

## ABSTRACT

ANDOW, BENJAMIN ERIC. Privacy Risks of Sensitive User Data Exposure in Mobile Ecosystems. (Under the direction of Dr. William Enck).

Mobile applications frequently collect and share a wide-range of privacy-sensitive user data. Such data is an extremely valuable commodity for legitimate business purposes, but also for nefarious and illicit purposes. Therefore, identifying the privacy risks of exposing privacy-sensitive user data to applications has been a topic of great research interest over the past decade. However, prior works in this domain are plagued with significant limitations that result in incomplete or imprecise approximations of the privacy risks. These limitations are mainly due to an incomplete characterization of the types of data that applications request and the context-insensitivity of their privacy policy analysis techniques, or lack thereof.

In this dissertation, we characterize and identify privacy risks resulting from disclosing privacy-sensitive user data to applications, addressing much of the limitations of prior works. In particular, we analyze how privacy-sensitive user data is obtained and used, how privacy disclosures are discussed, and whether all uses of such data are disclosed. First, we characterize the space of privacy-sensitive user data sources to understand what types of data applications are requesting from users. We design and implement UiRef, an analysis framework to resolve the semantics of user input requests, and apply it to study privacy risks associated with user input requests in 50,162 Android applications from Google Play. Our analysis uncovers several concerning developer practices, including insecure exposure of account passwords and privacy violations due to non-consensual disclosures of privacy-sensitive user input to third parties. Second, we characterize the semantics of sharing and collection statements within privacy policies to understand how privacy practices are being disclosed. We demonstrate the importance of holistic analysis of privacy policies through our identification and formalization of self-contradictory policy statements. We design and implement PolicyLint to extract sharing and collection statements from privacy policies and identify self-contradictory policy statements. We perform a large-scale study on the privacy policies for 11,430 Android applications using PolicyLint and find that around 14.2% have potentially deceptive and ambiguous privacy policies due to self-contradictory statements. Third, we combine insights gained from our prior studies to provide a formal specification for an entity-sensitive (e.g., first-party vs. third-party) and negation-sensitive (e.g., collect

vs. not collect) flow-to-policy consistency model. We design and implement POLICHECK to perform a large-scale study on 13,796 Android applications and their corresponding privacy policies and find that up to 42.4% of applications either incorrectly disclose or omit disclosing their privacy-sensitive data flows.

Our characterization of the problem space, formal specifications, analysis techniques, and insights drawn from our empirical studies lay the foundation for identifying privacy-sensitive user data, precisely and soundly reasoning over privacy policies, and evaluating flow-to-policy consistency at scale. The findings from our empirical studies highlight significant privacy risks associated with exposing privacy-sensitive user data to applications and provide concrete examples of such cases that impact tens-to-hundreds of millions of users. Our results show the poor state of privacy disclosures for Android applications, which demonstrates the need for additional oversight and auditing by application markets and regulatory agencies. Further, our findings identify several future areas of research in assessing and preventing privacy risks, such as analyzing consent, improving the usability aspects of creating privacy policies, and introducing stronger privacy protection mechanisms.

© Copyright 2019 by Benjamin Eric Andow

All Rights Reserved

Privacy Risks of Sensitive User Data Exposure in Mobile Ecosystems

by  
Benjamin Eric Andow

A dissertation submitted to the Graduate Faculty of  
North Carolina State University  
in partial fulfillment of the  
requirements for the Degree of  
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2019

APPROVED BY:

---

Dr. Alexandros Kapravelos

---

Dr. Bradley Reaves

---

Dr. Laurie Williams

---

Dr. William Enck  
Chair of Advisory Committee

## **DEDICATION**

To Dandan and Evelyn.

## **BIOGRAPHY**

Benjamin Andow was born and raised in Garfield Heights, Ohio. He received his Bachelor of Science in Computer and Information Science from Cleveland State University in May 2013. He joined the doctoral program at North Carolina State University that following August. During his time at North Carolina State University, he interned at the IBM T.J. Watson Research Center in Yorktown Heights, New York and at Hewlett-Packard Laboratories in Palo Alto, California. He also served as the Program Committee Chair for the 2017 Mobile Security Technologies Workshop (co-located with the IEEE Symposium on Security and Privacy) and the Web Chair for both the 2018 and 2019 IEEE Symposium on Security and Privacy. He will be joining the IBM T.J. Watson Research Center after completing his doctorate.

## ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. William Enck, for his guidance and support throughout my six years of graduate school at North Carolina State University (NCSU). His mentoring, advice, unconditional support, and understanding helped me evolve and advance as an independent academic researcher and was one of the leading factors for the successful completion of my Ph.D. I wholeheartedly agree with the statement that choosing the right advisor is crucial to graduate school success.

I would also like to thank Dr. Bradley Reaves for his guidance, advice, and expertise, which began almost immediately since he arrived at NCSU. I can state with utmost certainty that my last two works of this dissertation would not have been nearly as strong without Brad's guidance. Brad's perspectives and insights on approaching and conducting research and empirical studies has had a significant positive impact on my research approach and helped me develop into a stronger independent researcher.

I would like to thank my collaborators, specifically, Dr. Kapil Singh, Dr. Tao Xie, Dr. Adwait Nadkarni, Sigmund Albert Gorski III, Dr. Luke Deshotels, Dr. Jason Gionta, Samin Yaseer Mahmud, Justin Whitaker, Akhil Acharya, Dr. Wei Yang, Dr. Serge Egelman, Dr. Heqing Huang, Dr. Xusheng Xiao, Peipei Wang, Rui Shu, and Dr. Xiaohui Gu for their efforts and guidance. I would also like to thank my doctoral committee members, alphabetically, Dr. Alexandros Kapravelos and Dr. Laurie Williams, along with the aforementioned members, for their feedback and discussions during the course of my degree. I would like to explicitly thank, Dr. Kapil Singh, for his unbiased advice, support, lively discussions, and guidance throughout the years for my academic research, my internship at IBM, and guiding me through the next steps of my career. I would also like to thank, Dr. Tao Xie, for his discussions, insights, and valuable feedback on both my research and writing.

I would like to thank the other faculty members of the Wolfpack Security and Privacy Research Lab, both past and present, for their feedback throughout the years, specifically, Dr. Alessandra Scafuro, Dr. Jessica Staddon, and Dr. Douglas Reeves. I would also like to thank all of my labmates, both past and present, for their support and expertise, specifically, Dr. Adwait Nadkarni, Dr. Luke Deshotels, Dr. Jason Gionta, Sigmund Albert Gorski III, Dr. Micah Bushouse, Isaac Polinsky, Sanghyun Ahn, Akash Verma, Kunal Patel, Dr. TJ O'Connor, Dr. Ashwin Shashidharan, Vasant Tendulkar, Kyle Martin, Russell Meredith, Andrew McNamara, Samin Yaseer Mahmud, Justin Whitaker, Lucas Enloe, Abida Haque, Jordan Jueckstock, Iffat

Anjum, Dr. Zach Jorgensen, Sanket Goutam, Athishay Kiran, and Shaown Sarker.

I would also like to thank Dr. Haodong Wang for his guidance and for introducing me to security research during my undergraduate degree at Cleveland State University. I likely would have not considered pursuing a Ph.D. had it not been for his guidance during my undergraduate research experience.

