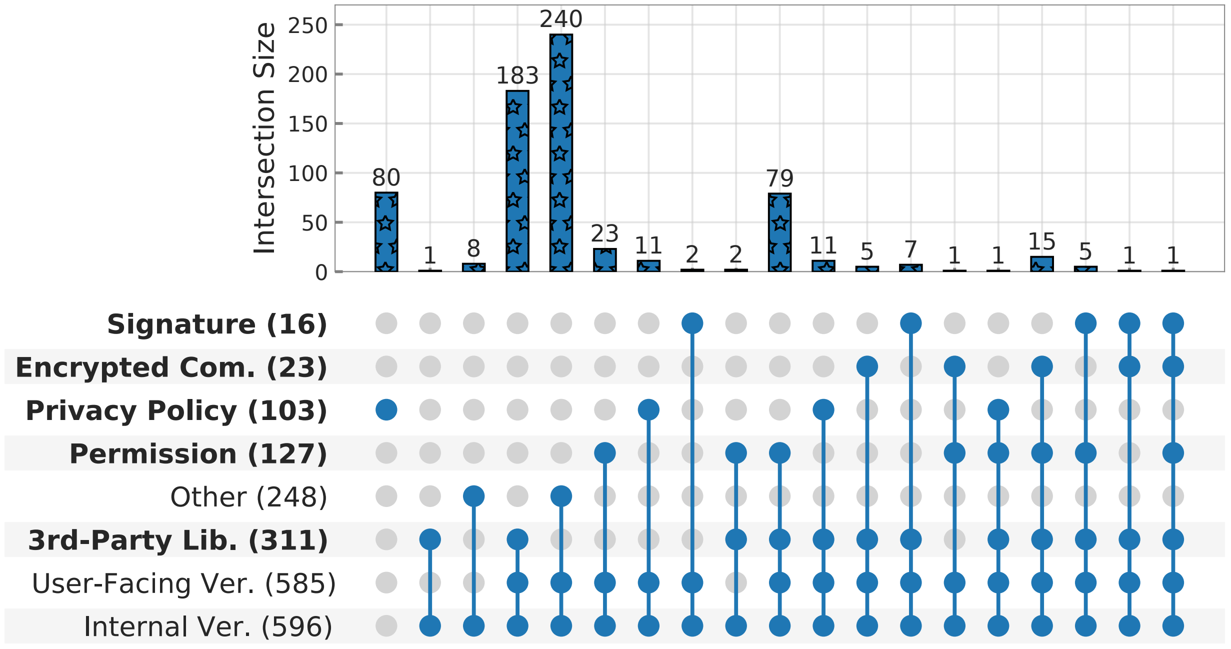


Figure 4.9: **Geodifferences in App Features.** The app feature and the number of apps with geodifferences in the specific feature is shown on the left. The number of apps with intersecting geodifferences (indicated by the dark circles) are annotated above. ”Other” indicates features outside of the security and privacy related features in bold.

4.5.3 Geodifferences in Security and Privacy

Until now, we studied the phenomena of geoblocking in the mobile app ecosystem. In this section, we go beyond and ask whether the apps available in more than one country have geodifferences that lead to differences in security and privacy offerings to users in a region. To the best of our knowledge, we are the first to conduct such a study.

Despite using a measurement design that ensures app updates to be rare, we observe geodifferences in 596 apps as seen by a binary diff of our `apks` across countries. To confirm that the observed number of apps with geodifferences is statistically higher than the expected number of app updates, we conduct a z-test by considering each app as a sample and the presence of geodifference in an app as a binary value. With this, we define the null hypothesis as the average percentage of apps with geodifferences (11.1%, here) to be less than or equal to the average percentage of apps updated on a given day (N(1.1, 0.05) from preliminary experiments in Section 4.3). The resulting p-value of 0 is less than the significance level 0.05, which allows us to reject our null hypothesis and confirm a significantly large number of apps with geodifferences.

The presence of apps with geodifferences is consistent with Google's country targeting feature that allows developers to distribute different versions to different countries [100]. Since we expect apps with geodifferences to also have version differences, we study app versions and find that not all of these apps have differences in an app's user-facing version. While all 596 apps have differences in the internal versions, 11 apps have the same user-facing versions in all available countries[2].

Figure 4.9 shows the distribution of geodifferences in app features. We find apps with geodifferences in privacy policies (sometimes even when their `apks` have no geodifferences), permissions, signatures, settings for encrypted communication, third-party libraries, and in other app features such as assets and app components. Below, we focus on geodifferences in security and privacy related features in more detail. While overall, the geodifferences appear to be more prominent in certain countries, we did not observe regional differences related to localization.

Permissions Requested. We compare an app's permissions in all 26 countries and determine the number of additional requested permissions in each country. The number of extra permissions requested by an app a in a country is computed as $|P_{ac} - P_a|$, where P_{ac} is the set of permissions requested by the app in country c, and $P_a = \cap_{i \in Countries} P_{ai}$ is the intersection of requested permissions in countries where the app is available.

We found 127 apps that exhibit geodifferences in permissions requested. On average, the most frequently requested extra permissions are `READ_EXTERNAL_STORAGE` and `READ_PHONE_STATE`, both *Dangerous*, and `RECEIVE_BOOT_COMPLETED`, which is *Normal*. We find 49 apps that request *Dangerous* permissions only in certain countries. For instance, the app `com.fun.top.video` has two *Dangerous* permissions (`ACCESS_BACKGROUND_LOCATION` and `CAMERA`) only in AE, BH, IR, and TN. `com.honor.global` has one extra permission `ACCESS_FINE_LOCATION` only in DE and UK. On average, we observe apps in BH, TN, CA,

[2]There are two app versions in an app's manifest—a version that is displayed on the app's home page on Google Play, which we call the user-facing version, and an internal developer version.

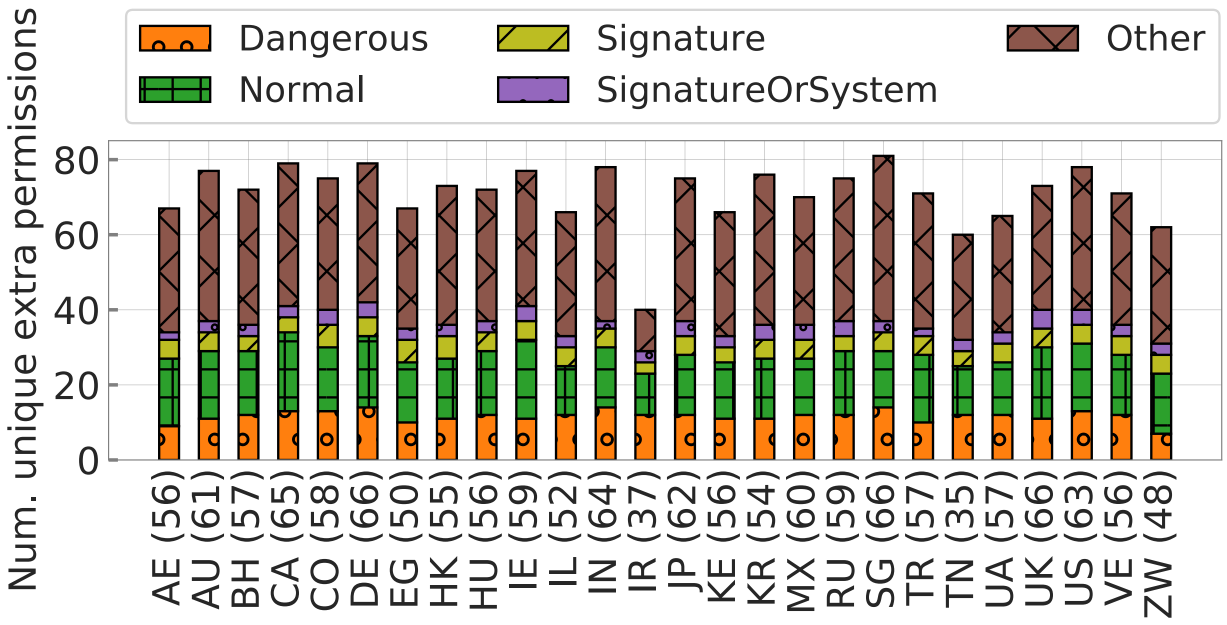


Figure 4.10: **Extra Permissions**– The Y-axis shows unique number of extra permissions requested of each type. X-axis paranthesis shows # apps requesting extra permissions in each country. Most apps only request a handful of extra permissions, the median being four. "Other" indicates custom or vendor specific permissions

and DE to request the most extra *Dangerous* permissions (e.g., `READ_EXTERNAL_STORAGE` and `READ_PHONE_STATE`). Figure 4.10 shows the breakdown of extra permissions of each permission type by country.

We analyze app categories that request extra *Dangerous* permissions. We find that five categories request extra *Dangerous* permissions most often—LIFE, VID, DATE, ENT, and TOOL. While it makes sense for TOOL and VID apps to request *Dangerous* permissions, it is surprising that DATE and LIFE apps request extra *Dangerous* permissions, especially only in certain countries. For instance, `com.datemyage` and `com.netatmo.camera`, request `READ_EXTERNAL_STORAGE` and `RECORD_AUDIO` permissions in only 11 and 3 countries, respectively.

Third-party Libs. Similar to how we compute the number of extra permissions requested, we compute the number of extra third-party libraries in an app per country, focusing on ad trackers. We found 118 apps with additional ad trackers, with the top five most included ad trackers be-

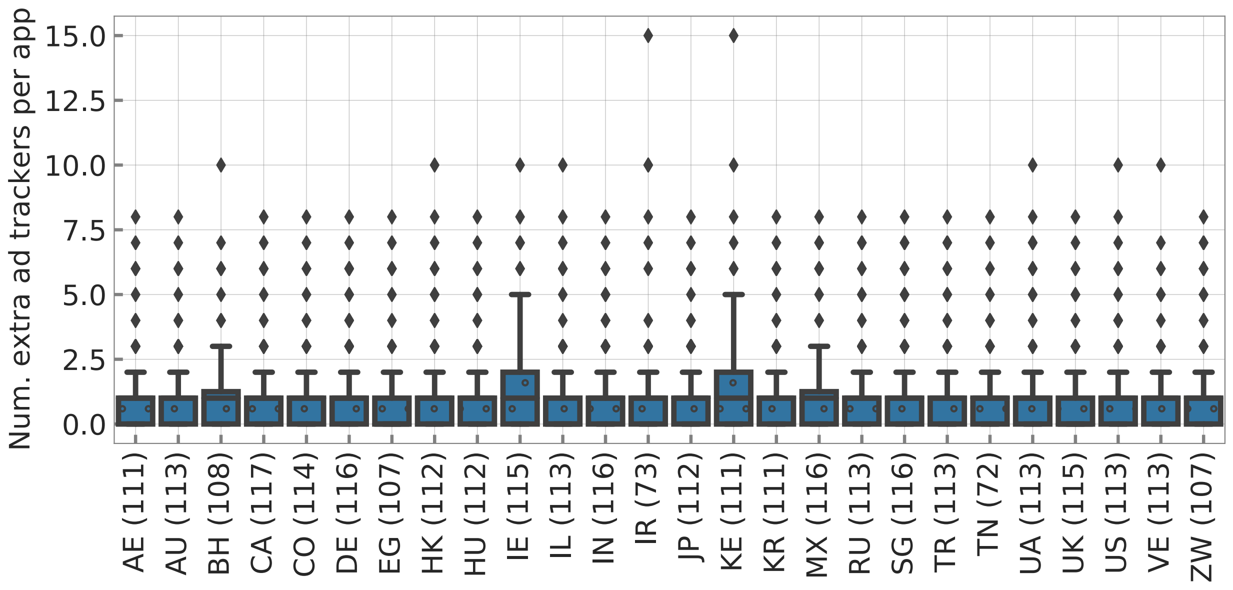


Figure 4.11: **Extra Ad Trackers.** The number of apps with geodifferences in ad trackers per country is shown in parenthesis on the X axis. While the majority of apps only include one extra ad tracker in certain countries, there are outliers present in all countries, with one app including 15 extra ad trackers in both IR and KE.

ing Integral Ad Science, Moat, and Facebook's—Analytics, Places, and Share. On average, IR has the most extra ad trackers, followed by KE and UA. Figure 4.11 shows the geodifferences in extra ad trackers per country. There are outlier apps in every country, notably in IR and KE where the app `com.outfit7.movingeye.swampattack` includes 15 additional ad trackers. `com.uc.vmate` and `com.mapfactor.navigator` request 10 additional ad trackers each, the former in IE and UA, and the latter in BH, HK, IL, KE, US, and VE. The median number of extra ad tracker per country is one (e.g., `com.shareitagain.bigemoji` includes `AdColony` only in DE and IN). The top categories with the most extra ad trackers are GAME (e.g., `com.outfit7.movingeye.swampattack`), ENT (e.g., `com.graymatrix.did`), and SOC (e.g., `messenger.chat.social.messenger.lite`). Overall, on average, TN, AE, and UA have the most extra third-party libraries.

Encrypted Communication. We find 23 apps that selectively use unencrypted communication settings for some countries. For instance, `com.honor.global` (which has two additional

Dangerous permissions only in UK and DE) uses encrypted communication only in DE and UK. Three of the apps that have geodifferences in this communication setting are VPN apps. For instance, `com.vpnproxy.connect` uses unencrypted communication only in UK, TN, and VE.

Signature Algorithms. We find that 16 apps have geodifferences in the signature algorithms used, suggesting that some apps in certain geolocations are at a higher security risk from using weak signatures. For instance, `com.thomsonreuters.reuters`, which is available everywhere except TN, is signed with just the signature scheme V1 in 16 countries, including the US, CA, and DE, and uses the V2 scheme in other countries. `ca.bell.selfserve.mybellmobile` uses V1, V2, and V3 signature schemes everywhere, except in JP and MX, where the app does not use the V3 signature scheme. This is against Google's recommendation of signing the apps with all three signature schemes for maximum security [16].

Privacy Policy. We find 103 apps with geodifferences in privacy policies. Our data shows that regional legislation such as the GDPR in the EU and the CCPA in the US [132, 224] actually have a positive influence on app developers. However, we also find that countries not covered by CCPA or GDPR have a higher privacy risk. For example, 71 apps from Google have additional clauses to comply with GDPR only in our EU countries and for the CCPA only in the US. While the US policy of `gbis.gbandroid` was updated recently in 2020 to comply with CCPA, the privacy policies in countries with older legislation (e.g., AU - Privacy Act 1988) were last updated in 2017 [73]. Norwegian company Opera has six apps, including `com.opera.browser`, with policies that use two different data protection standards: European in the European Economic Area and Singaporean in other countries.

We find instances of privacy policies that fail to download even when the apps themselves are available. Privacy policy downloads of 37 apps returned 403 Forbidden errors (at least once in all countries), and for another 20 apps, returned explicit server-side blocking error pages (at least once in 21 countries). Notably, 12 policies hosted on Google's App Engine (GAE) did not download in IR because GAE is blocked due to embargo rules there [161]. We could not download the privacy policy for `org.openobservatory.ooniprobe` (hosted on ooni.torproject.org) in TR, where Tor is known to be blocked [190] and `org.telegram.messenger` in IR, where Telegram is blocked [92]. Surprisingly, we received the IR government blockpage for policy URLs of four apps (e.g., `com.geekslab.applockpro`).

While we expect privacy policies to differ across regions when apps request different *Dangerous* permissions, we found 28 apps with identical privacy policies despite having geodifferences in requested *Dangerous* permissions. Additionally, not all privacy policy URLs point to a valid policy page as required by Google. For example, the policies of the apps `com.jagbani` and `net.bitburst.pollpay` link to a company website and to a deactivated page, respectively.