I would like to especially thank my parents, Janice and Robert Andow, and my in-laws, Daozhi Ju and Xiaoyun Wang, for their support throughout the years and for understanding when deadlines took priority over visiting home. Most importantly, I want to thank my wife, Dandan, and daughter, Evelyn, for their unwavering encouragement, understanding, patience, support, companionship, and love. You both have been my source of light and happiness through the long hours and occasional sleepless nights that I have spent working on this dissertation. I love you both dearly.

The works in this dissertation were supported in part by NSF grants CNS-1513690, CNS-1513939, and a Google Faculty Research Award. Any findings and opinions expressed in this material are those of the author and do not necessarily reflect the views of the funding agencies.

# TABLE OF CONTENTS

<b>LIST OF TABLES</b> .....	<b>viii</b>
<b>LIST OF FIGURES</b> .....	<b>ix</b>
<b>Chapter 1 Introduction</b> .....	<b>1</b>
1.1 Thesis Statement .....	3
1.2 Contributions .....	5
1.3 Thesis Organization .....	6
<b>Chapter 2 Foundations and Related Work</b> .....	<b>8</b>
2.1 Android Applications .....	9
2.2 Program Analysis .....	10
2.2.1 Permission Request Analysis .....	10
2.2.2 Permission-protected API Call Analysis .....	11
2.2.3 Information Flow Analysis .....	12
2.3 Privacy Policies .....	15
<b>Chapter 3 UiRef: Analysis of Sensitive User Inputs in Android Applications</b> .....	<b>19</b>
3.1 Introduction .....	20
3.2 Background .....	23
3.3 Problem and Challenges .....	23
3.4 System Approach and Design .....	26
3.4.1 Layout Extraction .....	26
3.4.2 Label Resolution .....	28
3.4.3 Semantics Resolution of Input Widgets .....	30
3.4.4 Outlier Analysis .....	33
3.5 Evaluation .....	36
3.5.1 Layout-Extraction Performance .....	36
3.5.2 Label-Resolution Performance .....	37
3.5.3 Semantics-Resolution Performance .....	39
3.6 Security and Privacy Analysis .....	41
3.6.1 Sensitive Information Requests .....	41
3.6.2 Identifying Anomalous Input Requests .....	43
3.7 Discussion .....	46
3.8 Related Work .....	47
3.9 Conclusion .....	48
<b>Chapter 4 PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play</b> .....	<b>49</b>
4.1 Introduction .....	50

4.2	PolicyLint	53
4.3	Preprocessing Privacy Policies	55
4.3.1	Ontology Generation	57
4.3.2	Policy Statement Extraction	61
4.3.3	Policy Contradictions	67
4.4	Privacy Study	73
4.4.1	General Policy Characteristics	73
4.4.2	Candidate Contradictions	74
4.4.3	Deeper Findings	78
4.4.4	Notification to Vendors	84
4.5	Limitations	85
4.6	Related Work	86
4.7	Conclusion	88
<b>Chapter 5</b>	<b>Actions Speak Louder than Words: Entity-sensitive Privacy Policy and Data Flow Analysis with POLICHECK</b>	<b>89</b>
5.1	Introduction	90
5.2	Flow-to-Policy Consistency	93
5.2.1	Clear Disclosures	93
5.2.2	Vague Disclosures	94
5.2.3	Omitted Disclosure	95
5.2.4	Incorrect Disclosure	95
5.2.5	Ambiguous Disclosure	96
5.3	Consistency Model	97
5.3.1	Data Flow and Policy Statements	97
5.3.2	Ontological Operations	98
5.3.3	Consistency	100
5.4	Design	104
5.5	Flow-to-policy Consistency Characterization	106
5.5.1	Consistency Analysis	106
5.5.2	Evaluation	113
5.5.3	Additional Case Studies	117
5.6	Limitations	120
5.7	Related Work	121
5.8	Conclusion	122
<b>Chapter 6</b>	<b>Conclusion and Future Directions</b>	<b>124</b>
<b>BIBLIOGRAPHY</b>		<b>128</b>
<b>APPENDIX</b>		<b>144</b>
Appendix A	Appendix	145

## LIST OF TABLES

Table 3.1	Semantic Bucket Excerpt (5/78) . . . . .	34
Table 3.2	Performance Evaluation Results . . . . .	37
Table 3.3	Disambiguation Evaluation Results . . . . .	38
Table 3.4	Consolidated Sensitive Information Requests . . . . .	42
Table 4.1	NER Performance: Comparison of spaCy’s stock en_core_web_lg model versus our domain adapted model . . . . .	58
Table 4.2	Lexicosyntactic patterns for subsumptive relations . . . . .	59
Table 4.3	Seed terms used for ontology construction . . . . .	60
Table 4.4	SoC verbs used by PolicyLint . . . . .	62
Table 4.5	Sentence Generation Templates . . . . .	64
Table 4.6	Rules that transform a CPS into an SPS . . . . .	68
Table 4.7	Contradictions ( $C$ ) and Narrowing Definitions ( $N$ ) . . . . .	70
Table 4.8	First-Party Synonyms . . . . .	72
Table 5.1	Types of conflicting policy statements in a privacy policy: logical con- tradictions ( $C_{1-5}$ ), flow-sensitive contradictions ( $C_{6-12}$ ), and narrowing definitions ( $N_{1-4}$ ). . . . .	101
Table 5.2	Data Flows and Apps for each Disclosure Type . . . . .	107
Table 5.3	Sensitivity Analysis of Flow-to-Policy Consistency Models for Classify- ing Disclosure Types: Entity-insensitive analysis results in the frequent misclassification of data flows. . . . .	114
Table A.1	Sensitive User Input Request Outliers . . . . .	146

## LIST OF FIGURES

Figure 3.1	Example Android layout and corresponding XML specification with stripped namespaces . . . . .	21
Figure 3.2	Screenshot of the Android layout in Figure 3.1 . . . . .	21
Figure 3.3	UiRef Architecture . . . . .	24
Figure 4.1	Overview of PolicyLint . . . . .	53
Figure 4.2	Transformation of Example 2 from its dependency-based parse tree to its DED tree. . . . .	61
Figure 4.3	CDF of Contradictory Policy Statements: 50% of contradictory policies have 2 or fewer logical contradictions. . . . .	75
Figure 4.4	Ratio of Contradictory Policies per Category: 79.7% of contradictory policies have at least one or more logical contradictions ( $C_{1-5}$ ) that may indicate potentially deceptive statements. . . . .	75
Figure 4.5	Average number of unique candidate contradictions per category: Logical contradictions ( $C_{1-5}$ ) and narrowing definitions ( $N_{1-4}$ ) are both widely prevalent across Google Play categories. . . . .	76
Figure 4.6	Log 10 Frequency of Data Type Pairs in Contradictions: Negative statements that discuss broad categories of data are problematic (i.e., $C_{1-5}$ ). . . . .	76
Figure 4.7	Email Template . . . . .	84
Figure 5.1	POLICHECK determines the consistency of a mobile application’s data flows to its privacy policy. . . . .	91
Figure 5.2	Policies are more clear about the first-party receiving a specific data type than a third-party doing so. . . . .	108
Figure 5.3	Policies often discuss the sharing of Android and advertising IDs in vague terms. . . . .	109
Figure 5.4	Policies are likely to use vague terms to describe both data types and entities. . . . .	109
Figure 5.5	POLICHECK identified 1,912 incorrect disclosures across 719 applications . . . . .	111
Figure 5.6	Policies often do not disclose when they share Android and advertising IDs with third parties. This may be due to the perception that such collection is implied when they disclose that they use an advertiser. . . . .	111
Figure 5.7	Privacy policies are often contradictory when they discuss the sharing of Android and advertising IDs. . . . .	112
Figure 5.8	The majority of applications have less than five data flows for each disclosure type, but a small percentage have significantly more. . . . .	117

## CHAPTER

# 1

## INTRODUCTION

With over 4.3 billion smartphone subscriptions worldwide and an estimated 7.2 billion by 2023 [Eri], mobile devices are continuing to become increasingly ubiquitous. In certain countries, such as the United States, mobile devices have become so pervasive that over three-fourths of adults own a smartphone [Pew]. In addition, mobile devices have surpassed traditional personal computers to become the primary method in which users access the internet, with 73% of internet traffic predicted to originate from mobile devices in 2018 [Zen].

When using mobile devices, users interact with native applications on the device. Applications are often claimed to be one of the main driving forces for the rise in mobile device popularity. From entertainment purposes to personal finance management, there is an application to assist with almost any conceivable task. Due to the utility provided by these applications, their use has become part of mobile device owners' daily routines. In fact, users in the United States are spending around 150 minutes per day on average using applications [Appa], and are using 10 unique applications per day and around 30 unique applications per month on average [Appa].

While manufacturers and vendors oftentimes bundle applications with the device, centralized application markets, such as Google Play and Apple's App Store, are the predominant way that users discover and install applications. These applications are generally developed by third-parties and the barrier-to-entry is relatively low to release an application. In fact, there are currently over 2.5 million applications on Google Play [Appd] and 2.1 million applications on Apple's App Store [Appc]. Users are actively seeking out and installing applications from these markets. In 2017 alone, global application downloads reached 175 billion and revenue generated in the application markets surpassed \$86 billion [Appa].

While providing functionality to users, applications consume a wide array of sensitive data, such as email addresses, date of birth, passwords, location data, and device identifiers. In many cases, it is required to provide applications with sensitive data to allow them to complete their given tasks. For example, when making a purchase in a shopping application, users generally must provide their credit card information, billing and shipping addresses, their name, and their email address to complete the transaction. In general, applications obtain sensitive data in two different ways. First, application may request that the user directly input sensitive data through user interfaces (e.g., email address, date of birth, password). Second, applications may automatically collect certain types of sensitive data through the use of the API calls provided by the operating system (e.g., location data, device identifiers, incoming text messages).

Such sensitive data is an extremely valuable commodity for legitimate business purposes, but also for nefarious and illicit purposes. For example, applications may re-purpose data for ulterior motives, such as requesting an email address for processing a transaction and then also hashing the email address to share it with third parties for use as an identifier for targeted advertisements. While behaviors that lead to privacy loss are clearly not entirely positive aspects, they are also not inherently negative. In general, many users may not want to physically pay money for applications, yet monetization of applications is necessary to incentivize developers and drive the application market. Therefore, users generally make a trade-off between privacy loss and the utility provided by the application. From a legal standpoint, these behaviors that lead to privacy loss are commonly deemed acceptable, as long as they are disclosed within the application's privacy policy. On the other side of the spectrum, a malicious application may also request the sensitive data in the guise of performing some task, but instead use the data for illicit purposes (e.g., fraud, spam, blackmail). Due to the value of user data, it is a high-value target for potential cases of

misuse. Thus, auditing an application’s usage of privacy-sensitive user data is required to reduce potential risks of abuse.

## 1.1 Thesis Statement

To reduce the risk of users encountering harmful applications, application markets generally assume the role as the initial gatekeepers that determine which applications become available to users. At a minimum, application markets generally place the following two requirements on applications before they are published to the market: (1) the application’s code must pass the market’s security and privacy vetting procedures; and (2) the application must provide a privacy policy if it handles sensitive user data.

To vet applications, application markets, such as Google Play, use a combination of static analysis, dynamic analysis, and manual application review (when needed) [Anda; Andb]. While the exact technical details of the techniques that markets deploy is not publicly available, the currently deployed analysis techniques are not the panacea for all harmful applications, as demonstrated by various news reports [Sym; Mal; Tec; Mas] and prior research [Han13b; Enc10; OM12]. One limiting factor is that performing more than simple and lightweight techniques is impractical at the market-level due to the scale of their analysis. Further, based on Google Play’s definitions of potentially harmful applications [Anda; Andb], their analysis scope appears similar to the majority of the prior research [Enc10; Arz14b; FWR14] that only consider sensitive data obtained through the well-defined API calls provided by the operating system that denote semantics of the data returned (e.g., `getLastLocation()` returns location data), which omits the privacy risks associated with sensitive user input disclosure to applications. We attribute this limitation to the difficulty of resolving the semantics of user input, as user input is obtained through API calls that do not denote the semantics of the data returned. For example, a user’s address and credit card number are both fetched from the same API method `getText()`. Resolving the semantics of user input is required for automated analysis of user input.

In an attempt to ensure transparency of an application’s privacy risks, application markets also generally require that developers provide a link to their privacy policy that describes the application’s privacy practices if the application handles sensitive user data [Goo]. However, the findings from our studies in Chapters 4 and 5, along with the results of prior research [Zim17b; Sla16; Wan18], provide evidence that application markets, such as Google

Play, do not perform a deep inspection, if any, on the content of privacy policies to identify problematic areas, such as contradictory privacy policy statements and whether the application complies with their given privacy policy. Existing techniques [Zim17b; Sla16; Wan18] that analyze data sharing and collection practices of privacy policies extract information at a coarse-granularity (i.e., paragraph-level or document-level), which does not allow for a precise reasoning over privacy practices. We attribute this limitation to the difficulty of extracting and representing sharing and collection practices at a fine-grained granularity (e.g., sentence-level) while capturing the correct meaning of the practice (i.e., “collect” or “not collect”) and the entities involved (i.e., first-party versus third-party).

The focus of this dissertation is to characterize and analyze the privacy risks associated with disclosing privacy-sensitive user data to applications. To enable analysis of user data and their uses, we design and implement several analysis and information extraction techniques that improve the state of the art. We observe that natural language processing and text analytics can be leveraged to address much of the aforementioned limitations.

Through our characterization of privacy-sensitive user data and privacy policies, and our design and implementation of analysis techniques, we provide the technical underpinnings required for analyzing the privacy risks associated with disclosing sensitive user data to applications at-scale. We use these techniques to perform large-scale privacy studies on Android applications from Google Play and their corresponding privacy policies. In particular, we seek to identify the privacy implications of: (1) how applications are requesting and handling privacy-sensitive user data; (2) how privacy-sensitive user data is discussed within privacy policies; and (3) how applications use privacy-sensitive user data in ways that conflicts their privacy policy.

We observe that considering the entities involved in an application’s privacy-sensitive user data practices (e.g., first-party vs. third-party) along with whether it is discussed with a positive or negative meaning (e.g., collect vs. not collect) is vital for assessing privacy risks of applications. The findings and observations from our studies of applications and their privacy policies informs the following thesis statement.