4.6 Limitations & Future Work

Our work is the first large-scale data-driven study into the prevalence of geodifferences in the mobile app ecosystem. We chose to look at the popular apps where we expected developers to put effort to maintain the apps. Any geodifferences observed in these apps greatly impact users because of their popularity (and we found many). There may be region-specific trends that can be discovered by further studying the long tail of less popular apps on Google Play.

While we contribute significantly to characterizing geodifferences and who is responsible for them in the mobile app ecosystem, we are limited in our study on why these geodifferences exist. Prior work on web geoblocking summarizes the motivations for geoblocking to be one or more factors such as data protection law, economic sanctions, revenue, national security, political censorship, or unintentional [249]. They emphasize that distinguishing between these motives is hard given how they are interdependent on each other. Thus, future work could include focused investigations to distill the reasoning i.e., from financial to data protection or others, and their impacts on users, or study popular regional apps itself to understand that. Currently, we only examine an app's static features, and future studies could use results from runtime behavior analysis of apps within a region.

Though our study is focused on Google Play, future work could do a broader study that includes other app markets. Our study is limited to 26 countries and 5,684 popular apps. While we consider our choice of countries reasonable for a first geographic study, additional countries could provide more insights. We use the FHIF score as a metric for country selection following prior work on the web but acknowledge a potential bias in the metric [226]. Hence, we do not draw any large conclusions based on the metric.

4.7 Discussion

We focus on Google Play because it is the largest and the most accessible app market. Our findings suggest differences in app equity as a result of geoblocking and geodifferences in apps. We find certain countries experiencing more blocking than others, alarming cases of the same apps asking for different permissions in different countries, and overt violations of privacy disclosures in some countries, to name a few. While Google has taken some steps towards a transparent ecosystem [13], the geodifferences that we observe are not generally known due to lack of research in this space. Our work highlights the shortcomings in Google's auditing of the app ecosystem.

There are a few steps that Google and other global app market proprietors (e.g., Amazon Appstore) could take to address some of the issues we find. For instance, Google could consider pushing for transparency from developers to specify regional differences in app features, including permissions, ad trackers, and privacy policies. Google could also do better testing for an app's compliance with its existing guidelines (e.g., on privacy policy and signatures) across countries since our study discovers both non-compliance and geodifferences for the same app. Considering that auditing external links are hard, Google could host an app's privacy policy themselves to track policy changes. Google could provide an app's release history (as provided by some app markets like APKMirror and Apple's AppStore), which could help audit developer behavior in the ecosystem.

Developers on Google Play have fine-grained access to country and device targeting features.

Prior web geoblocking studies have shown how an unrestricted country targeting feature provided by Cloudflare, a web CDN, led to significant blocking, isolating certain countries more than the others [164]. In 2018, Cloudflare reverted to its older business model that limited country targeting only to their enterprise customers. In the mobile app ecosystem too, we note certain countries (e.g., Iran) that are more isolated than the others, even though the apps we study are all free apps and popular apps. We believe that some of the geoblocking is partly because developers have unrestricted access to country targeting features on Google Play. Global centralized app market owners like Google can prevent exacerbating this divide.

Recently there have been several instances of governments relying on Google Play to ban content. In June 2020, the Indian government banned 59 Chinese apps [241]. Per our longitudinal metadata, Google took down the apps in India on July 2, 2020, less than a week after the government issued a public notice. A similar pattern emerged in the US when the government threatened to ban Chinese apps like Tiktok [232]. These were high-profile takedown threats. But, our work shows that takedowns can also happen without much public knowledge or debate. As a platform owner, Google could provide better insights on why they removed each app in its transparency reports [101] and clearly disclose the reason an app is not available in a country on the app's homepage. Detailed transparency reports coupled with informative error messages (as in the web ecosystem) will enable researchers and third parties to audit the app market for app equity and for citizenry to be better informed on reasons for unavailability.

Potentially as a result of Google's policies in some countries, regional app markets are becoming increasingly popular. For instance, Google's approach towards Iranian apps and developers [85,243] has likely contributed to the popularity of local app markets like Cafe Bazaar in Iran. We find that some apps that are developer-blocked on Google Play are available on Cafe Bazaar, potentially driving users to the local app market. What is concerning here is that these local app markets may not necessarily have adequate app vetting policies. Moreover, app market owners may provide altered versions of apps to users; a user has no straightforward way to distinguish an altered app from a legitimate one.

4.8 Related Work

Geoblocking has garnered much attention from regulating organizations in recent years. In a bid to curtail discriminatory practices by service providers through geoblocking, the EU commission introduced regulations in 2017 to prevent unjustified geoblocking, and foster an unfragmented digital market [81]. A study by the Australian parliament in 2013 found exorbitant prices in Australian markets due to geoblocking and concluded that geoblocking should be regulated [32].

Prior research has studied geoblocking in the web ecosystem. Tschantz et al. conducted a preliminary study on the motivations for server-side geoblocking and showed that root-causing the reasons for blocking is non-trivial [249]. McDonald et al. performed a wide-scale measurement study on geoblocking by customers of web CDNs by using Cloudflare's CDN as a case study [164]. Afroz et al. used a combination of automated page loads, manual checking, and traceroutes to confirm geoblocking of developing countries [2].

Geoblocking by mobile app markets has been studied at limited scale. Ververis et. al [256] looked at censorship of 11 censorship-circumvention apps in Russia and China by querying public search engines of app markets (Google, Apple, Tencent). In a recent study on digital filtering in Saudi Arabia, authors noted an increasing availability of 18 mobile apps from none in 2017 to 93% in 2019 [5]. Some studies on Apple's and Google's app markets examine app admissions and removals to understand their reasons for such decisions [27, 122, 152, 159, 262], but do not discuss geoblocking or geodifferences in app features.

Prior research has looked at finding privacy policy violations by analyzing policy texts and apps [10, 11, 269], characterizing and querying privacy policies [119], and studying differences in the policies of free and paid apps [117]. Sun et al. showed that policies generated by automated privacy policy generators (APPG) are often incomplete [230]. Longitudinal studies of web privacy policies show that longer policies are slow to comply with recent legislation such as the GDPR [9, 66, 153]. Shen et al. conducted a study on the types of sensitive information that leak from mobile apps [219].

Research on app markets such Google Play and Tencent has studied their app auditing processes, transparency efforts, impact of app releases, download distribution of apps, app monetization schemes, and behavior of app developers [139, 162, 163, 197, 261, 263, 271]. Lim et al. [151] conducted a large-scale survey on mobile app user behavior that affects app market downloads. Other large scale studies examined update behavior of apps on Google Play [235], update timing delay [180], update frequency [200] and target fragmentation as a result of developers targeting older Android versions [174].

4.9 Chapter Conclusion

We performed the first large-scale study of geodifferences in the mobile app ecosystem as seen by users in 26 countries. We designed and implemented a parallel, semi-automatic measurement testbed using which we collected 5,684 popular mobile apps from Google Play in our countries using direct measurement vantage points. Our data showed high amounts of geoblocking in all 26 countries. While we corroborated anecdotal instances of takedowns due to government requests, we found that blocking by developers had the most influence on the geoblocking. We also found instances of developers that release different app versions to different countries, with some apps having weaker security settings and privacy disclosures. Based on our findings, we provided recommendations for app market proprietors to address the issues we found.

CHAPTER 5

Building Secure & Open Mobile Ecosystem At Scale

Over the last decade, mobile computing and mobile Internet have evolved to dominate the traditional web ecosystem. In many economies, including in the US, mobile phones have become the primary, and for some, the only medium for accessing the Internet [48, 258]. The multi-geo platforms we study in this thesis, like Google Play and UPI, have enabled tech inclusion and free and open access to mobile services, fueling mobile growth and tech inclusion worldwide. We conducted a security analysis of these ecosystems, one in-depth, from within the confines of a region where it was first released, and the other from the vantage point of 26 diverse countries. Our study shows the following:

1. There continues to exist critical and fundamental violations in security and privacy offerings of the platforms even when seen from the vantage point of highly-rated apps from an official app market like Google Play. These vulnerabilities are beyond the scope of what users can address. While these systems are deployed with security and privacy in mind, in reality, decisions made by platform owners fail to secure users in both the ecosystems we study.

2. Our experience with app disclosures reveals that many of these vulnerabilities result from design decisions made by these platform owners and are left behind knowingly. As a result, many of the issues we find may take months, if not years, to resolve leaving users unprotected for a prolonged period. Vulnerabilities are addressed only at the discretion of the app vendor or platform owner. Users may never even get a security patch.

3. Despite there being Google's recommendation for app developers for secure app development, we find that security and privacy implementations are subject to developer discretion. Our study of India's UPI ecosystem shows how different apps have different security and privacy implementations that enabled us to leak the internal handshake of a widely used payment protocol.

4. We show how attackers can leverage region-specific vulnerabilities with app vulnerabilities to launch attacks at scale. For instance, for attacks on UPI, we (as an attacker) leveraged the fact that India is on the list of countries with (i) the most potentially harmful applications (PHAs) pre-installed on phones, (ii) geodifferences in device behavior, and (iii) users sideloading apps from unofficial app markets. We leveraged app differences with these region-specific vulnerabilities to launch attacks. We also leveraged older versions of apps available on third-party app markets to reverse-engineer internals of UPI.

5. We show how the same app, when seen from different countries at the same time, has weaker security and privacy offerings and exposes users in some countries to higher security and privacy risks. This raises concerns about how an attacker can potentially exploit these app differences to launch attacks.

6. Our research shows how effective control over security and privacy is impossible to achieve when left to the decisions of platform owners alone. While security and privacy policies exist, the controls that enforce these policies are insufficient.

In this chapter, we discuss our experiences conducting large-scale investigations that evaluate the security and privacy offerings of these two large mobile ecosystems. We disclosed all the vulnerabilities we discovered in the platform and in the apps integrated with it to the platform owners. Below we first discuss our experiences with app disclosures that further gave us insights into how the policies of the platform owners affect the different stakeholders of this ecosystem. We then provide recommendations to large platform owners to address the issues we find. While these recommendations are made based on our understanding of UPI and Google Play, they apply to any platform owner wanting to build a secure and open mobile ecosystem at scale.

5.1 Experience with Vulnerability Disclosures

We disclosed all the vulnerabilities and potential attacks to Google and UPI. In our study on UPI, all the vulnerabilities in the payment apps we discovered were disclosed to the payment app vendors. We obtained the following CVEs: CVE-2017-9818, CVE-2017-9819, CVE-2017-9820, CVE-2017-982, CVE-2018-15660, CVE-2018-15661, CVE-2018-17401, CVE-2018-17402, CVE-2018-17403, CVE-2018-17403, for our disclosures on payment apps from CERT-US. Barring an acknowledgment of our disclosures, we received no further communication from CERT-IN, indicating a difference in how vulnerability disclosures are dealt with in different regions.

Based on our study on UPI, we reported the privacy leaks, the potential attacks that exploit their multi-factor authentication workflow, and our pre-publication version of the paper over the 24 months that this research spanned. Below is a response we got from the app developer called PhonePe, which is integrated with UPI's backend infrastructure and has over 46% market share for UPI transactions in India today. The response is specific to the publication draft we shared with the payment service provider in June 2020.

> "In both the cases Victim User has been exploited enough to make him/her willingly install the attacker/malicious app and has given permissions to read sms/Accessibility/Overlay Permissions in order to carry out the attack. Please note, that all three of these permissions are explicitly given permissions and are not granted by the platform by default. Even after all of this, it will affect the individual user and not the platform in general.
>
> We do, however, accept that this is a 'Risk' of using OTP based authentication flows inside Android Platform. Unfortunately though, we only control the security inside the boundary of our application, we do not and can not control the victim's device com-

> promise. We have multiple layers of security to ensure the veracity of the transactions. This includes, our real time fraud checks, device linking checks (as you've noted in your paper already) and multiple other layers of security to reduce this risk as much as possible. Additionally, we have enabled the Android platform provided biometric + pattern based auth as a standard feature across all users. We perform regular user awareness programs to make user aware of these risks and keep their devices safe from attackers as well."

Similary, in our study on Google Play, we also reported the geodifferences in apps in May 2021, specifically those with weaker security and privacy settings in select countries, to Google Play's Security Vulnerability Program. We received the following response from Google.

> "As previously mentioned, this behavior is not in the scope of our program. For this reason, we will be closing this report, but we still want to thank you for all your efforts here."

Though our disclosures on these apps were outside the scope of Google Play's vulnerability discovery program, our findings from this work on geodifferences in the app ecosystem were acknowledged at the highest levels of Google's Privacy team. As of writing this thesis, we have observed a few positive changes in line with our recommendations. For instance, as was our recommendation, Google has begun hosting privacy policy summaries in a user-readable format on the landing pages of these apps to ensure that: (i) users will not have to navigate to external web pages to view an app's privacy policy, and (ii) users can find out what data will be collected before downloading an app. As of April 2022, Google has mandated developers to include a privacy policy with their apps [57]. Though this comes as a surprise that such enforcement did not exist, Google is moving in the right direction to secure this space.

What is common to both disclosures is that the platform owners pushed back on our findings during our first contact with the issues we found. For the same reason, getting these issues to be acknowledged as vulnerabilities that had significant impacts required months of deliberation and navigating an ecosystem of app vendors and platform owners that lacked accountability for their design choices. While there are vulnerability and rewards programs that bug bounty hunters could approach for filing vulnerabilities, convincing the platform owners for the impact and significance of the types of vulnerabilities we found (which are more of design choices) and subsequently achieving a resolution for them may take several months or even years, if at all.

Disclosures are challenging, especially when the research is carried out across nation-state boundaries, as we are potentially subject to legislations of other countries and threatened by nation-state actors or regional app vendors. For security researchers to conduct research on ecosystems that are outside of their jurisdictional purview will require additional safety measures. For instance, during vulnerability disclosures, we encountered situations with hostile app vendors that threatened us with a legal lawsuit for defamation, despite having sent them multiple disclosures on the vulnerabilities in their app prior to public disclosures. Also, we were coerced into taking down vulnerable versions of their app that we had hosted on our GitHub repository, even though these versions are available on third-party app markets by independent publishers. Given how India is moving towards an economy that is increasingly lacking freedom of expression, we faced other types of pressure that dissuaded us from going ahead with the publication.

As can be seen from PhonePe's response, the resolution of the attacks we observed rests largely on the user's shoulder. A user is expected to not install a malicious app even though India's in the top three countries with the most number of potentially harmful apps pre-installed. A user is expected to know what permissions are dangerous and not grant access to them, even though the vast majority of dangerous permissions are vital to the functioning of most of the highly-rated apps. These platforms use OTP-based authentication flows despite knowing that they are vulnerable (as is in PhonePe's response). PhonePe's response to our disclosure also shows that a user exploited via social engineering attack tactics is outside their threat model. ***Thus, in an ecosystem where the user has no control over the security and privacy decisions taken by platform-owners and app developers when building their devices, operating systems, apps, and the networks they use, the onus of a user's security lies entirely on her ability to do the right thing.***

5.2 Why Should Platform Owners Re-think Their Policies?

For platform owners like Google, the business of apps is a very profitable one that generates billions of dollars in revenue; in 2022 alone, Google made over $30 billion in revenue [186]. By democratizing app development, platform owners have empowered citizen developers to release apps worldwide at a nominal cost. This has also spurred an entire ecosystem of app developers trying to monetize their apps, many without adequate knowledge of best practices in security and privacy, regional content and data protection regulations, or the evolving threat landscape with powerful and globally colluding threat actors. Similarly, by encouraging several apps to compete on its shared platform, UPI has also enabled an ecosystem of payment app developers (both third parties to UPI and banks) that have integrated with their shared backend infrastructure. Because these are centralized platforms with wide reach, our research has shown that any interferences here or differences in security and privacy offerings can significantly impact millions of users.

Previous research has conducted several literature studies focusing on digital platform owners' governance and strategies to generate profit and preserve market dominance [49, 97]. Prior research has also shown how inaccessible content significantly prevents Internet uptake in emerging economies. Mobile platform owners (e.g., iOS, Android) exercise control over the various players in the ecosystem—for instance, they regulate developers by choosing to open their programming interfaces selectively and by forcing developers to use their interfaces; they regulate market trends by setting policies of use on the apps integrated with their platform, effectively also controlling how users consume and interact with these apps. Because these apps are made available for download on these large platforms, users trust them. But our research shows that there is a lag in how these platform owners regulate the security and privacy offerings of these apps by failing to regulate and set a bar for how app developers follow best practices when integrating with these platforms. Our research also shows significant geoblocking in the mobile ecosystem, which can significantly deter Internet uptake in certain countries.

Prior research has shown how platform owner's policies have affected not just a few individual users or developers but have effectively isolated entire regions as such [58, 87, 135, 243]. Similarly, by enabling developers to target countries with apps with weaker security and privacy settings, platform owners have exposed millions of users to security and privacy risks. Based on our empirical study, we present an opportunity for the mobile research community to re-examine the policies of platform owners and, by extension, the app developers that use these platforms. We provide several recommendations for platform owners that we believe, if implemented, can secure and FOCI the mobile ecosystem at scale. While our experiences with conducting this empirical research are in the context of Google Play and UPI, our recommendations can also be applied to other multi-geo ecosystems.

5.2.1 Platform Policies Drive Harmful Developer Behavior

Google Play provides extensive country and device targeting abilities to developers . Through their content delivery networks, global platforms like Google Play give developers easy access to country-target and device-target their applications. This is despite countries taking a state-centric approach to secure their network boundaries. Developers can provide different app versions to different regions and have granular control over the app version that will be available for download based on the specifications of a user's device (such as the device's make, model, and RAM) or the cellular service provider. Such capabilities enable developers to target entire sections of users based on their geolocation, device features, or cellular service provider. By granting such unmoderated capabilities to developers, platform owners are further fragmenting the Android ecosystem and enabling the splintering of the mobile ecosystem.

Platforms take no responsibility for the apps (product) or their content. While platform owners have democratized app development and also profited from it, the onus of developing apps in keeping with security best practices and data and content regulations of hundreds of countries worldwide is entirely the responsibility of app developers. Per Google's developer policy [109]:

> "Google does not undertake an obligation to monitor the Products or their content. If Google becomes aware ... then Google may reject, remove, suspend, limit the visibility

> of a Product on Google Play, or reclassify the Product from Google Play or from Devices."

Similarly, even though the NPCI has enabled UPI by default for all users without their consent, a user that is defrauded because of using UPI is not under their purview, as can be seen from NPCI's terms and conditions [169].

> "The user agrees and acknowledges that NPCI shall not be liable and shall in no way be held responsible for any damages whatsoever whether such damages are direct, indirect, incidental or consequential and irrespective of whether any claim is based on loss of revenue, interruption of business, transaction carried out by the user, information provided or disclosed by issuer bank regarding user's account(s) or any loss of any character or nature whatsoever and whether sustained by the User or by any other person."

That apart, per the UPI specifications, though UPI claimed to support three factors of authentication, of which one of the factors was optional and to be enforced by a third-party payment service provider (app), UPI does not take responsibility for whether or not these apps that integrated with their platform choose to use the third-factor or not.

Highly-rated apps released with geodifferences in security and privacy to users of certain regions. Given the lack of sufficient controls in Google Play to moderate developer behavior, app developers exploit holes in the policies of Google to distribute malicious and potentially harmful apps worldwide. While this is obvious from the continued influx of malicious apps that millions of users have installed through Google Play [202, 270], our study also reveals alarming geodifferences in security and privacy of globally popular apps based on a user's geolocation. For instance, we found that developers released different app versions to different countries, with apps requesting additional dangerous permissions, enabling unencrypted communication, and having different privacy policies depending a user's geolocation. While apps are vetted for maliciousness, geodifferences in the security and privacy of globally popular apps and the reasons for such differences is not well understood for lack of research in this space.

Missing developer information and fallacious claims about apps. While in our study, we attempted to correlate app geodifferences we observed with developers' countries of origin, we could not do so for lack of information about developers for many apps on Google Play. For instance, many app developers do not have valid addresses and use non-business email IDs for their contact information. This prevents auditability of the app market and traceability of offending apps to the corresponding app developer. We manually tested a random sample of app developer information on both Google Play and Apple Store, and we found that this behavior was more pronounced on Google Play.

Our study also find apps that make false claims about their availability and security features. For instance, we found that Zoom on their app landing page on Google Play has noted that their app is unavailable in Iran and Ukraine [273], even though the app was available via Cafe Bazaar in Iran and via Google Play in Ukraine. The same is true for a few apps developed by Google that we downloaded from Iran. Even though the install button for these Google apps were unavailable on Google Play, we could download the app from Google's Play store in Iran by directly accessing the download link. Similarly, UPI claimed to have three authentication factors, but the workflow that used all three factors only existed for a user registering with UPI for the first time. All other workflows had a lower bar and did not even require strong secrets for authentication.

5.2.2 Lack of Transparency

Platforms takedown apps without developer knowledge. Our study shows that governments take a state-centric approach to allow access to mobile apps, and many app takedowns occur without public knowledge or debate. For instance, in South Korea, one of the most interconnected countries with the largest smartphone penetration, we found government-directed takedown of popular gay dating apps like Jack'd. Public reports on this corroborate that the takedown happened without the developer's knowledge [80]. Such government-directed takedowns occur because of regional content and data regulatory differences that app developers may not have knowledge of.

Besides government-directed takedown, we also found instances of Google taking down apps even when they do not necessarily violate the platform's policy. For example, in early 2020, Google took down a few popular crypto wallet apps such as *Metamask* and crypto news apps such as *Cointelegraph* from the Play store without notifying the developers, citing ”deceptive services” as the reason for their removal [58, 63]. Though Google's developer policy disallows crypto apps from mining on a user's device, neither were mining apps. Google claimed to revert the takedowns later [87], but our study found that Metamask was unavailable for a while even after.

Google also has a history of takedowns of Iranian apps and erratic policies toward Iranian developers. In 2017, Google took down the apps of several Iranian developers [243]. Recently, Google also took down *AC19*, the Iran government's app for Covid-19, citing the app as spyware, despite reports otherwise [135]. Below is a quote from an Iranian developer on Google's support forum that corroborates Google's policy towards Iranian developers [112]:

> "Google has been denying Iranian developers access to developer resources in a wide range of services including Android resources (Android docs website and maven repos), Google cloud (including gcr container registry), etc.
>
> Please note that Internet access is already messed up and restricted in Iran. So, even if we try to access these services using VPN software, it will be very difficult, because most VPN connections are blocked by the government. On behalf of Iranian developers, I urge Google to take action and remove such restrictions for developers, most of whom are students, enthusiast individuals and freelancer developers that need these resources to learn (Android, Kubernetes (Minikube), Cloud, etc.) or make a living.
>
> As a side note, there has been a recent development on the side of US government that apparently eases some of these restrictions (if they have been the result of US government policies in the first place). I am not sure this is the right place to discuss this topic. But, we Iranian developers really need to be heard. Please help us."

This further corroborates our findings on how certain countries are isolated due to Google's policies. Developers in some countries need guidance on how to gain access and, if having access, how to retain their apps on the app market without incurring losses. The alternative for them is to publish apps in alternate app markets that do not have the reach Google Play has, and thus less earning potential as well.

Users subject to platform policies without knowledge or consent. Large platform owners can regulate mass trends in the ecosystem, significantly impacting a user's security. For instance, when the Indian government released UPI in India, users were enrolled for mobile transactions via UPI without prior consent or knowledge. This opened a huge security hole wherein attackers could empty a user's bank account even when they did not use a UPI app. To date, UPI does not allow users to unenroll from their platform. Similarly, by using apps on Google Play, users are automatically subject to invasive profiling by apps without the user's consent (for lack of proper privacy policies, as our study shows), which Google has full knowledge of. However, Google's lax policies may stem from the fact that their revenue comes from ad trackers that collect user data. Users are in the dark about how much of their data these platforms have access to.

Researchers encounter opaque and ambiguous signals to measurements. During our download crawls of apps from 26 countries, we received ambiguous and undocumented responses to our download requests in case of download errors. Root-causing these server-side errors required significant amounts of triaging with control experiments to map these errors with actual phenomena causing these errors. While the web ecosystem has standard errors to identify failures, Google provides no such transparency to facilitate audits of their mobile ecosystem.

5.2.3 Rise of Alternate Platforms That May Exploit Users

Opaque platform policies drive developers to alternate app markets. As a result of Google's unmoderated country-targeting capabilities (Section 5.2.1) and lack of transparent practices (Section 5.2.2), developers have had to find alternate app markets to host and distribute their apps. For instance, our study found that in Iran, many apps that are developer-blocked on Google Play are available on Cafe Bazaar, a local app market, possibly also because of Google's policies towards Iranian developers [44, 243], thus driving users to the local app market. There are also third-party app markets like APKPure and Aptoide that provide developers with an alternate channel for app distribution. These app markets may have inadequate (or no) security vetting of apps enabling them to supply altered versions of apps to users.

A recent Google regulatory filing has confirmed that the European Union's competition division is looking into how it operates the Play Store [237]. A similar regulation was recently passed by the Indian government's antitrust division to curtail Google Play's anti-competition practices and to foster the inclusion of alternate app markets [56]. These regulations will enable users to sideload mobile apps from alternate app markets. This comes with a caveat from Google Play that their platforms are potentially at security and privacy risks by sideloading apps, besides the concerns around lack of app vetting in these app markets.

Regional platform-owners have leveraged overseas user behavior for building censorship apparatus. Users are increasingly mobile, causing regional apps to diverge into overseas markets. Regional apps are no longer confined to a region. Users may either use VPNs to access apps that are unavailable in their region or install them locally and then continue to use them internationally once installed on their phones. Prior research has shown how platform owners of regionally popular apps such as WeChat and Tencent have exploited their international user base to study the usage patterns and behaviors of non-regional users to influence their regional policy decisions and strengthen their censorship appartus [54]. For instance, files and content shared between non-Chinese user bases are examined for content deemed politically sensitive in China and then used to train their classifiers for censorship. By being centralized and operating outside the purview of regional jurisdiction of countries like China and Iran that have sophisticated censorship apparatuses, global platform owners can better protect users.

5.3 What Can Platform Owners Do?

Our study has found that the vast majority of inequities found in the mobile app ecosystem are driven to a large extent by erratic policies of platform owners that have led to erratic behavior from app developers as well. We discuss below the steps platform owners could take to address the issues we bring out in Section 5.2.

Design with security, build with end-to-end evaluations. Our study has shown that despite being designed and developed with security and mind, fundamental flaws plague the ecosystems driven by two of the world's largest platforms. While designing with security and privacy in mind is essential, even more, significant is carrying out evaluations of these ecosystems from an end-to-end perspective. Our work highlights how evaluating these ecosystems from multiple vantage points can help uncover previously unseen vulnerabilities, especially when these systems scale across nation-state boundaries or when region-specific inequities are factored in.

Provide barriers to country-targeting by developers. Besides our research in this domain, prior web geoblocking studies have also shown how unrestricted country targeting feature provided by Cloudflare, a web CDN, led to significant blocking, isolating certain countries more than the others [164]. In 2018, Cloudflare reverted to its older business model, limiting country-targeting only to enterprise customers. Similarly, in the mobile app ecosystem, we also note certain countries (e.g., Iran) that are more isolated than others, even though the apps we studied are all free and globally popular.

For one, platform owners could install barriers for developers wanting to country-target their apps. Developers could be prompted with an interactive and guided experience that takes them through a series of checklists per country, requesting evidence of compliance with regional markets. This further raises developer awareness of the regional market before releasing apps in a country. Platform owners could build region-specific checklists by leveraging their regional points of presence. Prior research has shown how Google is a long-tailed app market with a small percentage of top-rated apps. Given that, Google could start with the top apps globally.

Vet apps for unjustified geodifferences. Despite Google and UPI vetting apps for security violations, we found fundamental violations in the security and privacy of these apps that are integrated with their common backend infrastructure, effectively compromising the ecosystems' safety as a whole. This is despite the fact that operating system vendors like Google publish recommended best practices for app vendors wanting to release apps on devices that run the Android kernel. The geodifferences in the apps we observed are only from a static analysis of these apps, which, given Google's more sophisticated app vetting engines and tools, can easily be detected. Being a platform owner, Google could push back on app developers for such basic violations. For instance, we found an instance of app continuing to use the default Android debug certificate when using debug certificates is a practice that has been long discouraged.

Our study also hows how certain regions get different versions of apps during an app update. This leads us to think that while the app may be consistent across all countries in its early releases, developers may still choose to alter the apps via app updates. App updates may be less vetted than their originally released version. Our study found that some apps have over ten permissions added during an update. It is unclear how such invasive changes are communicated to the user across the different versions of Android devices and operating systems. Google Play can notify users of such drastic changes and better vet apps for unjustified geodifferences and security violations.

Audit app developers and push for developer transparency. Google could take a host of steps to regulate app developers. For instance, Google could better vet app developers on their contact information, without which a user has no path for support, given how Google does not take responsibility for apps or their content. For researchers auditing app markets, developer information helps correlate app behavior with their developer information. In addition, Google could consider pushing for transparency from developers to specify regional differences in security-sensitive app features (e.g., the app requests additional dangerous permission for compliance with regional legislation). Google could also do better testing for an app's compliance with its existing guidelines (e.g., on privacy policy and signatures) and regional legislations since our study discovers both non-compliance and geodifferences for the same app. Considering that auditing external links are hard, Google could host an app's privacy policy to track policy changes.

UPI could take similar steps to vet apps integrated with their platform. While our study of UPI was conducted in the Indian context, given how UPI now supports cross-border transactions and considering how the volume of UPI frauds has grown, UPI could audit app developer behavior and incentivize app developers to provide a contact path and adequate support to users experiencing scams and frauds.

Disclose and prevent takedowns of essential apps. To bring accountability into the ecosystem, Google could make the reasons for takedown of an essential app transparent to developers and users. To avoid a takedown, Google could provide, through interactive screens, region-specific guidelines for developers wanting to release apps to diverse geographies, given how the data protection and content regulations may vary based on the region. Also, before taking down apps, Google Play could engage with and assist developers in bridging the gaps in their apps, given how developers have paid to release their apps. Prior research on Chinese apps has shown how apps from ByteDance have evolved to become the most internationalized purely due to nation-states (e.g., US, India, Indonesia) slapping fines on them due to privacy violations [141]. While large platforms have the means to endure such penalties, small app developers often need guidance as to how to navigate through such regulatory nuances.

Incentivize developers and users. Google Play can incentivize developers to adopt best practices by offering rewards for being responsible citizen developers. Given how nation-states are regulating Google's anticompetition practices to foster alternate app markets, Google could incentivize users to prevent sideloading apps by providing better app vetting and content moderation. Google can also take the citizen's help to vet app developers. For instance, Google could offer a support and disclosure channel to report offending app developers or app behavior, thus empowering users to assist Google with app and developer moderation as well as giving the platform a chance to work with the developer before a takedown. Our experiences with app disclosures show that security and privacy violations of apps are outside the scope of Google Play's Vulnerability Disclosure Program. Corrective actions on offending app developers are only taken once there is a public hue and cry or an explicit Google or country-specific policy violation.