---

*Identifying privacy violations by mobile applications requires fine-grained entity-sensitive and negation-sensitive analysis of the data flows and text within privacy policies.*

---

Evidence of this thesis can be found throughout Chapters 3-5. Findings 5 and 6 in Chapter 3 demonstrates that applications are stealthily sharing privacy-sensitive user data with third-parties and including third-party SDKs in their applications that directly display GUIs to collect such data, which demonstrates the importance of considering the entities that are receiving the data. In Chapter 4, we find that 14.2% of privacy policies contain contradictory policy statements, which may be indicative of potentially deceptive and ambiguous policy statements. Identifying contradictory statements requires entity-sensitive, negation-sensitive, and holistic analysis of privacy policies. In Chapter 5, we demonstrate that prior works that use entity-insensitive and negation-insensitive models may falsely classify up to 38.4% of applications as having privacy-sensitive data flows consistent with their privacy policy. With our entity-sensitive and negation-sensitive flow-to-policy consistency model, we find that up to 42.4% of applications either incorrectly disclose or omit disclosing their privacy-sensitive data flows in their privacy policies.

## 1.2 Contributions

In this dissertation, we make the following main contributions:

- *We characterize the space of privacy-sensitive user data sources to identify what types of data applications are requesting from users.* We design and implement an analysis framework to resolve semantics of sensitive user inputs. Resolving the semantics of sensitive user inputs is a fundamental building block to enable automated analysis of such data. We propose UiRef (*WiSec'17* [And17]) to resolve semantics of sensitive user inputs. UiRef uses novel techniques for semantics analysis of sensitive user inputs to considerably improve over the current state of the art, such as resolving ambiguity using word embeddings. We demonstrate the utility of UiRef by performing security and privacy analysis through a large-scale study on 50,162 Android applications. Our study highlights various forms of questionable, and potentially malicious, practices by developers and emphasizes the need to consider user inputs in the security and privacy analysis of mobile apps.
- *We characterize the semantics of sharing and collection statements within privacy policies and demonstrate the importance of holistic analysis through our identification and formalization of self-contradictory policy statements.* We propose PolicyLint

(*USENIX Security*'19 [And19]), a privacy policy analysis tool inspired by lightweight static analysis tools like `Lint`. PolicyLint automatically generates ontologies capturing semantic relationships between data types and entities from a large corpus of privacy policies. It then uses sentence-level natural language processing to capture both positive and negative statements of data collection and sharing and the entities involved with those statements. We formally model 9 types of contradictions that result from semantic relationships between terms, providing an algorithmic method to detect contradictory policy statements. In a study of 11,430 privacy policies from mobile apps, we are the first to find that such contradictions are *rampant*, affecting 14.2% of policies. We manually reviewed 510 contradictions across 260 policies, finding that many contradictions are indeed indicators of misleading or problematic practices.

- *We formally specify an entity-sensitive and negation-sensitive flow-to-policy consistency model to detect when applications are inconsistent with their own privacy policies.* We design and implement POLICHECK to analyze flow-to-policy consistency at-scale. We use POLICHECK to study 13,796 applications and their privacy policies and find that at least 5.2% of apps may have data flows that are direct conflict with their policies. Our results demonstrate the significance of considering the entity: without considering entity, prior approaches would falsely classify up to 38.4% of applications as having privacy-sensitive data flows consistent with their privacy policies. These false classifications include data flows to third-parties that are omitted (e.g., the policy states only the first-party collects the data type), incorrect (e.g., the policy states the third-party does not collect the data type), and ambiguous (e.g., the policy has conflicting statements about the data type collection). By defining a novel automated, entity-sensitive flow-to-policy consistency analysis, POLICHECK provides the highest-precision method to date to determine if apps properly disclose their privacy-sensitive behaviors.

### 1.3 Thesis Organization

The goal of this dissertation is to understand the privacy risks associated with disclosing sensitive user data to applications by analyzing how applications are collecting and sharing such data, how these practices are discussed within their privacy policy, and when they

are in conflict with each other. To achieve this goal, we focus our analysis to Android applications, which runs on an operating system that accounts for 85.9% of the market share in 2017 [Gar].

The remainder of this dissertation is organized as follows. In Chapter 2, we discuss the current state of security and privacy analysis techniques on Android and privacy policies. In Chapter 3, we present an analysis framework to resolve the semantics of sensitive user input and perform a large-scale security and privacy analysis of sensitive user input requests. Chapter 4 presents a privacy policy analysis framework that extracts fine-grained sharing and collection statements from privacy policies to identify contradictory policy statements and presents the results from our large-scale study on self-contradictory privacy policies. In Chapter 5, we present an entity-sensitive and negation-sensitive flow-to-policy analysis consistency model to determine when privacy-sensitive data flows are inconsistent with the application's privacy policy and present the results from our large-scale study on flow-to-policy consistency. Chapter-6 provides future directions, takeaways, and concludes.

## CHAPTER

# 2

## FOUNDATIONS AND RELATED WORK

A generally agreed upon notion is that people have a right to privacy, which has been affirmed by court cases [US65] and various regulations [USC99; USC70; USC16; Law03; PEU16]. Samuel Warren and Louis Brandeis were the first to explicitly mention a person's right to privacy [WB90], which was in direct response to the advancements in technology of the era, such as the invention of the Kodak's portable box camera in 1888. Specifically, Warren and Brandeis's article states, "Instantaneous photographs and newspaper enterprise have invaded the sacred precincts of private and domestic life; and numerous mechanical devices threaten to make good the prediction that 'what is whispered in the closet shall be proclaimed from the house-tops. [WB90]'" The sentiment of this statement is echoed and significantly amplified when considering privacy implications imposed by mobile devices. Given the types of data to which mobile devices have access and their near persistent state of connectivity to the outside world, mobile devices have the means to encroach on a user's privacy to an extent that almost every detail about their daily life runs the risk of being "proclaimed from the house-tops" (i.e., becoming public knowledge).

Fortunately, various techniques have been proposed to identify or communicate the

security and privacy risks of applications to users. These techniques are the main focus for the remainder of this chapter and serves as the basis for the remaining chapters of this dissertation. Note that while mobile device operating systems, such as Android, deploy a variety of protection mechanisms (e.g., permissions-based security model, file-based discretionary access control, application sandboxing) and prior research propose a wide-range of additional protection mechanisms (e.g., decentralized information flow control [NE13; Nad16; Jia13; XW15], privilege-separation [RK13; Dav12; She12; Pea12], mandatory access control [Heu14; Bug13; SC13]), we limit our discussion to security and privacy analysis techniques, as the focus of this dissertation is on analyzing applications to understand security and privacy risks of sensitive data disclosure rather than enhancing protection mechanisms.

## **2.1 Android Applications**

Android is an application-centric operating system where users perform tasks and interact with the device via applications. Android applications are composed as a set of components that correspond to the entry points of the application. Users directly interact with components called activities, which typically represent a single graphical user interface in the application. Services are components that run in the background without interaction from the user to typically perform a long-running task. Broadcast receivers are components that accept system or application events to perform some short-running task. Content providers are components that provide a database-style interface for managing and sharing application data. Each component has their own life-cycle, which are a set of callbacks that are invoked whenever the component's status changes (e.g., create, start, pause, resume, stop).