Adopt secure workflows. In our study on UPI, our experience with app disclosures reveals that many app vendors refused to move away from OTP-based authentication to zero-trust approaches primarily because app developers believe that OTP-based authentication is the industry standard for authentication despite several studies having shown otherwise. This is despite the emergence of usable zero-trust approaches via authenticator apps like Duo, which is widely adopted in the industry for two-factor authentication. However, large platforms like UPI can push app developers to adopt secure workflows at scale, for instance, by demanding that developers adopt new and safer practices to have continued access to the UPI platform. Similarly, Google can push app developers to provide legitimate privacy policies and vet the policies before releasing their apps through the platform (an initiative it has recently taken up [57]).

To Summarize, Global platforms are the best equipped to understand region-specific regulations and can set and regulate policies that benefit developers and users at scale. Through their vantage points across nation-state boundaries, Google and UPI can leverage their regional points of presence to provide visibility into how these regions operationalize apps, understand region-specific vulnerabilities of a user, and leverage that to strengthen the ecosystem further. For instance, Google Play can influence regions with lower security and privacy bars to up their game by enforcing GDPR-like regulations when using their platforms. Large platforms can help developers understand the variables in the ecosystem that may have a detrimental effect on user's security and privacy by asking the following questions:

1. What challenges exist that can deter users' access to apps in a region? For instance, are there local sanctions or data protection regulations affecting access to apps?

2. What challenges exist that deter access to websites hosted on certain service provider networks? This becomes crucial for platform owners that allow developers to host privacy policy links on external hosting websites (e.g., privacy policies hosted on Google App Engine (GAE) is inaccessible since GAE is blocked in Iran).

3. What are the vulnerabilities specific to a region; e.g., how prominent are low-cost device vendors in a region, or what are the most popular versions of Android in this region, or does this region have a high incidence of PHA's preinstalled on devices?

4. Is there an active censorship apparatus operating in the region, and if so, what are its implications for users, regional and overseas? How can developers protect users in a region with active censorship?

5. For global app developers, how does the security and privacy of their app compare with that of their regional counterparts? Are there best practices that can be adopted by apps of overseas developers releasing apps outside of their region to better secure users belonging to specific regions?

6. What types of social engineering attacks are users the most vulnerable to in a region? Can app developers put in additional controls to protect users from social engineering attacks?

5.4 What Can Developers Do?

Our manual analysis of hundreds of apps reveals that developers make fundamental mistakes in the design and development of their apps. There are several steps that app developers can take to address the issues we find. Below we enumerate a few of them.

1. **Follow MITRE's best practices.** App developers can follow MITRE's best practices for app developers [168]. Google also has specfic recommendations that Android developers can follow for secure app development, which includes not asking for excessive dangerous

permissions, signing the app with a secure signing algorithm, and disabling unencrypted communication, amongst others [110].

2. **Secure critical workflows with secrets**: Developers must examine data flows across different workflows of an app and ensure that they never send sensitive data in the clear to the client side. Developers must also ensure that they secure critical workflows with a strong secret. For instance, UPI never required a bank-specific secret for any of its workflows, even though UPI transactions are carried out bank to bank. We exploited this vulnerability to launch attacks.

3. **Avoid security by obscurity**: Both Google and the makers of UPI rely on obscurity in their design. For instance, Google's opaque error codes during downloads made auditing the ecosystem a significant challenge. Similarly, by not publishing and publicly vetting the UPI protocol, the makers of UPI relied on security by obscurity. In both cases, though we could successfully reverse-engineer the systems, by not publicly vetting their designs, both platforms made fundamental mistakes in their design.

4. **Factor in app (geo)differences and region-specific vulnerabilities in design.** Our work on UPI shows that weak apps integrated with a common backend interface can be exploited to leak the workflow or secrets that can be used to launch attacks even via apps that are secure. In addition, we also showed how attackers can exploit region-specific vulnerabilities to automate these attacks. In our study of Google Play, though we did not showcase any attack, given how some countries have weaker security and privacy settings in apps than others, attackers can leverage the weaker apps in the ecosystem to exfiltrate protocol data and launch attacks (just as we did for UPI). Thus, besides geodifferences in apps, developers must factor in region-specific vulnerabilities when designing their systems, given how some regions have devices with weaker security settings.

5. **Assume a fully compromised user as a their threat model.** Our experience with app disclosures reveals that small changes to a developer's threat model are enough to compromise a user completely. For instance, our work on UPI shows that a user's device that is

compromised by a PHA either pre-installed on the device or via social engineering attacks is outside a developer's threat model. However, given that a user is the weakest link in the ecosystem, a realistic threat model cannot rely on the users to do the right thing. For a user's security, app developers must assume that an attacker can compromise anything on the client side and use zero-trust approaches to secure a user.

6. **Use strong factors for authentication & authorization.** UPI uses a device's fingerprint as a factor for authentication and subsequent device authorization. However, most apps we analyzed also use device fingerprints during their initial handshake. The device fingerprint contains information about a device that can be easily collected from a user's phone. It typically constitutes the device's details, such as IMEI number and device ID, along with other attributes, such as geolocation. This information which formed the basis of device binding in UPI, could easily be extracted from any other app. Studies over the years have also shown how ad trackers have already curated this information from users. App developers must thus take caution when using data that can be (or has already been) easily harvested about a user's device.

7. **End-to-end testing of apps via multiple vantage points.** Our work shows how an attacker can leverage any stakeholder in the ecosystem to effectively compromise a user or a client-side app to eventually compromise a server. While conventional testing primarily focuses on pre-canned vulnerabilities or OWASP's top vulnerabilities, to truly secure a system at scale, developers of large ecosystems must test their systems from multiple geographic vantage points and apps from the perspective of an attacker trying to collect data about a system. Our work shows how multi-geo ecosystem does not factor in ”geo” in their design and testing. Just as we proposed for platform owners, app developers that release apps to different regions, must go beyond conventional testing approaches to carry out end-to-end testing from multiple vantage points, as our study shows significant benefits from it.

CHAPTER 6

Future Work and Conclusion

The benefit of technological advances should be something every citizen must have, yet not everyone benefits from it. While in the past there has been a debate on the necessity of access to Internet content and services, the recent emergencies from the global pandemic has shown how access to Internet is a necessity, though online access is also a necessity during non-emergencies to stimulate economic growth, for fair opportunities to work and study, to engage with government, and to exercise our political freedoms. Though tremendous technological leaps are being made, the accessibility and benefits from it are geographically disbursed inequally, if not excluded entirely.

Our studies show that access to apps changes based on a user's geolocation; so does their security and privacy protections even when using highly-rated apps from Android's official app market. While equal access to apps is a bar that may be harder to achieve given how nation-states take a state-centric approach to their networks, a high bar in security and privacy of apps is only reasonable to expect in the shorter term given how it will only protect users better. While our studies have provided several insights on security and privacy posture of these mobile ecosystems, it also identified several issues that should be addressed in future research.

In this chapter, we first identify and briefly discuss avenues of future research and exploration which we believe can provide several improvements to build a secure and open mobile ecosystem. Finally, we conclude with a brief summary of our research efforts and our previous discussion.

6.1 Going Forward

Develop automated tools for large-scale analysis. A significant hurdle for our large-scale investigations is the lack of sophisticated tools and the ability to set up test beds quickly to collect and evaluate apps. Our work showed instances of app download failures when the device specifications or the cellular service provider did not match the test setup we had. Emulators were only built for testing and debugging of apps and were not designed to be used as real-world proxies for mobile devices. That apart, our work on payment apps show that many apps detect the presence of emulators and behave differently in the presence of an emulated environment to prevent reverse-engineering of the apps. As a result, we had to confine ourselves to using a handful of mobile devices for our research, and often time resorting to manual examination or static analysis to analyze apps. Analysis thus done has it limitations as static analyzers will throttle system resources for large apps. For instance, our analysis of PayTM, a large payment app on an enterprise grade system with 128GB RAM did not complete even after 24 hours. Thus, to truly assess these apps for all possible devices and Android OS versions, we need to be able to scale using emulators that can mask themselves from these apps. The alternative is using mobile phones, which may be an expensive way to scale.

Study diverse regions. Both studies can factor in more regions in future studies. While our initial work on UPI was in the context of the Indian ecosystem, given how the pilot launch of UPI was there, future work will benefit from learning how other countries have integrated their payment infrastructure with UPI's backend infrastructure. For instance, POS systems in the UK accept UPI payments by partnering with the payment solutions provider called PayXpert. UPI is also now integrated with several regions, including countries in Asia and G20 nations, and allows users with international cell numbers to register with their backend infrastructure [42]. Future research could conduct security analyses of UPI while factoring in the geodifferences in how these regions have integrated with UPI's common backend infrastructure and the security and privacy holes as a result to secure this payment ecosystem better.

That apart, studies could examine other region-specific wallet apps that are widely popular such as Alipay and WeChat Pay in China, which carries out trillions of dollars of transactions. Alipay, WeChat Pay, Apple Pay, and Google Pay are the world's most popular payment wallet apps. Future studies could conduct comparative studies on such hugely popular wallet apps to understand their security and privacy differences, authentication mechanisms, and design that make them hugely popular while providing adequate security.

Future research could explore more countries, even for an empirical study on Google Play. While our choice of 26 countries is good for the first study in this space, future studies could also include other countries like Brazil with specific content rating regulations or countries like Cuba and Saudi Arabia, where there is a significant influence of nation-state regulations in securing their networks. We excluded Brazil, Cuba, and Saudi Arabia from our research for lack of access to reliable vantage points.

Study diverse app markets: Apple's App Store & Alternate App Markets. The studies presented in this thesis focused on mobile ecosystems that catered to Android users that could (or attempted to) download apps from Android's official app market, Google Play. We consider this a reasonable choice since Google Play is the largest and most accessible app market in the world. However, Apple's App Store is a close second, is the second most accessible app market globally, and also has the highest revenue [211]. Apple also has a significant presence in China, where Google Play does not operate. China also has other popular local app markets with massive userbases like Tencent [240]. China is an interesting country to conduct large-scale studies in, given how many of their apps (e.g., WeChat, Tiktok) have spawned global trends and have millions of non-Chinese users. Tencent also has a massive gaming user base. Similar studies can uncover further insights arising from app differences in the mobile ecosystem as well the policies of platform owners.

While researchers at *applecensorship.com* [27] have done some early work on geoblocking in the Apple ecosystem, our experience collaborating with them briefly early on shows that their data is primarily crowd-sourced and thus may be inadequate to prove geodifferences in a data-driven manner. Comparative studies on the prominent app markets like Google Play, Apple App Store, and Tencent Appstore can provide significant insights into how these platforms operate globally.

This research comes at a time when nation-states are bringing about new regulations to mitigate anti-competition practices by global app market proprietors like Google Play. This will lead to the insurgence of local app markets everywhere, from where users can easily sideload mobile apps. Google's erratic policies towards countries like Iran has already led to the emergence of local app markets there [44, 85, 135] . These alternate markets may have no app vetting policies and may also distribute altered versions of apps (potentially under pressure by businesses and governments). Thus, when users sideload apps they run the risk of getting a less secure versions of it, and a user has no way of finding out. While there are open source app markets like F-Droid, they are not the goto places for the vast majority of common users. Future research could investigate app differences of popular apps across these alternate app markets.

Runtime analysis of an app's risks based on a user's country and device. Since developers country-target and device target their apps, to truly analyze app behavior, we must be able to analyze apps in the context of a user's geolocation and device. Given the geodifferences our study exposes, an app's behavior will also differ based on where the user runs it from. Thus, an accurate assessment of the security risks arising from using an app can only be made if it can be analyzed in an environment that closely mimics a user's actual environment. This is unlike how app vetting is commonly done today. Most automated behavioral analysis engines today run the app in an emulated environment for a pre-defined time before generating a report for the app. For proper security analysis, runtime analysis of apps must be able to emulate a user's behavior from within a geolocation and device.

There are a few challenges to doing this. First, there is no straightforward way to fool an app of a user's geolocation. At the time of this research, Google Play only allowed changing the country settings on the phone once per year, making it difficult to scale. Even if a security analyst were to find a way to change the location settings on an emulator, our analysis shows apps can bypass device settings to fetch a user's location—for instance, via a GPS provider or a network provider. Second, it is non-trivial to automate user behavior comprehensively for a wide array of apps, though this may be viable using some form of gesture fuzzing in limited scope. While we are limited to studying security and privacy geodifferences using static analysis approaches, truly examining the risks introduced to a user due to these geodifferences require an ability to conduct exhaustive runtime analysis of these apps and is a topic for further research.

Focused studies of categories of apps to uncover vulnerabilities. Studies similar to the one we conducted on payment apps that leveraged app differences to uncover vulnerabilities can help find vulnerabilities in other categories of apps as well. Our empirical investigation of apps on Google Play showed that apps belonging to *Dating* and *Lifestyle* categories have the most number of additional dangerous permissions requested, indicating that users of these apps in some countries are at a higher risk than others. Given how these apps contain sensitive data about users, attackers can leverage these apps to profile users in great detail. Focused studies of apps in these categories could explore why such differences exist, whether or not these differences can be exploited in the same way we exploited payment apps to leak sensitive data from critical workflows (e.g., authentication workflows), and its impact on users.

Attributing developer motivations and trends. A significant challenge we encountered during our study is establishing developer motivations for the geodifferences we observed in a data-driven manner. This needed additional data that we could not curate from Google Play. Our crawls showed incomplete or missing developer information for several apps we crawled. However, a manual inspection of a few random apps on Apple's App Store showed complete developer information. As a result, we could not correlate the trends we observed with the country of origin of these app developers. Future research could conduct such a study that focuses on developer trends based on the developer's geolocation.

To attribute reasons for blocking or the geodifferences we observe, one could conduct focused studies comparing globally popular apps with their locally popular (or region-specific) counterparts. For instance, we found instances of payment apps like Paypal unavailable only in Hungary among the EU countries we study. Whether or not the geoblocking is for consumer market segmentation or a developer action for other reasons will require further investigations and comparative studies of an app's context within a region while factoring in consumer trends and region-specific regulations.

Threat modeling with region-specific differences. Studies have shown how globally popular consumer devices like Samsung, Huawei, and Xiaomi phones in countries like India, China, and Russia have vulnerabilities that allow device compromise due to region-specific settings [233]. Similarly, countries like Brazil, Germany, India, and Indonesia are amongst the top few countries with the most number of potentially harmful applications. Brazil is also an outlier when it comes to their policy on content-ratings [111]. These studies show that mobile vulnerabilites vary by region, and these region-specific differences can potentially make exploiting apps easier in some countries than in others. We leveraged region-specific vulnerabilities such as access to devices running older (and less secure) Android versions and PHAs preinstalled to carry out automated attacks. However, broader studies could factor in different threat models, e.g., low-cost devices with modified kernels, or vulnerabilities resulting from a cloned SIM card. Some app markets also maintain a history of older releases of apps (which we also leveraged for our study). Future studies could conduct data-driven investigations of the impact of these region-specific differences in the mobile ecosystem.