In most cases, components are started with by intent messages, which is an abstraction that describes the operation to be performed, such as starting an activity, and optionally includes any data required to perform said operation, such as a URI. Intents can be explicitly addressed to components by specifying the component's name and application's namespace (i.e., package name). Alternatively, intents can be implicitly addressed by specifying an action string, which allows the operating system to resolve which component to deliver the intent. Developers specify intent filters on components, which specifies the intent actions that a specific component registers to receive. To avoid unwanted interactions with

components, developers also specify whether components are publicly accessible to other applications (i.e., exported) and also allow the developer to specify any permissions that are required to send an intent to the component.

Each application includes a manifest file (`AndroidManifest.xml`) that is specified before compilation and packaged with the application. All components along with their intent filters, exported attribute, and required permission must be specified in the manifest file, except for certain cases of broadcast receivers, which can be constructed dynamically at runtime. As Android uses permissions to protect certain sensitive API calls that provide access to protected resources (e.g., GPS, internet, device information), developers must also specify the permissions to which their application requests access.

## **2.2 Program Analysis**

Program analysis is performed by physically running the application to execute its code (dynamic analysis), directly analyzing the application's code without executing its code (static analysis), or a combination of both approaches (hybrid analysis). While program analysis techniques were initially proposed for compiler optimization and software reliability testing, these techniques have been adapted and applied to identify security and privacy risks of programs. While a myriad of research has used program analysis to identify security and privacy risks for traditional programs, analyzing Android applications pose new challenges, such as containing multiple execution entry points and having complex component life-cycles. Due to these challenges, various program analysis approaches have been proposed for assessing the security and privacy risk of Android applications.

### **2.2.1 Permission Request Analysis**

One of the first techniques proposed for analyzing the security and privacy implications of mobile applications is through the analysis of the application's permissions requests in its manifest file. Kirin [Enc09] uses permission requests to statically define sets of security rule to identify potential malicious behaviors and mitigate malware installation. Felt et al. [Fel11a] use permission requests (i.e., raw counts, common requests, and common sets of requests) to reason about malware. Similarly, Zhou and Xiang [ZJ12] use permission requests to reason about the differences between malware and benign applications. Drebin [Arp12] and DroidApiMiner [Aaf13] use permission requests as a subset of their

feature vector to detect malware. Barrera et al. [Bar10] use the self-organizing maps on permission requests in Android applications to study the effectiveness of Android’s permission model. WHYPER [Pan13], CHABADA [Gor14], and AutoCog [Qu14] examine permission request to application fidelity by using natural language processing.

Several prior works used permission requests to rank the risk of applications. Peng et al. [Pen12] uses probabilistic generative models to rank the risks of installing applications based on the application’s permission requests. Sarma et al. [Sar12] uses an application’s permission requests in comparison to the permission requests of other applications within the same application category to rank the risk of applications. MAST [Cha13] uses permission requests along with other lightweight features, such as intent filters, presence of native code, presence of zip files, to triage malware for deeper analysis. DroidRanger [Zho12] uses permission requests to filter applications for deeper behavioral analysis to detect malware. MalloDroid [Fah12] uses internet permission requests to filter applications for further analysis to check for SSL vulnerabilities.

### **2.2.2 Permission-protected API Call Analysis**

Due to the coarse granularity of Android’s permissions, other techniques propose to analyze the presence of permission-protected API calls to identify security and privacy risks in mobile applications. Several prior works analyzed the usage of permission-protected API calls to detect over-privileged applications. Stowaway [Fel11b] uses dynamic analysis to create a permission-to-API call mapping to study whether applications are using the principle of least privilege. Vidas et al. [Vid11] uses lightweight static analysis to identify discrepancies in permission requested and permission-protected API calls within the code to detect and notify developers of over-privilege. Wu et al. [Wu13] use permission requests and the permission-protected API calls invoked to determine over-privilege in vendor applications.

Other works leveraged the presence of permission-protected API calls within the application’s code to identify malware or privacy risks. Drebin [Arp12] uses lightweight static analysis to detect the presence of permission-protected API calls within the application’s code, which it uses in its feature vector to detect malware. DroidApiMiner [Aaf13] uses the presence of permission-protected API calls in the application’s code along with package level information and parameters to classify malware. AdRisk [Gra12b] performs reachability analysis from an application’s entry points to permission-protected API calls that

they classify as dangerous (e.g., invocation causes financial loss) to study the risks of advertisement libraries. Zimmeck et al. [Zim17b] use the presence of permission-protected API calls in the application’s code to check privacy policy compliance. Wijesekera et al. log invocations of permission-protected API methods to gauge whether users expected the access to occur. Nguyen et al. [Ngu19] map security and privacy-relevant user reviews to permission-protected API calls and demonstrate that security and privacy-relevant user reviews lead to privacy improvements. Han et al. [Han13a] compare API calls within iOS and Android applications, and show iOS applications call significantly more privacy-sensitive APIs than their Android counterparts. Moreover, other works [Bac16b; Ma16] enhance this analysis by proposing code analysis techniques to resolve the party responsible for the permission-protected API calls.

In addition to analyzing the permission-protected API calls, other works resolved additional application context of the call to identify security and privacy risks in applications. RiskRanker [Gra12a] resolves the callbacks that cause the invocation of permission-protected APIs to use as signatures for malware detection. AsDroid [Hua14] resolves the graphical user interfaces displayed when permission-protected APIs are invoked to detect stealthy behaviors. AppContext [Yan15] resolves the callbacks and conditions that cause the execution of permission-protected API calls to identify malware. TriggerScope [Fra16] similarly resolves the conditions that trigger permission-protected API calls to identify logic bombs.

### **2.2.3 Information Flow Analysis**

While analyzing permission-protected API calls denotes that an application is accessing a resource, API calls alone do not indicate how the application is using the resource. Therefore, prior works have proposed tracking information flows of an application or performing deep packet inspection at the network-level to determine how the application is using the sensitive data.

Much of the focus of prior work has been on performing either static or dynamic taint tracking. TaintDroid [Enc10] designs and implements dynamic taint tracking on Android to study privacy leaks in applications. BayesDroid [TR14] extends TaintDroid by using Bayesian classification to classify whether data flows to sinks are legitimate or illegitimate. TaintArt [Sun16] design and implements a new dynamic taint tracking on Android that works with the ART compiler to track privacy-sensitive data flows. Enck et al. [Enc11] de-