Longitudinal geographic studies of app updates. Our research showed that updates of highly-rated apps, even when it is a minor update as indicated by the change the in the app's version number, are sometimes accompanied by drastic changes to an app's functionality. For instance, we found a difference of over 14 permissions added across app updates for certain apps, even within a 24-hour period. It is unclear whether such invasive updates are justified and whether the user is aware of the extent of these changes happening to apps installed on their phones. Our research also shows how developers release different app versions to different countries, raising the possibility that some countries may get app updates later. Future research could explore how long a developer takes to update other countries and whether or not the GDP and Internet Freedom Scores of these countries correlate with the frequency of updates. Researchers could also conduct longitudinal investigations that study security implications on users in countries that receive slower updates.

Evaluate risks of device targeting. Google Play's developer console provides fine-grained options for developers to country-target and device-target users. While our research shows how developers, given a choice, could country-target their apps by releasing different app versions to different countries, future studies could explore the impact of device-targeting on users. We believe that device targeting capabilities provided by Google have the potential to target sections of society based on their economic standing; e.g., users of expensive mobile phones can be seen as high-income users and potentially more valuable from an attack standpoint. Future work could explore different facets of device targeting, such as benefits in terms of app performance, categories of apps that are the most targeted, permissions differences of the same app across devices, and plausible reasons for any differences. We believe that the impact of such a work will not just improve a user's security but will also help mitigate the fragmentation because of different app versions on different devices.

Threat from social engineering. Our study on UPI has shown how the defenses of large mobile systems designed with security and privacy in mind topple in the face of social engineering attacks. Though our study shows how attackers can automate the exploits we found to scale, attackers in India have found easier ways to steal money from users. They resort to social engineering tactics to launch attacks. The cheap labor there makes scaling social engineering attacks easy. Our experience with app disclosures reveals that a user compromised by a PHA either pre-installed by a device vendor or through a social engineering attack is outside the threat model of platform developers. Despite users being the weakest link, platform owners consider social engineering attacks outside their threat model. Future research could evaluate threats arising from social engineering attacks on mobile users and find defenses to safeguard users in the face of such attacks.

6.2 Conclusion

With the recent digitization of economies worldwide, access to most services requires a mobile smartphone and the Internet at the very least. For some users, a smartphone is the only means to access the Internet. Mobile apps are central to how billions worldwide access Internet content and services such as banking, education, and healthcare through a mobile phone. Much of the growth in the mobile ecosystem is fostered by large platforms that provide a common backend infrastructure for developers to integrate their apps and release them for public consumption. These competing apps can be used by users worldwide to access Internet content and services. However, vulnerabilities in these platforms can affect entire ecosystems.

In this thesis, we made several contributions to understanding the state of security in the mobile ecosystem by conducting large-scale security analysis of two of the world's largest mobile ecosystems—the Unified Payments Interface used for free and instant mobile payments and Google Play used for app distribution. We conducted security analysis and measurements of these ecosystems through principled techniques from the vantage point of highly-rated mobile apps downloaded from Google Play, Android's official app market. We showed how these complex, black-box systems could be analyzed at scale even within the confines of a severely fragmented ecosystem, despite having no sophisticated tools or access to their backend infrastructure. We identified region-specific variables in these ecosystems that can be used to automate attacks, such as differences in mobile devices and threat vectors, and showed how attackers could leverage them to launch attacks. We also showed how measurements can be carried out despite regional differences in access to these backend infrastructures. We reverse-engineered these security-hardened ecosystems across nation-state boundaries from the point of view of an attacker (or user) having access to multiple vantage points, specifically, multiple versions of highly-rated apps integrated with these platforms.

In the second chapter, we discussed the interplay of the different stakeholders of the mobile ecosystem and described the various attack vectors available to an adversary. We give an overview of how as an attacker we reverse-engineered mobile apps and the multiple barriers in the form of security hardening techniques that we had to overcome. We described the implementation of a static analyzer tool that we developed to help with our analysis. We also summarized prior large-scale studies.

In the third chapter, we presented our 2020 publication, ”Security Analysis of the Unified Payments Interface and Payment Apps in India,” which was the first study on the UPI ecosystem from within the confines of a region, India, when UPI was first launched. UPI later evolved to support transactions from different countries worldwide. We showed how we could meticulously reverse-engineer the client-server handshake of the multi-factor authentication protocol used by UPI to remotely launch large-scale attacks even without any knowledge of its user. The attacks we showed have devastating implications for both users and non-users of UPI. Our disclosures led to the Indian Government acknowledging and addressing the core vulnerabilities we found and releasing an upgraded 2.0 version of the payments infrastructure. We also obtained several CVEs for our vulnerability disclosures on payment apps. A reanalysis of the UPI 2.0 protocol showed that several underlying security flaws remained, suggesting a need for further vetting and security analysis of UPI 2.0 under different threat models. We discussed the lessons learned and potential mitigation strategies.

In the fourth chapter, we presented our 2022 publication, ”A Large-scale Investigation into Geodifference in Mobile Apps,” detailing our empirical investigation of thousands of highly-rated, essential apps on Google Play from vantage points in 26 countries. We showed the design and implementation of a semi-automatic testbed that makes this large-scale measurement study feasible while factoring in all the variables that can potentially impact measurements. We uncovered significant geoblocking of essential apps on Google Play that disproportionately isolated some countries and root caused the actor responsible for it. While we corroborated anecdotal instances of takedowns due to government requests, we found that blocking by developers was the biggest enabler of geoblocking. We showed how users in some countries are at a higher risk of attack because developers selectively release apps with weaker security settings or privacy disclosures. We collected the largest multi-country app dataset with 117,233 app binaries and 112,607 privacy policies for those apps, which we released to the research community to foster further research.

Based on our findings, we provided recommendations for app market proprietors to address the issues we found.

In the fifth chapter, we presented our experiences with disclosures of our work with these platform owners and the challenges driving our work to closure. By stepping back and looking at it from multiple perspectives, we were able to identify the policies of these platform owners that impacted the different stakeholders of the mobile ecosystem. Our experience revealed that many of these security and privacy issues are left behind knowingly and are considered a design of these systems. As a result, the vulnerabilities we present may take months, if not years, to resolve. Thus there is a significant burden on users to do the right thing to protect themselves, and even with that, users are at high-risk. Based on our research, we articulated how the policies of platform owners impacted the other stakeholders in the ecosystem and why they must re-think their approach to security. We provided several practical recommendations for platform owners and developers to address the issues we found.

In conclusion, this thesis comes at a time when nation-states are increasingly adopting UPI's model for democratizing mobile banking for tech inclusion and regulating global app market proprietors like Google Play to foster alternate app markets. Our principled techniques and meticulously designed measurement testbeds are a much-needed and valuable foundation for security researchers, practitioners, and the community as a whole wanting to build, analyze, or audit multi-geo mobile ecosystems. However, much work remains to secure the mobile ecosystem effectively. We provide several future research directions the community can take to find and resolve the issues persistent in the ecosystem proactively. Our recommendations will help the security community, governments, and businesses wanting to build a secure and open mobile ecosystem at scale.

CHAPTER 7

Appendix

7.1 Code Overview

This section provides an overview of the code released by our study " A Large-scale Investigation into Geodifferences in Mobile Apps" [149]. The code for this project can be found at `https://github.com/censoredplanet/geodiff-app`.

The code for this research consists of three parts:

1. `apk-downloader`: to download apps from Google Play.

   We forked and customized an existing python repository called PlaystoreDownloader [55], a tool for downloading Android applications directly from the Google Play Store. After an initial (one-time) configuration, applications can be downloaded by specifying their package name. This project was originally forked from PlaystoreDownloader v1.1.0, which was released under the MIT License.

   Before interacting with the Play Store the researcher has to provide valid credentials and an ANDROID ID associated to your account. The credentials file (default PlaystoreDownloader/credentials.json) has the following format:

   ```
   [
     {
       "USERNAME": "username@gmail.com",
   ```

```
    "PASSWORD": "password",
    "ANDROID_ID": "XXXXXXXXXXXXXXXX",
    "LANG_CODE": "en_US",
    "LANG": "us"
  }
]
```

The JSON file is setup with a Google email and password in the USERNAME and PASSWORD fields that we want to use for crawls. This information is needed to authenticate with Google's servers. The above credentials can be used either on an Android device or an emulator. For our research, we used real Android devices. You can obtain the ANDROID ID by installing the Device ID application on your device, then copy the string corresponding to Google Service Framework (GSF) (use this string instead of the Android Device ID presented by the application). In case of errors related to authentication after the above steps, consider allowing less secure apps to access your account by visiting https://myaccount.google.com/lesssecureapps (visit the link while you are logged in). The input file should contain one application ID per line. The program creates the output directory OUTPUT_DIRECTORY_ROOT/COUNTRY/CURRENT_DATE containing output files info.log, finished.txt, failure.txt, and transient.txt. CURRENT_DATE is formatted YYYY-MM-DD. Sub-directories for the APK files are also created within the output directory on successful downloads.

2. `gpcrawler`: download app metadata from Google Play

   This program provides functions to scrape metadata from Google Play. It is forked from google-play-scraper v0.0.6, which was released under the MIT License [142]. This scraper provides the ability to get the Google Play store location from the home page, scrape the details for an app, scrape similar app list for a given app (which is useful to collect apps belonging to an app category), collect developer information, or perform a search on Google Play

3. `privacy crawler`: download privacy policies for Google Play apps

   This is a chromium based crawler that downloads an app's privacy policy given an input app, and the URL of its privacy policy link. The scraper emulates a headless browser.

7.2 Examples of Unavailable Apps

Table 7.1 shows the most developer blocked categories and countries excluding Iran (since Iran is the most developer-blocked in all countries and app categories) . The categories SHOP, FOOD, LIFE, ENT, and DATE are the most developer-blocked.The African countries— KE, ZW, TU and EG— are in seven out of the top ten developer-blocked categories. DATE has the most variance with 3.75% blocking in the US, and 30% in EG and ZW. Our results are consistent with prior research on Internet geoblocking, which notes SHOP, BUS, NEW and ENT in the top 10 geoblocked categories [164]. JP has the most developer-blocking in VID and EDU, and RU in COMM. HU has the most blocking of NEWS apps.

	Category	Country	%		Category	Country.	%
1	SHOP	TU, ZW	59.7	12	TRVL	RU	12.3
2	FOOD	KE, ZW, TU	48	13	PROD	UA	11.8
3	LIFE	KE, ZW	37.8	14	SOC	KE	10.4
4	ENT	EG, TU	36.3	15	HLTH	KR, RU	9.8
5	DATE	EG, ZW	30.1	16	BOOK	EG, HU, IL, TR, TU, ZW	6.8
6	FIN	ZW	27.6	17	PHOT	UA	6.25
7	MAP	VE	21	18	TOOL	RU, TU	5.9
8	NEWS	HU	15.2	19	GAME	HK	4.4
9	COMM	RU	14.8	20	VID	JP, HK, TU	4.4
10	BUS	KE, TU	12.7	21	EDU	JP	3.2
11	MUS	HK	12.5	22	MED	HU, IL	1.5

Table 7.1: **Percentage developer-blocking in categories and countries** We show developer blocking in categories and countries excluding Iran, ordered by the most blocked categories. African countries — Egypt, Kenya, Tunisia, and Zimbabwe — are the most developer blocked in seven out of the top 10 categories. The US has the lowest blocking overall, and EDU and MED are the least developer-blocked categories.

Table 7.2 shows select examples of globally popular apps that are developer blocked, and Table 7.3 shows examples of apps that are taken down per government request.

App ID	App Category	Countries
com.amazon.kindle*	BOOKS_AND_REFERENCE	Iran
us.zoom.videomeetings	BUSINESS	Iran
com.Slack	BUSINESS	Iran
com.ubercab.driver*	BUSINESS	Singapore, Iran
com.indeed.android.jobsearch	BUSINESS	Kenya, Iran, Zimbabwe, Tunisia
com.opera.touch	COMMUNICATION	India
com.facebook.orca*	COMMUNICATION	Iran
com.google.android.apps.hangoutsdialer	COMMUNICATION	Egypt, Iran, Kenya Russia, South Korea
com.facebook.talk	COMMUNICATION	Germany, Hong Kong, Hungary, Ireland, Israel, Russia, Ukraine, UK,
ru.beboo	DATING	USA
com.microsoft.xcloud	ENTERTAINMENT	Bahrain, Egypt, Kenya, Iran, South Korea, Tunisia, Ukraine, Venezuela Zimbabwe
com.etoro.wallet	FINANCE	Candada, Japan, Iran, Turkey, Zimbabwe
com.coinhako	FINANCE	Russia, Ukraine
com.rubygames.assassin	GAMES	Turkey
com.tencent.ig	GAME	Japan, South Korea, Iran
com.tinder*	LIFESTYLE	Iran
com.drugscom.app*	MEDICAL	Germany, Hungary, Ireland, UK, Iran
com.soundcloud.android*	MUSIC_AND_AUDIO	Hong Kong, Iran
com.amazon.mp3	MUSIC_AND_AUDIO	Bahrain, Egypt, Hong Kong, Iran, Israel, Kenya, Russia, Singapore, South Korea, Turkey, UAE, Ukraine, Venezuela, Zimbabwe
mnn.Android*	NEWS_AND_MAGAZINE	Germany
com.surfshark.free.proxy.trust.dns	PRODUCTIVITY	India
com.microsoft.mobile.polymer*	PRODUCTIVITY	Iran, Russia
com.etsy.android	SHOPPING	Iran
com.ebates	SHOPPING	Germany, Hungary, Ireland, UK
com.blued.international	SOCIAL	Germany, Hungary, Ireland, UK
com.expressvpn.vpn	TOOLS	Egypt, India, Kenya, Zimbabwe, Iran
com.sixflags.android	TRAVEL_AND_LOCAL	Bahrain, Iran Russia, Tunisia, Zimbabwe

Table 7.2: **Developer-blocked apps.** The table contains examples of globally popular apps that are developer-blocked. The table consists of the app's ID, the category to which the app belongs, and the countries where the app is blocked by developer. The * next to the app name indicates that the app is also unavailable in Tunisia because of SSL Errors. Though Zoom and Slack are unavailable in Iran on Google Play, the apps are available via a local app market in Iran. The table is ordered by category

App ID	App Category	Countries
com.megalol.muttertag	BOOKS_AND_REFERENCE	South Korea
com.brightai.lottery	BOOKS_AND_REFERENCE	South Korea
com.aristoz.espanol.curriculumvitae	BUSINESS	USA
com.linkedin.android	BUSINESS	Russia
com.truecaller	COMMUNICATION	Turkey
com.callapp.contacts	COMMUNICATION	Turkey
com.vmkoom.wifimap.connection.hotspot.wifi analyzer.password.anywhere	COMMUNICATION	USA
com.telos.app.im	COMMUNICATION	UK
ru.photostrana.mobile	DATING	Iran, South Korea, Tunisia, UAE
ru.mobstudio.andgalaxy	DATING	Iran, South Korea, Tunisia, UAE
com.beansprites.rainbowslushytruckFREE	EDUCATION	Australia, Canada, Colombia, Germany, Hungary, India, Ireland, Israel, Japan, Mexico, Russia, Singapore, South Korea, Turkey, Tunisia, UK, Ukraine, Venezuela
com.avuscapital.trading212	FINANCE	Australia
com.andreys.blockchainwallet	FINANCE	Germany, Hungary, Ireland, and UK
com.veepoo.hband	HEALTH_AND_FITNESS	Germany, Hungary, Ireland, Tunisia, UK, Venezuela
com.popularapp.periodcalendar	HEALTH_AND_FITNESS	Japan
com.streetview.liveearth.routefinder.guide	MAPS_AND_NAVIGATION	USA
com.lkd.calculator	PHOTOGRAPHY	India
com.app.calculator.vault.hider	TOOLS	India
com.av1rus.adblockremover	TOOLS	Russia
com.internet.speedtest.check.wifi.meter	TOOLS	Germany
com.mi.globalbrowser.mini	TOOLS	USA
ca.walmart.ecommerceapp	SHOPPING	USA

Table 7.3: **Government-requested takedowns** The table contains examples of globally popular apps taken down in their respective countries. South Korea has the highest number of government-directed takedowns, primarily due to its content regulations on apps with gambling, and adult content. Turkey has a high number of takedown of communication apps. The table is ordered by category

BIBLIOGRAPHY

[1] Manal Adham, Amir Azodi, Yvo Desmedt, and Ioannis Karaolis. How to attack two-factor authentication internet banking. In *Financial Cryptography and Data Security*, pages 322–328, 2013.