compile applications and use static data flow analysis to determine when sensitive data, such as device identifiers, flow to the network. PiOS [Ege11] and AndroidLeaks [Gib12] are the initial static analysis duals of TaintDroid for iOS and Android, respectively. FlowDroid [Arz14b] uses context, flow, field, object, and life-cycle sensitive static taint analysis to detect privacy leaks in applications. DroidSafe [Gor15] models the Android runtime to increase precision of static information flow analysis on Android. IccTA [Li15] resolves inter-component communication to extend FlowDroid to detect cross-component privacy leaks. Amandroid [FWR14] provides a general static analysis framework that can detect data leaks, data injection vulnerabilities, and API misuse. AppIntent [Yan13] uses static analysis to build test cases of events and interactions required to trigger a sensitive data leak and then runs the test case to execute the application and observe the sensitive data leak. MudFlow [Avd15] uses static taint analysis to extract an application's sensitive data flow behaviors to differentiate between normal flows in benign applications and abnormal flows in malicious applications. FlowCog [Pan18] uses NLP to analyze the text within graphical user interfaces to infer whether the sensitive data leaks detected with static taint tracking are justified by the user interface. Feng et al. [Fen15] identify applications that aggregate privacy-sensitive user data by tracking accesses of such data and estimating the flow of the data by building dynamic link-ability graphs. Oltrogge et al. [Olt18] use a combination of static and dynamic analysis to analyze applications created by online application generators and find four application generators were stealthily collecting privacy-sensitive user data. Bhoraskar et al. [Bho14] identify several cases of advertisements requesting privacy-sensitive user data from children by static analysis to identify execution paths and binary rewriting for targeted execution. Zhang et al. [Zha15] create behavior graphs describing triggering conditions and information flows to generate application descriptions that disclose such behaviors. AutoPPG [Yu15] characterizes an application's data flows using static analysis and generates sentences for privacy policies to describe the behaviors. Reaves et al. manually analyze data flows involving financial and account information in branch-less banking applications to identify vulnerabilities in processing financial transactions or authenticating the user. Bashir et al. [Bas16] propose using re-targeted ads for tracking information flows between different ad exchanges.

Instead of tracking the propagation of information throughout the application and system, other approaches have examined network traffic to determine how applications are using sensitive data. The Lumen Privacy Tool [Raz15] (formerly HayStack) introduces

an on-device network monitoring tool that uses the VPN API to intercept network traffic of Android applications. Reyes et al. [Rey17] use the Lumen Privacy Tool to study the sensitive data leaks in Android applications targeted towards children to detect potential COPPA violations. Reyes et al. [Rey18] perform a follow-up longitudinal study and find that around one out of five children’s applications collect identifiers or personally identifiable information. Razaghpanah et al. [Raz18] identify advertising and tracking services through analysis of network traffic and present insights into the ecosystems and tracking behaviors. Ren et al. [Ren18] perform a longitudinal privacy study of sensitive data leaks on application updates over 8 years and find that applications are requesting more sensitive data in general. Reardon et al. [Rea19] monitor network traffic to determine when applications transmit privacy-sensitive user data, but do not have the permission to obtain such data. Han et al [Han19] use the Lumen Privacy Tool to study the privacy behaviors of free and paid applications and found that 38% of paid applications have the same privacy-invasive behaviors and the free versions. Continella et al. [Con17] propose detecting privacy leaks by using blackbox differential analysis to detect deviations from normal network traffic resulting from changing values of privacy-sensitive user data.

To track information flows, analysis techniques require identifying the sources of privacy-sensitive user data. Much of the initial works [Enc10; Enc11; FWR14; Arz14b; Li15; Yan13; Avd15] rely on a list of well-defined API calls provided by the system that return privacy-sensitive user data. Several prior works [Fel11b; Au12; Arz14a; Bac16a] propose various program analysis techniques to create a permission-to-API call mapping. In recent years, research efforts [Hua15; Nan15; Xia19] have focused on resolving semantics of privacy-sensitive user input fields to enable analysis of such data, including our work presented in Chapter 3. In a similar vein, Demetriou et al. [Dem16] resolve semantics of privacy-sensitive user data stored in structured files (e.g., XML, SQL) to track exposure of sensitive data to advertisers. Nan et al. [Nan18] resolves the semantics of privacy-sensitive data obtained via third-party APIs by analyzing the textual names of variables and method signatures. Several other works identify seemingly non-sensitive information sources that can infer privacy-sensitive user data. Cabañas et al. [Cab18] demonstrates that sensitive personal information can be inferred from Facebook’s automatic labeling of user interests for ad preferences and measures the impact on EU citizens to assess GDPR compliance. Hassan et al. [Has18] assess the privacy risks of fitness trackers and find that locations allegedly protected by privacy preferences could be inferred from other public activity.

## 2.3 Privacy Policies

Privacy policies are the primary mechanism by which mobile applications inform users about data collection and sharing practices. In this section, we discuss studies and techniques proposed by prior research to analyze the content privacy policies.

Various research efforts opt for manual analysis to perform exploratory studies on the content of privacy policies or to collect annotations. Bhatia and Breaux [BB18] manually annotate 5 privacy policies with semantic roles to study which roles result in incomplete descriptions of data practices and how the presence and absence of these roles impact users' perceptions of privacy risk. Bhatia and Breaux [BB17] manually annotate 5 policies to identify how purposes behind data collection are expressed and define categories of data purposes and identify a set of natural language patterns that could potentially be used for information extraction. Bhatia et al. [Bha16] manually analyze a set of 15 privacy policies to identify vague modifiers (e.g., "may," "some," "as needed") and study how these vague terms influence users' perceptions of privacy risk and willingness to share data. Note that our definition of vagueness in Chapter 5 is defined in terms of specificity of the terms used to disclose the specific data types and entities involved in data flows. While our definition of vagueness differs from Bhatia et al. [Bha16], they can be used to complement each other's analysis of privacy policy vagueness, as they are not in conflict. Breaux and Schaub [BS04] propose a task decomposition approach for gathering crowd-sourced annotations of privacy policies to reduce the cost of manual extraction while increasing the coverage. Wilson et al. [Wil16a] demonstrate that crowd-sourcing non-trivial privacy policy annotations is feasible and propose a method to improve annotation by highlighting paragraphs associated with certain data practices relevant to the annotation task. Wilson et al. [Wil16b] have skilled experts annotate a set of 115 privacy policies and identify several future directions that can leverage this dataset. Breaux et al. [BR13; Bre15] manually analyze a privacy policy for a case study to populate their Description Logic and find a few cases of conflicting statements.

As manual analysis is not feasible for large scale analysis, other works have focused on automated analysis of privacy policy content. Cranor et al. [Cra16] automatically analyzes the structured table-based model privacy forms of financial institutions and found many financial institutions are sharing your personal information without providing opt-out choices. Sathyendra et al. [Sat17] propose a two-stage architecture to identify opt-out choices in privacy policy text with the goal of assisting users to learn about the privacy choices

available. Costante et al. [Cos12] use machine learning to evaluate privacy policy completeness in respect to a set of topics (i.e., privacy categories) that are commonly discussed in privacy policies. Costante et al. [Cos13] extract data collection practices from privacy policies by using named entity recognition and a manually constructed set of shallow sentence patterns. Brodie et al. [Bro06] define two sentence patterns to parse sharing and collection practices from privacy policies for their privacy policy authoring tool. Privee [ZB14] uses machine learning to assign privacy grades to six coarse-grained properties of a privacy policy (e.g., collection, encryption, ad tracking). Zaeem et al. [Zae13] apply data mining to extract coarse-grained summaries of privacy policies representing what information is used and how. Stamey and Rossi [SR09] apply topic modeling to identify ambiguity in privacy policies. Polisis [Har18] uses deep learning to assign labels to paragraphs in privacy policies that summarizes the practice, such as coarse grained categories of data collected and the reason for such collection.

Several prior works [Cra16; BR13; Bre15; Yu16] document the possibility of conflicting privacy policy statements. Cranor et al. [Cra16] analyzed the tabular model privacy forms of financial institutions and identified cases of self-contradictory statements when performing deeper inspection. Breaux et al. [BR13; Bre15] provide a formal language to detect conflicts in privacy policies for multi-tier applications manually. Similarly, Yu et al. [Yu16] developed a system that verifies the consistency of an app's privacy policy with the policies of the app's libraries. These prior works define a single class of broad contradictions. In contrast, as we describe in Chapters 4-5, we characterize various different types of contradictory statement and discuss their impact on privacy policies.

In addition to extracting sharing and collection practices, several works propose techniques for detecting privacy policy violations. Zimmeck et al. [Zim17b] extract sharing and collection practices at the document-level and check whether sensitive data types retrieved from permission-protected API calls are justified by the extracted privacy practices. Zimmeck et al. [Zim17a] study cross-device tracking between a mobile browser and desktop browser and manually identify cases where companies omit discussing their cross-device tracking in their privacy policies. Slavin et al. [Sla16] and Wang et al. [Wan18] performs a similar task as Zimmeck et al., but they extract sharing and collection practices at the paragraph-level and uses information flows rather than just considering permission-protected API calls. Further, Slavin et al. only consider sensitive data obtain through well-defined API calls while Wang et al. also consider a limited amount of sensitive user input. Yu et al. [Yu16] also

performs a similar task, but extracts sharing and collection practices at the paragraph-level. However, Yu et al. also analyzes consistency between the application's privacy policy and the privacy policies of third-party libraries that the application includes. Okoyomon et al. [Oko19] analyze privacy policies and the data flows of applications and find that applications are contradicting their statements in the privacy policy about secure data transmission. Further, they perform a keyword search for the term "third-party" or direct mentions of the name of the companies and find that around 10.5% omit disclosing the behavior. Reyes et al. [Rey17; Rey18] analyze COPPA compliance by analyzing privacy-sensitive data leaks in applications targeted towards children, but do not analyze the content privacy policies, as the data flow alone is considered a privacy violation. Degeling et al. [Deg18] study GDPR compliance by analyzing the existence of privacy policies and consent notices for websites served in the EU. Eijk et al. [Eij19] study the impact of cookie consent notices for websites requested from the EU, USA, and Canada.

Analyzing the usability and effectiveness of privacy policies is another well-researched focus area. Research has shown that privacy policies are hard to comprehend by users [MC08] and proposals have been made to simplify their understanding [Nor15; Ram14]. Gluck et al. [Glu16] perform usability analysis of privacy policy length's impact on user awareness and suggest that limiting the length of privacy policies may increase awareness. Milne et al. [Mil06] perform a longitudinal study on the readability of 312 privacy policies and finds that readability has declined and the length of policies have increased in size. Reidenberg et al. [Rei15] study mismatches between the meanings of privacy policy statements and expert, knowledgeable, and typical users understanding and find disagreement between experts reflect ambiguous phrases in privacy policies. Jensen and Potts [JP04] study usability aspects (e.g., accessibility, readability, evolution) over a set of 64 privacy policies and find significant usability issues that impact users ability to understand the privacy disclosures and compliance with regulations.

As privacy policies are known for being notoriously difficult to understand, other research has focused on improving the effectiveness of privacy notices. While standardizing privacy policy specification has been attempted [Cra02] with limited success [Mar14], mobile apps' privacy policies have generally failed to adhere to any standards. Kelley et al. [Kel09] propose a privacy notice label inspired by food nutrition labels and demonstrate its the effectiveness of communicating privacy risk to users in comparison to natural language text policies. Schaub et al. [Sch15] characterize the design of privacy notices and

discuss that privacy policies alone are oftentimes insufficient for informing users of privacy practices, but rather need to be accompanied by privacy notices. Cranor et al. [Cra06] create a P3P privacy agent to compare users' privacy preferences with machine-readable P3P privacy policies and alert the user of deviations. Tsai et al. [Tsa07] create a user interface for an e-commerce search engine that annotates results with privacy icons and privacy reports and finds that users were more likely to purchase from retailers that better protected their privacy. AutoPPG [Yu15] characterizes an application's data flows using static analysis and generates sentences for privacy policies to describe the behaviors. Other approaches [Lin14; Liu14; Rao16] have attempted to bridge the gap between users' privacy expectations and application policies.

## CHAPTER

# 3

# UIREF: ANALYSIS OF SENSITIVE USER INPUTS IN ANDROID APPLICATIONS

In this chapter <sup>1</sup>, our goal is to understand the broad implications of privacy-sensitive user data requested via application user interfaces. To this end, we propose UiRef (User Input REsolution Framework), an automated approach for resolving the semantics of user inputs requested by mobile applications. UiRef's design includes a number of novel techniques for extracting and resolving user interface labels and addressing ambiguity in semantics, resulting in significant improvements over prior work. We apply UiRef to 50,162 Android applications from Google Play and use outlier analysis to triage applications with questionable input requests. We identify concerning developer practices, including insecure exposure of account passwords and non-consensual input disclosures to third parties. These findings demonstrate the importance of user-input semantics when protecting end users.

---

<sup>1</sup>Portions of this chapter appear in Andow et al. [And17]

## 3.1 Introduction

Mobile applications continue to consume an increasing amount of sensitive data to perform a variety of tasks, such as personal banking, health tracking, and online shopping. Vetting these applications to identify abnormal behaviors is paramount to ensure that privacy and security requirements of users are adequately satisfied.

Prior research [Ege11; Enc10; Arz14b; Gib12; TR14; Avd15] has demonstrated the utility of analyzing sensitive-data requests to identify security and privacy problems within applications, including insecure-data transmission, private-data leakage, and malware identification. Such approaches often rely on the semantics of the user data to enable automated analyses, e.g., for taint sources and runtime privacy notices. However, these approaches focus on *only* sensitive data originating from well-defined, system API calls, which explicitly represent the semantics of the data returned by the API invocation. Such focus addresses only a part of the challenge: applications also obtain a wide range of sensitive data, such as passwords and credit-card numbers, as input within their graphical user interfaces (GUIs). These data requests are largely ignored by prior analyses.

In this chapter, we focus on protecting sensitive data collected by third-party applications via their GUIs. Resolving the semantics of such data is challenging, as the API calls that retrieve a user input are generic and do not indicate the data’s semantics. For example, a user’s home address and credit-card number are retrieved by the same `getText()` API call if entered into `EditText` widgets. Instead, the semantics of a user input is denoted by natural-language text in the GUI, which prompt the user to enter specific types of data.

Semantics resolution of user inputs has been explored in two recent techniques: SUPOR [Hua15] and UIPicker [Nan15]. However, as we discuss in Section 3.3, both techniques have significant limitations that lead to inaccuracies. Further, both techniques fail to take into account the problem of *word ambiguity*, which is the coexistence of multiple meanings for a word or phrase. For example, the word “address” can refer to a postal address or an IP address depending on the context that the word is used, and hence can cause confusion for the existing techniques. In Section 3.5.3, we measure the prevalence of ambiguity during semantics resolution and show that 20.9% of input resolutions contain an ambiguous term. Although word ambiguity is a known and actively researched area in NLP, the limited amount of text within GUIs makes disambiguation especially challenging.