[2] Sadia Afroz, Michael Carl Tschantz, Shaarif Sajid, Shoaib Asif Qazi, Mobin Javed, and Vern Paxson. Exploring server-side blocking of regions. https://arxiv.org/pdf/1805.11606.pdf, 2018.

[3] Herman Aguinis and Sola O Lawal. elancing: A review and research agenda for bridging the science–practice gap. *Human Resource Management Review*, 23(1):6–17, 2013.

[4] Arian Akhavan Niaki, Shinyoung Cho, Zachary Weinberg, Nguyen Phong Hoang, Abbas Razaghpanah, Nicolas Christin, and Phillipa Gill. ICLab: A Global, Longitudinal Internet Censorship Measurement Platform. In *IEEE S&P*, 2020.

[5] Fatemah Alharbi, Michalis Faloutsos, and Nael Abu-Ghazaleh. Opening digital borders cautiously yet decisively: Digital filtering in Saudi Arabia. In *USENIX FOCI*, 2020.

[6] Mohammed Aamir Ali and Aad van Moorsel. Designed to be broken: A reverse engineering study of the 3D Secure 2.0 Payment Protocol. In *Financial Cryptography and Data Security*, pages 201–221, 2019.

[7] Suzan Ali, Mounir Elgharabawy, Quentin Duchaussoy, Mohammad Mannan, and Amr Youssef. Betrayed by the guardian: Security and privacy risks of parental control solutions. In *ACSAC*, 2020.

[8] Samaher AlJudaibi. Research paper for mobile devices security. 2016. https://www.researchgate.net/publication/309675787_Research_Paper_for_Mobile_Devices_Security.

[9] Ryan Amos, Gunes Acar, Elena Lucherini, Mihir Kshirsagar, Arvind Narayanan, and Jonathan Mayer. Privacy Policies over Time: Curation and Analysis of a Million-Document Dataset. https://arxiv.org/pdf/2008.09159.pdf, 2020.

[10] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. PolicyLint: Investigating internal privacy policy contradictions on Google Play. In *USENIX Security*, 2019.

[11] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. Actions speak louder than words: Entity-sensitive privacy policy and data flow analysis with PoliCheck. In *USENIX Security*, 2020.

[12] Android. Manifest.permission. https://developer.android.com/reference/android/Manifest.permission, 2021.

[13] New safety section in Google Play will give transparency into how apps use data. https://android-developers.googleblog.com/2021/05/new-safety-section-in-google-play-will.html, 2021.

[14] Android Open Source Project. Filters on Google Play. https://developer.android.com/google/play/filters, 2019.

[15] Android Open Source Project. App Manifest Overview. https://developer.android.com/guide/topics/manifest/manifest-intro, 2020.

[16] Android Open Source Project. Application signing. https://source.android.com/security/apksigning, 2020.

[17] Android Open Source Project. Command line tools. https://developer.android.com/studio/command-line, 2020.

[18] Android Open Source Project. Google Play. https://developer.android.com/distribute, 2020.

[19] Android Open Source Project. Network security configuration. https://developer.android.com/training/articles/security-config, 2020.

[20] Android Open Source Project. Sell digital purchases with Play In-app Billing. https://developer.android.com/distribute/best-practices/earn/in-app-purchases, 2020.

[21] Android Open Source Project. Version your app. https://developer.android.com/studio/publish/versioning, 2020.

[22] Android Open Source Project. android:usescleartexttraffic. https://developer.android.com/guide/topics/manifest/application-element, 2021.

[23] Google removes Fortnite from the Play Store for violating in-app payment policy. https://9to5google.com/2020/08/13/google-play-removes-fortnite/, 2020.

[24] Apktool. https://ibotpeaches.github.io/Apktool/.

[25] Android ad network statistics and market share. https://www.appbrain.com/stats/libraries/ad-networks.

[26] Appinventiv. Users are projected to spend $60 billion on google play by 2023. https://appinventiv.com/blog/google-play-store-statistics/. [Online; accessed November-2022].

[27] Apple Censorship. Apple Censorship. https://applecensorship.com/?l=en, 2020.

[28] ARTICLE 19. The Global Expression Report 2019 /2020. https://www.article19.org/wp-content/uploads/2020/10/GxR2019-20report.pdf.

[29] Simurgh Aryan, Homa Aryan, and J Alex Halderman. Internet censorship in Iran: A first look. In *USENIX FOCI*, 2013.

[30] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014.

[31] Elias Athanasopoulos, Sotiris Ioannidis, and Andreas Sfakianakis. CensMon: A web censorship monitor. In *USENIX FOCI*, 2011.

[32] Austrlian House Standing Committee on Infrastructure and Communications. At what cost? IT pricing and the Australia tax, July 2013. https://www.aph.gov.au/Parliamentary_Business/Committees/House_of_Representatives_Committees?url=ic/itpricing/report.htm.

[33] Michael Backes, Sven Bugiel, and Erik Derr. Reliable third-party library detection in Android and its security applications. In *ACM CCS*, 2016.

[34] John U Bacon. UPI transactions surge 86% to $844 bn, 2014.

[35] bankmycell. How many smartphones are in the world. https://www.bankmycell.com/blog/how-many-phones-are-in-the-world, 2022.

[36] Pratima Bansal, Stephanie Bertels, Tom Ewart, Peter MacConnachie, and James O'Brien. Bridging the research–practice gap, 2012.

[37] BHIM. https://play.google.com/store/apps/details?id=in.org.npci.upiapp, 2016. [Online; accessed October-2018].

[38] Android Developers Blog. Turning it up to 11: Android 11 for developers. https://android-developers.googleblog.com/2020/09/android11-final-release.html, 2020.

[39] S. Bojjagani and V. N. Sastry. VAPTAi: a threat model for vulnerability assessment and penetration testing of Android and iOS mobile banking apps. In *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*, pages 77–86, 10 2017.

[40] Sriramulu Bojjagani and V. N. Sastry. STAMBA: security testing for Android mobile banking apps. In *Advances in Signal Processing and Intelligent Recognition Systems*, pages 671–683, 2016.

[41] Mike Bond, Omar Choudary, Steven J. Murdoch, Sergei Skorobogatov, and Ross Anderson. Chip and Skim: Cloning EMV cards with the pre-play attack. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 49–64. IEEE Computer Society, 2014.

[42] India Briefing. India's UPI Interface to Become Accessible to More Users. https://www.india-briefing.com/news/global-acceptance-of-indias-digital-payment-systems-europe-latest-to-join-26183.html/#:~:text=Indian%20digital%20payment%20systems%20are,UPI%20interface%20for%20digital%20payments., 2023.

[43] BrowserStack. Testing for fragmentation: Understanding browser, os and device fragmentation. https://www.browserstack.com/blog/understanding-browser-os-and-device-fragmentation/, 2022.

[44] Business Insider. Judicial order seeks to block iranians access to google play app store. https://iranhumanrights.org/2019/10/judicial-order-seeks-to-block-iranians-access-to-google-play-app-store/, 2020.

[45] App stores list. https://www.businessofapps.com/guide/app-stores-list/#1, 2021.

[46] Doug Carnine. Bridging the research-to-practice gap. *Exceptional children*, 63(4):513–521, 1997.

[47] Sam Castle, Fahad Pervaiz, Galen Weld, Franziska Roesner, and Richard Anderson. Let's talk money: Evaluating the security challenges of mobile money in the developing world. In *Proceedings of the 7th Annual Symposium on Computing for Development*, ACM DEV '16, 2016.

[48] Pew Research Center. Digital divide persists even as americans with lower incomes make gains in tech adoption. https://www.pewresearch.org/fact-tank/2021/06/22/digital-divide-persists-even-as-americans-with-lower-incomes-make-gains-in-tech-adoption/. [Online; accessed November-2022].

[49] Liang Chen, Tony W Tong, Shaoqin Tang, and Nianchen Han. Governance and design of digital platforms: A review and future research directions on a meta-organization. *Journal of Management*, 48(1):147–184, 2022.

[50] Ping Chen, Lieven Desmet, Christophe Huygens, and Wouter Joosen. Longitudinal study of the use of client-side security mechanisms on the european web. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 457–462, 2016.

[51] Kelvin Chikomo, Ming Ki Chong, Alapan Arnab, and Andrew Hutchison. Security of mobile banking. 01 2006.

[52] Tom Chothia, Flavio D. Garcia, Chris Heppel, and Chris McMahon Stone. Why banker Bob (still) can't get TLS right: A security analysis of TLS in leading UK banking apps. pages 579–597, 01 2017.

[53] Primož Cigoj, Živa Stepančič, and Borka Jerman Blažič. A large-scale security analysis of web vulnerability: Findings, challenges and remedies. In *International Conference on Computational Science and Its Applications*, pages 763–771. Springer, 2020.

[54] Citizen Lab. We Chat, They Watch. https://citizenlab.ca/2020/05/we-chat-they-watch/, 2022.

[55] ClaudiuGeorgiu. PlaystoreDownloader. https://github.com/ClaudiuGeorgiu/PlaystoreDownloader, 2022.

[56] CNBC. Google vows to cooperate with India's antitrust authority after Android ruling. https://www.cnbc.com/2023/01/20/google-cooperates-with-india-antitrust-authority-after-android-ruling.html, 2023.

[57] CNET. Google to require privacy policy for Play store apps in April 2022. https://www.cnet.com/tech/services-and-software/google-to-require-privacy-policy-for-play-stor-apps-in-april-2022/, 2022.

[58] Coin Telegraph. Google lifts ban on popular ethereum wallet metamask. https://cointelegraph.com/news/google-play-store-takes-down-crypto-news-apps-including-cointelegraphs, 2020.

[59] Express Computer, 2019. https://www.expresscomputer.in/news/financial-cybercrime-and-identity-theft-in-india-are-increasing-fis/35099/.

[60] Stat Counter. https://www.statista.com/statistics/262157/market-share-held-by-mobile-operating-systems-in-india/, 2018. [Online; accessed October-2018].

[61] CuckooDroid. https://github.com/idanr1986/cuckoo-droid), 2018. [Online; accessed October-2018].

[62] Rowena Cullen. Addressing the digital divide. *Online information review*, 2001.

[63] Daily Hodl. Google lifts ban on popular Ethereum wallet MetaMask. https://dailyhodl.com/2020/01/02/google-lifts-ban-on-popular-ethereum-wallet-metamask/, 2020.

[64] Datareportal. Digital around the world. https://datareportal.com/global-digital-overview, 2022.

[65] J. Dean, D. Grove, and C. Chambers. Optimization of object-oriented programs using static class hierarchy analysis. In *Proceedings of the 9th European Conference on Object-Oriented Programming*, 1995.

[66] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, , and Thorsten Holz. We Value Your Privacy ... Now Take Some Cookies: Measuring the GDPR's Impact on Web Privacy. In *NDSS*, 2019.

[67] Louis F DeKoven, Audrey Randall, Ariana Mirian, Gautam Akiwate, Ansel Blume, Lawrence K Saul, Aaron Schulman, Geoffrey M Voelker, and Stefan Savage. Measuring security practices and how they impact security. In *Proceedings of the Internet Measurement Conference*, pages 36–49, 2019.

[68] Baden Delamore and Ryan KL Ko. A global, empirical analysis of the shellshock vulnerability in web applications. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 1129–1135. IEEE, 2015.

[69] Nurullah Demir, Tobias Urban, Kevin Wittek, and Norbert Pohlmann. Our (in) secure web: understanding update behavior of websites and its impact on security. In *International Conference on Passive and Active Network Measurement*, pages 76–92. Springer, 2021.

[70] Android Developers. https://developer.android.com/guide/topics/permissions/overview#normal_permissions, 2019. [Online; accessed August-2019].

[71] XDA Developers. https://www.xda-developers.com/stop-apps-reading-android-clipboard/, 2017. [Online; accessed October-2018].

[72] Gurpreet Dhillon, Kane Smith, and Indika Dissanayaka. Information systems security research agenda: Exploring the gap between research and practice. *The Journal of Strategic Information Systems*, 30(4):101693, 2021.

[73] DLA Piper. Data Protection Laws of the World. https://www.dlapiperdataprotection.com/system/modules/za.co.heliosdesign.dla.lotw.data_protection/functions/handbook.pdf?country=all, 2020.

[74] Alexandra Dmitrienko, Christopher Liebchen, Christian Rossow, and Ahmad-Reza Sadeghi. On the (in)security of mobile two-factor authentication. In *Financial Cryptography and Data Security*, pages 365–383, 2014.

[75] DMR. Google play statistics, user count and revenue (2022). https://expandedramblings.com/index.php/google-play-statistics-facts/. [Online; accessed November-2022].

[76] Best alternative Android markets: review by country. https://thinkmobiles.com/blog/alternative-app-stores-for-android/, 2021.

[77] Joshua J. Drake, Zach Lanier, Collin Mulliner, Pau Oliva Fora, Stephen A. Ridley, and Georg Wicherski. *Android Hacker's Handbook*. 1st edition, 2014.

[78] What are the biggest tracker networks and what can I do about them? https://spreadprivacy.com/biggest-tracker-networks/, 2021.

[79] Wolfgang Eberle. The system design process. *Wireless Transceiver Systems Design*, pages 17–43, 2008.

[80] Engadget. Samsung and Google censor LGBT apps in South Korea. https://www.engadget.com/2015-07-10-samsung-google-censor-lgbt-apps-south-korea.html, 2020.

[81] European Commission. A Digital Single Market for the benefit of all Europeans. https://ec.europa.eu/digital-single-market/en/policies/shaping-digital-single-market, 2019.

[82] Exodus Privacy. https://exodus-privacy.eu.org, 2021.

[83] Financial Express. https://www.financialexpress.com/money/beware-upi-app-user-loses-rs-6-8-lakh-from-his-sbi-account-was-it-fraud-why-it-happened/1426603/, 2018. [Online; accessed August-2019].

[84] Earlence Fernandes, Qi Alfred Chen, Justin Paupore, Georg Essl, J. Alex Halderman, Z. Morley Mao, and Atul Prakash. Android UI deception revisited: Attacks and defenses. In *Financial Cryptography and Data Security*, pages 41–59, 2017.

[85] Financial Tribune. Sanctions make Iran developers witty. https://financialtribune.com/articles/sci-tech/72057/sanctions-make-iran-developers-witty, 2017.

[86] Forbes. https://www.forbes.com/sites/zakdoffman/2019/08/10/google-warning-tens-of-millions-of-android-phones-come-preloaded-with-dangerous-malware/#5dcde47dddb3, 2019. [Online; accessed August-2019].

[87] Forbes. Google's censorship of cryptocurrencies goes way beyond youtube. https://www.forbes.com/sites/rogerhuang/2020/12/31/googles-censorship-of-cryptocurrencies-goes-way-beyond-youtube/#6387af9f32aa, 2020.

[88] Forbes. Watch as hackers hijack whatsapp accounts via critical telecoms flaws. https://www.forbes.com/sites/thomasbrewster/2016/06/01/whatsapp-telegram-ss7-hacks/?sh=32c97970178b, 2022.

[89] Fortune. Russia Demands LinkedIn App Takedown, Apple and Google Comply. https://fortune.com/2017/01/08/russia-linkedin-google-apple/, 2018.

[90] Sensors Tech Forum. https://sensorstechforum.com/android-ios-invasive-app-permissions-2018/), 2018. [Online; accessed October-2018].

[91] Mobile Security Framework. https://github.com/MobSF/Mobile-Security-Framework-MobSF, 2015. [Online; accessed October-2018].

[92] Freedom House. Freedom on the Net 2019: Iran. https://freedomhouse.org/country/iran/freedom-net/2019, 2019.

[93] Freedom House. Freedom on the Net Report 2019. https://freedomhouse.org/report/freedom-net/2019/crisis-social-media, 2019.

[94] Freedom on the Net Research Methodology. https://freedomhouse.org/reports/freedom-net/freedom-net-research-methodology, 2020.

[95] Christian Fuchs and Eva Horak. Africa and the digital divide. *Telematics and informatics*, 25(2):99–116, 2008.

[96] Agustin Garcia Asuero, Ana Sayago, and Gustavo González. The correlation coefficient: An overview. *Critical Reviews in Analytical Chemistry*, 2006.

[97] Annabelle Gawer. Digital platforms' boundaries: The interplay of firm scope, platform sides, and digital interfaces. *Long Range Planning*, 54(5):102045, 2021.

[98] John Geddes, Max Schuchard, and Nicholas Hopper. Cover Your ACKs: Pitfalls of Covert Channel Censorship Circumvention. In *ACM CCS*, 2013.

[99] Google. App components.

[100] Google. Distribute app releases to specific countries. https://support.google.com/googleplay/android-developer/answer/7550024?hl=en, 2020.

[101] Google. Google Transparency Report. https://transparencyreport.google.com/government-removals/overview?hl=en, 2020.

[102] Google. Prepare your app for review. https://support.google.com/googleplay/android-developer/answer/9815348, 2020.

[103] Google. Requirements for distributing apps in specific countries/regions. https://support.google.com/googleplay/android-developer/answer/6223646?hl=en, 2020.

[104] Google. Select a category and tags for your app or game. https://support.google.com/googleplay/android-developer/answer/113475, 2020.

[105] Google. Supported locations for distribution to Google Play users. https://support.google.com/googleplay/android-developer/table/3541286, 2020.

[106] Google. Try new Android apps before they're officially released. https://support.google.com/googleplay/answer/7003180?hl=en, 2020.

[107] Google. User data. https://support.google.com/googleplay/android-developer/answer/9888076, 2020.

[108] Google. View & restrict your app's compatible devices. https://support.google.com/googleplay/android-developer/answer/7353455?hl=en, 2021.

[109] Google. Google Play Developer Distribution Agreement. https://play.google.com/about/developer-distribution-agreement.html, 2022.

[110] Google. App security best practices. https://developer.android.com/topic/security/best-practices, 2023.

[111] Google. Apps & Games content ratings on Google Play. https://support.google.com/googleplay/answer/6209544?hl=en#zippy=%2Cbrazil, 2023.

[112] Google. Google please remove restrictions for Iranian developers. https://support.google.com/android/thread/197162207/google-please-remove-restrictions-for-iranian-developers?hl=en, 2023.

[113] Google Play Help. https://support.google.com/googleplay/thread/22123351?hl=en, 2019.

[114] Google Transparency Report. https://transparencyreport.google.com, 2021.

[115] Google-Play-Scraper. https://github.com/JoMingyu/google-play-scraper.

[116] D. Grove, G. DeFouw, J. Dean, and C. Chambers. Call graph construction in object-oriented languages. *ACM SIGPLAN Notices*, 1997.

[117] Catherine Han, Irwin Reyes, Alvaro Feal, Joel Reardon, Primal Wijesekera, Narseo Vallina-Rodriguez, Amit Elazari, Kenneth A. Bamberger, and Serge Egelman. The Price is (Not) Right: Comparing Privacy in Free and Paid Apps. In *PETs*, 2020.

[118] Eszter Hargittai. The digital divide and what to do about it. *New economy handbook*, 2003:821–839, 2003.

[119] Hamza Harkous, Kassem Fawaz, Rémi Lebret, Florian Schaub, Kang G. Shin, and Karl Aberer. Polisis: Automated Analysis and Presentation of Privacy Policies Using Deep Learning. In *USENIX Security*, 2018.

[120] Andrew Harris, Seymour Goodman, and Patrick Traynor. Privacy and security concerns associated with mobile money applications in Africa. *Washington Journal of Law, Technology and Arts*, 01 2013.

[121] Yongzhong He, Xuejun Yang, Binghui Hu, and Wei Wang. Dynamic privacy leakage analysis of Android third-party libraries. *Journal of Information Security and Applications*, 2019.

[122] Luis Hestres. App neutrality: Apple's app store and freedom of expression online. *Journal of Communication*, 2013.

[123] Ahstair Hewison and Stuart Wlldman. The theory-practice gap in nursing: A new dimension. *Journal of Advanced Nursing*, 24(4):754–761, 1996.

[124] HideMyAss! HideMyAss! is pulling out of Russia. https://blog.hidemyass.com/en/hidemyass-is-pulling-out-of-russia, 2019.

[125] Martin Hilbert. The bad news is that the digital access divide is here to stay: Domestically installed bandwidths among 172 countries for 1986–2014. *Telecommunications Policy*, 2016.

[126] The Hindu. https://www.thehindu.com/news/national/kerala/hackers-compromise-upi-apps/article25692100.ece, 2019. [Online; accessed August-2019].

[127] Hulu. Why can't I use Hulu internationally? https://help.hulu.com/s/article/cant-use-internationally, 2020.

[128] Internet Fragmentation. https://icannwiki.org/Internet_Fragmentation, 2016.

[129] Android Security 2017 Year in Review. https://source.android.com/security/reports/Google_Android_Security_2017_Report_Final.pdf, 2019. [Online; accessed August-2019].

[130] Android Security 2018 Year in Review. https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf/, 2019. [Online; accessed August-2019].

[131] Infinum. Securing Mobile Banking on Android with SSL Certificate Pinning. https://infinum.com/blog/securing-mobile-banking-on-android-with-ssl-certificate-pinning, 2023.

[132] Information Commissioner's Office. Guide to the GDPR. https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/, 2020.

[133] MWR Infosecurity. https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-drozer-user-guide-2015-03-23.pdf, 2015. [Online; accessed October-2018].

[134] Business Insider. https://www.businessinsider.com/upi-market-share-in-inda-open-for-taking-2019-8, 2019. [Online; accessed August-2019].

[135] Business Insider. The Iranian government released an official coronavirus app for Iranians, but Google pulled it from its app store. https://www.businessinsider.com/iran-coronavirus-app-pulled-by-google-2020-3, 2020.

[136] Infosec Institute. Hooking and patching Android apps using Xposed framework. https://resources.infosecinstitute.com/android-hacking-and-security-part-25-hooking-and-patching-android-apps-using-xposed-framework/#gref. [Online; accessed November-2019].

[137] Internet Society. Brief History of Internet. https://www.internetsociety.org/wp-content/uploads/2017/09/ISOC-History-of-the-Internet_1997.pdf, 2017.

[138] Investopedia. https://www.investopedia.com/news/india-demonetization-993-money-returned/, 2018. [Online; accessed October-2018].

[139] Slinger Jansen and Ewoud Bloemendal. Defining app stores: The role of curated marketplaces in software ecosystems. In *Software Business*. Springer, 2013.

[140] JEB. https://www.pnfsoftware.com/, 2018. [Online; accessed October-2018].

[141] Lianrui Jia and Lotus Ruan. Going global: Comparing chinese mobile applications' data and user privacy governance at home and abroad. *Internet policy review*, 9(3):1–22, 2020.

[142] JoMingyu. Google Play Scraper. https://github.com/JoMingyu/google-play-scraper, 2022.

[143] Ben Jones, Tzu-Wen Lee, Nick Feamster, and Phillipa Gill. Automated Detection and Fingerprinting of Censorship Block Pages. In *IMC*, 2014.

[144] Joe H. Ward Jr. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 1963.

[145] Jin-Hyuk Jung, Ju Young Kim, Hyeong-Chan Lee, and Jeong Hyun Yi. Repackaging attack on Android banking applications and its countermeasures. 73, 12 2013.

[146] keytool. https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html.

[147] Radhesh Krishnan Konoth, Victor van der Veen, and Herbert Bos. How anywhere computing just killed your phone-based two-factor authentication. In *Financial Cryptography and Data Security*, pages 405–421, 2017.

[148] Y. Kouraogo, K. Zkik, E. J. El Idrissi Noreddine, and G. Orhanou. Attacks on Android banking applications. In *2016 International Conference on Engineering MIS (ICEMIS)*, pages 1–6, 9 2016.

[149] Renuka Kumar, Apurva Virkud, Ram Sundara Raman, Atul Prakash, and Roya Ensafi. A large-scale investigation into geodifferences in mobile apps. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1203–1220, 2022.

[150] Benedikt Lebek, Jörg Uffen, Markus Neumann, Bernd Hohler, and Michael H. Breitner. Information security awareness and behavior: a theory-based literature review. *Management Research Review*, 37(12):1049–1092, 2014.

[151] Soo Ling Lim, Peter J Bentley, Natalie Kanakam, Fuyuki Ishikawa, and Shinichi Honiden. Investigating country differences in mobile app user behavior and challenges for software engineering. *IEEE Trans. on Software Engineering*, 2014.

[152] Fuqi Lin, Haoyu Wang, Liu Wang, and Xuanzhe Liu. A Longitudinal Study of Removed Apps in IOS App Store. In *WWW*, 2021.

[153] Thomas Linden, Rishabh Khandelwal, Hamza Harkous, and Kassem Fawaz. The Privacy Policy Landscape After the GDPR. In *PETs*, 2020.

[154] Beverly Lindsay and Maria T. Poindexter. The Internet: Creating Equity through Continuous Education or Perpetuating a Digital Divide? *Journal of Comparative Education Review*, 2003.

[155] The Hindu Business Line. https://www.thehindubusinessline.com/info-tech/senior-executives-vulnerable-to-social-engineering-attacks-verizon-2019-report/article27068079.ece, 2019. [Online; accessed August-2019].

[156] Xing Liu, Jiqiang Liu, Sencun Zhu, Wei Wang, and Xiangliang Zhang. Privacy Risk Analysis and Mitigation of Analytics Libraries in the Android Ecosystem. *IEEE Trans. on Mobile Computing*, 2020.

[157] Luminati. https://luminati.io.

[158] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. LibRadar: Fast and Accurate Detection of Third-Party Libraries in Android Apps. In *IEEE/ACM Software Engineering Companion*, 2016.

[159] Daithı Mac Sıthigh. App law within: rights and regulation in the smartphone age. *Journal of Law and Information Technology*, 2013.

[160] El Nour Madhoun, Bertin Emmanuel, and Guy Pujolle. The EMV payment system: Is it reliable? In *The 3rd IEEE Cyber Security in Networking International Conference (CSNet 2019)*, 2019.

[161] Alexander J Martin. Google faces calls to lift anti-censorship blocks in Iran. https://news.sky.com/story/google-faces-calls-to-lift-anti-censorship-blocks-in-iran-11192888, 2018.

[162] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. A Survey of App Store Analysis for Software Engineering. *IEEE Trans. on Software Engineering*, 2017.

[163] William Martin, Federica Sarro, and Mark Harman. Causal Impact Analysis for App Releases in Google Play. In *ACM SIGSOFT*, 2016.

[164] Allison McDonald, Matthew Bernhard, Luke Valenta, Benjamin VanderSloot, Will Scott, Nick Sullivan, J. Alex Halderman, and Roya Ensafi. 403 Forbidden: A Global View of CDN Geoblocking. In *IMC*, 2018.

[165] Top 10 Android App Analytics Platforms. https://medium.com/android-news/top-10-android-app-analytics-platforms-b3496c55b6f1, 2017.

[166] Microsoft. Skype Lite. https://www.skype.com/en/skype-lite/, 2020.

[167] MITMProxy. https://mitmproxy.org, 2019. [Online; accessed August-2019].

[168] MITRE. Application Developer Guidance. https://attack.mitre.org/mitigations/M1013/, 2023.

[169] MoneyLife. BHIM UPI: NPCI says it won't be responsible for loss or fraud, user fully takes the risk. https://www.moneylife.in/article/bhim-upi-npci-says-it-wont-be-responsible-for-loss-or-fraud-user-fully-takes-the-risk/50270.html, 2022.

[170] Collin Mulliner, Ravishankar Borgaonkar, Patrick Stewin, and Jean-Pierre Seifert. Sms-based one-time passwords: attacks and defense. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013.

[171] Collin Mulliner, Ravishankar Borgaonkar, Patrick Stewin, and Jean-Pierre Seifert. SMS-based One-Time Passwords: Attacks and defense. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 150–159, 2013.

[172] multipass. https://github.com/canonical/multipass.

[173] S. J. Murdoch, S. Drimer, R. Anderson, and M. Bond. Chip and PIN is broken. In *2010 IEEE Symposium on Security and Privacy*, pages 433–446, 2010.

[174] P. Mutchler, Y. Safaei, A. Doupé, and J. Mitchell. Target Fragmentation in Android Apps. In *IEEE S&P*, 2016.

[175] NASA. Systems engineering handbook. https://www.nasa.gov/seh/4-design-process, 2022.

[176] William Navidi. *Statistics for Engineers and Scientists*, chapter 4. McGraw-Hill Education, 2014.

[177] Palo Alto Networks. https://researchcenter.paloaltonetworks.com/2017/09/unit42-android-toast-overlay-attack-cloak-and-dagger-with-no-permissions, 2017. [Online; accessed October-2018].

[178] NIST Policy on Hash Functions. https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions, 2015.

[179] NIST Special Publication. Transitioning the Use of Cryptographic Algorithms and Key Lengths. https://csrc.nist.gov/CSRC/media/Publications/sp/800-131a/rev-2/draft/documents/sp800-131Ar2-draft.pdf, 2019.

[180] E. Novak and C. Marchini. Android App Update Timing: A Measurement Study. In *IEEE Mobile Data Management*, 2019.

[181] NPCI. NPCI live members. https://www.npci.org.in/upi-live-members. [Online; accessed August-2019].

[182] NPCI. NPCI third party apps. https://www.npci.org.in/upi-PSP%263rdpartyApps. [Online; accessed August-2019].

[183] Barak W. Nyamtiga and Loserian S. Laizer. Enhanced security model for mobile banking systems in Tanzania. *International Journal of Technology Enhancements and Emerging Engineering Research*, 01 2013.

[184] 10 Mobile Usage Statistics Every Marketer Should Know in 2020. https://www.oberlo.com/blog/mobile-usage-statistics, 2020.

[185] Business of Apps. App stores list 2018. https://www.businessofapps.com/guide/app-stores-list/. [Online; accessed August-2019].

[186] Business of Apps. App Revenue Data (2023). https://www.businessofapps.com/data/app-revenues/, 2023.

[187] National Payments Corporation of India. https://www.npci.org.in/sites/default/files/UPI-PG-RBI_Final.pdf, 2016. [Online; accessed October-2018].

[188] National Payments Corporation of India. https://www.npci.org.in/product-statistics/upi-product-statistics, 2019. [Online; accessed August-2019].

[189] Open Observatory of Network Interference. https://ooni.org.

[190] Open Observatory of Network Interference. OONI API. https://api.ooni.io.

[191] OWASP. https://sushi2k.gitbooks.io/the-owasp-mobile-security-testing-guide/content/, 2018. [Online; accessed October-2018].

[192] OWASP. https://github.com/OWASP/owasp-mstg/tree/master/Checklists, 2018. [Online; accessed October-2018].

[193] Saurabh Panjwani and Edward Cutrell. Usably secure, low-cost authentication for mobile banking. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, Proc. SOUPS, 2010.

[194] Sameer Paradkar. *Mastering non-functional requirements*. Packt Publishing Ltd, 2017.

[195] PayTM. Upi transactions crossed the mark of 730 cr in oct 2022, grew 73% y-o-y. https://business.paytm.com/blog/upi-transactions-in-oct-2022-ft/. [Online; accessed November-2022].

[196] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. Global measurement of DNS censorship. In *USENIX Security*, 2017.

[197] Ella Peltonen, Eemil Lagerspetz, Jonatan Hamberg, Abhinav Mehrotra, Mirco Musolesi, Petteri Nurmi, and Sasu Tarkoma. The hidden image of mobile apps: Geographic, demographic, and cultural factors in mobile usage. In *Human-Computer Interaction with Mobile Devices and Services*, 2018.

[198] PlaystoreDownloader. https://github.com/ClaudiuGeorgiu/PlaystoreDownloader.

[199] Android Police. Android 12 is getting serious about keeping clipboard snippets private. https://www.androidpolice.com/2021/06/09/android-12-will-tell-you-when-apps-read-from-your-clipboard/. [Online; accessed November-2022].

[200] R. Potharaju, M. Rahman, and B. Carbunar. A Longitudinal Study of Google Play. *IEEE Trans. on Computational Social Systems*, 2017.

[201] The Print. As digital divide widens, india risks losing a generation to pandemic disruption. https://theprint.in/india/education/as-digital-divide-widens-india-risks-losing-a-generation-to-pandemic-disruption/568394/. [Online; accessed November-2022].

[202] Privacy Hub. Google Play Store Removes Popular Apps with 20 Million+ Downloads for Malicious Activity. https://www.cyberghostvpn.com/en_US/privacyhub/play-store-clicker-malware/, 2022.

[203] Proxychains. https://github.com/rofl0r/proxychains-ng.

[204] Qualys. SSL Server Test. https://ssllabs.com/ssltest, 2020.

[205] Massimo Ragnedda, Maria Laura Ruiu, and Felice Addeo. Measuring Digital Capital: An empirical investigation. *New Media & Society*, 2020.

[206] Ram Raman, Adrian Stoll, Jakub Dalek, Reethika Ramesh, Will Scott, and Roya Ensafi. Measuring the Deployment of Network Censorship Filters at Global Scale. In *NDSS*, 2020.

[207] Reethika Ramesh, Ram Sundara Raman, Matthew Bernhard, Victor Ongkowijaya, Leonid Evdokimov, Anne Edmundson, Steven Sprecher, Muhammad Ikram, and Roya Ensafi. Decentralized Control: A Case Study of Russia. In *NDSS*, 2020.

[208] Readabilipy. https://github.com/alan-turing-institute/ReadabiliPy.

[209] Bradley Reaves, Nolen Scaife, Adam Bates, Patrick Traynor, and Kevin R.B. Butler. Mo(bile) money, mo(bile) problems: Analysis of branchless banking applications in the developing world. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 17–32, 2015.

[210] Bradley Reaves, Luis Vargas, Nolen Scaife, Dave Tian, Logan Blue, Patrick Traynor, and Kevin R. B. Butler. Characterizing the security of the SMS ecosystem with public gateways. *ACM Trans. Priv. Secur.*, 22:2:1–2:31, December 2018.

[211] Grand View Research. Mobile Application Market Size, Share, & Trends Analysis Report By Store Type (Google Store, Apple Store, Others), By Application, By Region, And Segment Forecasts, 2023. https://www.grandviewresearch.com/industry-analysis/mobile-application-market#:~:text=Store%20Type%20Insights,app%20purchases%20and%20premium%20apps., 2023.

[212] Leonard Richardson. Beautiful Soup. https://www.crummy.com/software/BeautifulSoup/, 2020.

[213] RISKIQ. 2020 Mobile App Threat Landscape Report. https://www.riskiq.com/wp-content/uploads/2021/01/RiskIQ-2020-Mobile-App-Threat-Landscape-Report.pdf, 2020.

[214] Everett M Rogers. The digital divide. *Convergence*, 7(4):96–111, 2001.

[215] M. Roland, J. Langer, and J. Scharinger. Applying relay attacks to Google wallet. In *2013 5th International Workshop on Near Field Communication (NFC)*, pages 1–6, 2 2013.

[216] Michael Roland and Josef Langer. Cloning credit cards: A combined pre-play and downgrade attack on EMV contactless. In *Proceedings of the 7th USENIX Conference on Offensive Technologies*, WOOT'13, pages 6–6. USENIX Association, 2013.

[217] M. Sabt, M. Achemlal, and A. Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64, 8 2015.

[218] Selenium. https://github.com/SeleniumHQ/selenium/.

[219] Yun Shen, Pierre-Antoine Vervier, and Gianluca Stringhini. Understanding Worldwide Private Information Collection on Android. In *NDSS*, 2021.

[220] Hossein Siadati, Toan Nguyen, Payas Gupta, Markus Jakobsson, and Nasir Memon. Mind your SMSes. *Comput. Secur.*, 65:14–28, March 2017.

[221] Soot. https://www.sable.mcgill.ca/soot/tutorial/profiler2/index.html, 2018. [Online; accessed October-2018].

[222] Sophos Naked Security. Analytics firm's VPN and ad-blocking apps are secretly grabbing user data. https://nakedsecurity.sophos.com/2020/03/12/analytics-firms-vpn-and-ad-blocking-apps-are-secretly-grabbing-user-data/, 2020.

[223] "The item you were attempting to purchase could not be found" Android in-app billing. https://stackoverflow.com/questions/23918190/the-item-you-were-attempting-to-purchase-could-not-be-found-android-in-app-bil.

[224] State of California Department of Justice. California Consumer Privacy Act (CCPA). https://www.oag.ca.gov/privacy/ccpa, 2020.

[225] Statista. Percentage of mobile device website traffic worldwide from 1st quarter 2015 to 2nd quarter 2022. https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/, 2022.

[226] Nils D. Steiner. Comparing Freedom House Democracy Scores to Alternative Indices and Testing for Political Bias: Are US Allies Rated as More Democratic by Freedom House? *Journal of Comparative Policy Analysis: Research and Practice*, 2016.

[227] P. H. Stocks, B. G.Ryder, W. A. Landi, and S. Zhang. Comparing flow and context sensitivity on the modification-side-effects problem. In *Proceedings of the 1998 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '98, 1998.

[228] Chris McMahon Stone, Tom Chothia, and Flavio D. Garcia. Spinner: Semi-automatic detection of pinning without hostname verification. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC 2017, pages 176–188, 2017.

[229] Avinash Sudhodanan, Roberto Carbone, Luca Compagna, Nicolas Dolgin, Alessandro Armando, and Umberto Morelli. Large-scale analysis & detection of authentication cross-site request forgeries. In *2017 IEEE European symposium on security and privacy (EuroS&P)*, pages 350–365. IEEE, 2017.

[230] Ruoxi Sun and Minhui Xue. Quality Assessment of Online Automated Privacy Policy Generators: An Empirical Study. In *Evaluation and Assessment in Software Engineering*, 2020.

[231] Ram Sundara Raman, Prerana Shenoy, Katharina Kohls, and Roya Ensafi. Censored Planet: An Internet-wide, Longitudinal Censorship Observatory. In *ACM CCS*, 2020.

[232] Ana Swanson, David McCabe, and Jack Nicas. Trump Administration to Ban TikTok and WeChat From U.S. App Stores. https://www.nytimes.com/2020/09/18/business/trump-tik-tok-wechat-ban.html, 2020.

[233] The Daily Swig. Android security: Regional differences make mobiles devices in some countries more hackable than others. https://portswigger.net/daily-swig/android-security-regional-differences-make-mobiles-devices-in-some-countries-more-hackable-than-others, 2023.

[234] Vincent F. Taylor and Ivan Martinovic. Short paper: A longitudinal study of financial apps in the Google Play store. In *Financial Cryptography and Data Security*, pages 302–309, 2017.

[235] Vincent F. Taylor and Ivan Martinovic. To Update or Not to Update: Insights From a Two-Year Study of Android App Evolution. In *Asia CCS*, 2017.

[236] HT Tech. Some Android apps are secretly taking screenshots of your activity. https://tech.hindustantimes.com/tech/news/popular-apps-secretly-take-screenshots-of-activity-send-them-to-third-parties-study-story-MPnd5BX8PUK1mR7UAYgyeO.html, 2022.

[237] TechCrunch. Google filing says EU's antitrust division is investigating Play Store practices. https://techcrunch.com/2022/10/27/google-play-eu-antitrust/, 2023.

[238] France Telecom. Asm, 1999.

[239] Telegram. Keep calm and send telegrams! https://telegram.org/blog/15million-reuters, 2022.

[240] Tencent. Tencent Appstore. https://appstore.tencent.com/, 2023.

[241] The Hindu. Government bans 59 apps including China-based TikTok, WeChat. https://www.thehindu.com/news/national/govt-bans-59-apps-including-tiktok-wechat/article31947445.ece, 2020.

[242] The Telegraph. Facebook faces legal hurdle to launch Messenger Kids app in UK. https://www.telegraph.co.uk/technology/2020/05/02/facebook-faces-legal-hurdle-launch-messenger-kids-app-uk/, 2020.

[243] Think Progress. Iranians outraged over removal of apps from Google Play store. https://archive.thinkprogress.org/google-play-iranian-apps-sanctions-2e7ba4b1649d/, 2017.

[244] Economic Times. https://economictimes.indiatimes.com/news/politics-and-nation/new-form-of-otp-theft-on-rise-many-techies-victims/articleshow/67521098.cms, 2019. [Online; accessed August-2019].

[245] F. Tip and J. Palsberg. Scalable propagation-based call graph construction algorithms. In *Proceedings of the 15th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*. ACM SIGPLAN Notices, 2000.

[246] India Today. https://www.indiatoday.in/technology/news/story/fraudsters-steal-rs-91-000-from-a-man-s-e-wallet-1382689-2018-11-05, 2018. [Online; accessed August-2019].

[247] TorGuard. Has Hong Kong started blocking VPN providers? https://torguard.net/blog/has-hong-kong-started-blocking-vpn-providers/, 2020.

[248] Patrick Traynor, Thomas La Porta, and Patrick McDaniel. *Security for telecommunications networks*. Advances in information security. 2008.

[249] Michael Carl Tschantz, Sadia Afroz, Shaarif Sajid, Shoaib Asif Qazi, Mobin Javed, and Vern Paxson. A Bestiary of Blocking: The Motivations and Modes behind Website Unavailability. In *USENIX FOCI*, 2018.

[250] Aggeliki Tsohou, Spyros Kokolakis, Maria Karyda, and Evangelos Kiountouzis. Investigating information security awareness: research and practice gaps. *Information Security Journal: A Global Perspective*, 17(5-6):207–227, 2008.

[251] Jan Van Dijk. *The digital divide*. John Wiley & Sons, 2020.

[252] Jan Van Dijk and Kenneth Hacker. The digital divide as a complex and dynamic phenomenon. *The information society*, 19(4):315–326, 2003.

[253] Jan AGM Van Dijk. Digital divide research, achievements and shortcomings. *Poetics*, 34(4-5):221–235, 2006.

[254] Tom Van Goethem, Ping Chen, Nick Nikiforakis, Lieven Desmet, and Wouter Joosen. Large-scale security analysis of the web: Challenges and findings. In *International Conference on Trust and Trustworthy Computing*, pages 110–126. Springer, 2014.

[255] Tom Van Goethem, Victor Le Pochat, and Wouter Joosen. Mobile friendly or attacker friendly? a large-scale security evaluation of mobile-first websites. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 206–213, 2019.

[256] Vasilis Ververis, Marios Isaakidis, Valentin Weber, and Benjamin Fabian. Shedding Light on Mobile App Store Censorship. In *User Modeling, Adaptation and Personalization*, 2019.

[257] Nicolas Viennot, Edward Garcia, and Jason Nieh. A measurement study of google play. In *The 2014 ACM international conference on Measurement and modeling of computer systems*, pages 221–233, 2014.

[258] M Vimalkumar, Jang Bahadur Singh, and Sujeet Kumar Sharma. Exploring the multi-level digital divide in mobile phone adoption: A comparison of developing nations. *Information Systems Frontiers*, 23(4):1057–1076, 2021.

[259] Wandera. What are app permissions – a look into Android app permissions. https://www.wandera.com/mobile-security/app-and-data-leaks/app-permissions/. [Online; accessed August-2019].

[260] H. Wang, H. Liu, X. Xiao, G. Meng, and Y. Guo. Characterizing Android App Signing Issues. In *IEEE/ACM Automated Software Engineering*, 2019.

[261] Haoyu Wang, Hao Li, and Yao Guo. Understanding the evolution of mobile app ecosystems: A longitudinal measurement study of Google Play. In *WWW*, 2019.

[262] Haoyu Wang, Hao Li, Li Li, Yao Guo, and Guoai Xu. Why are Android apps removed from Google Play? A large-scale empirical study. In *MSR*, 2018.

[263] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. Beyond Google Play: A large-scale comparative study of Chinese Android app markets. In *IMC*, 2018.

[264] Barney Warf. Geographies of global Internet censorship. *GeoJournal*, 2011.

[265] Jane K Winn and Louis De Koker. Introduction to mobile money in developing countries: Financial inclusion and financial integrity conference special issue. *University of Washington School of Law Research Paper*, (2013-01), 2013.

[266] Wenbo Yang, Yuanyuan Zhang, Juanru Li, Hui Liu, Qing Wang, Yueheng Zhang, and Dawu Gu. Show me the money! Finding flawed implementations of third-party in-app payment in Android apps. In *NDSS*, 2017.

[267] Changsok Yoo, Byung-Tak Kang, and Huy Kang Kim. Case study of the vulnerability of OTP implemented in internet banking systems of South Korea. *Multimedia Tools and Applications*, 74:3289–3303, 05 2015.

[268] K. Yoshida, H. Imai, N. Serizawa, T. Mori, and A. Kanaoka. Understanding the origins of weak cryptographic algorithms used for signing Android apps. In *IEEE Computer Software and Applications*, 2018.

[269] L. Yu, X. Luo, Jiachi Chen, H. Zhou, T. Zhang, H. Chang, and H. Leung. PPChecker: Towards Accessing the Trustworthiness of Android Apps' Privacy Policies. *IEEE Trans. on Software Engineering*, 2018.

[270] ZDNET. Google Play malware: If you've downloaded these malicious apps, delete them immediately. https://www.zdnet.com/article/google-play-malware-if-youve-downloaded-these-malicious-apps-delete-them-immediately/, 2022.

[271] Nan Zhong and Florian Michahelles. Google Play is not a long tail market: an empirical analysis of app adoption on the Google Play app market. In *ACM Applied Computing*, 2013.

[272] Jonathan Zittrain and Benjamin Edelman. Internet filtering in China. *IEEE Internet Computing*, 2003.

[273] Zoom. Restricted countries or regions. https://support.zoom.us/hc/en-us/articles/203806119-Restricted-countries-or-regions, 2020.