In this chapter, we propose UiRef, a User Input REsolution Framework that automatically

```

1 <LinearLayout width="match_parent" height="wrap_content" orientation="vertical">
2   <LinearLayout width="match_parent" height="wrap_content">
3     <TextView width="wrap_content" height="wrap_content" text="First Name"/>
4     <EditText width="match_parent" height="wrap_content" id="@+id/first"/>
5   </LinearLayout>
6   <LinearLayout width="match_parent" height="wrap_content">
7     <TextView width="wrap_content" height="wrap_content" text="Last Name"/>
8     <EditText width="match_parent" height="wrap_content" id="@+id/last"/>
9   </LinearLayout>
10  <TextView width="wrap_content" height="wrap_content" text="Address"/>
11  <EditText width="match_parent" height="wrap_content" id="@+id/address"/>
12 </LinearLayout>

```

**Figure 3.1** Example Android layout and corresponding XML specification with stripped namespaces

First Name \_\_\_\_\_

Last Name \_\_\_\_\_

Address \_\_\_\_\_

**Figure 3.2** Screenshot of the Android layout in Figure 3.1

resolves the semantics of user-input widgets by analyzing the GUIs of Android applications. UiRef advances the state of the art using novel techniques in its three main stages: layout extraction, label resolution, and semantics resolution. First, UiRef’s layout extraction provides a new hybrid analysis that accurately renders custom widgets. Prior techniques do not fully support custom widgets, which are used by 48.7% of the applications under our study. Second, UiRef’s label resolution models patterns within the spatial arrangement of widgets, achieving a 20.8% improvement over prior work [Hua15]. Finally, UiRef’s semantics resolution disambiguates words within input labels. Existing word-disambiguation techniques [Les86] cannot be applied directly, as GUIs rarely display text in full sentences. Instead, UiRef learns different word senses based on text that appears in other widgets within layouts.

We use UiRef to perform a large-scale analysis of 50,162 applications from Google Play. UiRef resolves the user-input semantics for each application. We then use outlier detection to identify questionable, and potentially malicious, input requests. The goal of our analysis is to provide an understanding of what sensitive information applications are asking for and

broadly identify various questionable practices in collecting user inputs that can potentially compromise users' security and privacy. Note that for this work, we do not differentiate a questionable practice from malicious intent. Such differentiation can be done by further analysis using existing techniques [Ege11; Enc10; Arz14b; Gib12; TR14; Avd15]. Rather, UiRef can filter the application dataset for focused deeper inspection.

Our analysis leads to three main security and privacy findings (see Section 3.6). First, we find that applications request a wide range of sensitive data, including SSNs, passport numbers, and healthcare information. While legitimate requests for this data may exist, we find that many such requests do not align with the purpose of the requesting application. Second, we identify 66 applications that directly request the user's third-party account passwords (e.g., Gmail). These requests expose the user's third-party accounts to compromise, as well as violating the primary tenet behind OAuth-like solutions. Finally, we identify 6 popular applications that send sensitive-input data to advertisers without disclosing the behavior.

During our analysis, we also identify application design flaws that may result in loss of privacy. First, applications allow third-party libraries to directly request sensitive inputs from users, likely resulting in user confusion. In other words, the user cannot identify whether the library or the application is requesting data. This flaw can be exploited by malicious libraries to compromise sensitive data. Second, applications display untrusted third-party components in the same GUIs as sensitive-input requests, leaving the applications vulnerable to private-data theft due to layout-traversal attacks [RK13].

In summary, this chapter makes the following main contributions:

- We develop novel techniques for semantics analysis of GUI inputs to considerably improve over the current state of the art (5.5%-25.6% more accurate at extracting layouts and 20.8% more accurate at resolving labels). We perform a direct comparison with prior work by implementing a representative solution in SUPOR [Hua15].
- We propose a novel approach to address ambiguity of descriptive text in mobile applications. Ambiguity is a key challenge unaddressed by prior work, and it is particularly challenging for GUIs due to limited text. We demonstrate that word embeddings can effectively perform this task.
- We demonstrate the utility of UiRef in performing security and privacy analysis through a large-scale study on 50,162 Android applications. Our study highlights various forms of questionable, and potentially malicious, practices by developers and

emphasizes the need to consider user inputs in the security and privacy analysis of mobile apps.

The remainder of this chapter describes the design and implementation of UiRef, followed by its application towards security and privacy threats in user inputs.

## 3.2 Background

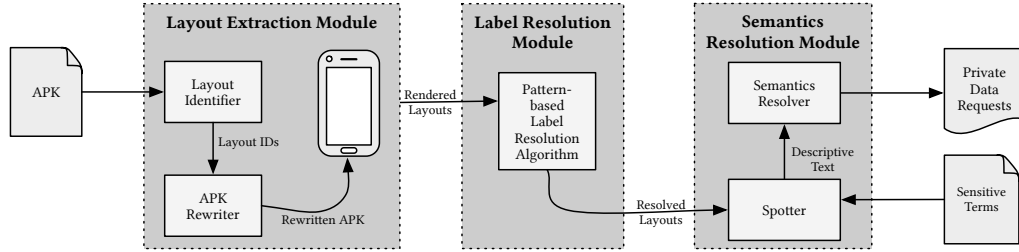
When using Android applications, users interact with layouts (i.e., GUIs) to enter inputs and perform actions. Layouts consist of widgets, such as push buttons, text fields, and check boxes. These widgets are arranged and grouped within layouts through the use of *view groups*, which are containers for holding other *views* (i.e., view groups or widgets). Due to view groups, layouts are structured as hierarchical trees named as *view hierarchies*.

Layouts are typically defined by the developer at compile time using XML. Figure 3.1 shows a simplified version of a layout’s XML file (i.e., *main\_layout.xml*) and Figure 3.2 shows the corresponding screenshot. To render a layout, applications pass the resource identifier of the layout’s XML file to the `setContentView()` or `LayoutInflater.inflate()` methods. To access a resource, developers use the Java R class generated during compilation, while the compiled application uses integer constants to internally refer to resources. For example, to render the layout in Figure 3.1, an app invokes `setContentView(R.id.main_layout)`. The `public.xml` file in the application package (APK) provides a mapping of the integer constants to the strings used in the XML layout.

Android’s SDK provides a wide range of pre-defined view groups and widgets. However, developers may also define custom views in their Java code by extending these pre-defined classes, which can then be referenced in the layout’s XML files. The implementation of these custom views may customize rendering and also dynamically insert view groups and widgets within its view hierarchy.

## 3.3 Problem and Challenges

Unlike information retrieved from Android platform APIs (e.g., GPS), the semantics of text inputs is often poorly defined. As such, it received limited investigation by prior research [Hua15; Nan15] in contrast to the wealth of literature studying the abuse of privacy-sensitive platform APIs [Enc10; Arz14b; Gib12; TR14].



**Figure 3.3** UiRef Architecture

**Problem Statement:** *This work seeks to understand what information Android apps are requesting through their GUIs by automatically resolving the semantics of the user inputs and to further analyze their potential security and privacy repercussions.*

Semantics resolution is the first step in ensuring proper handling of security and privacy sensitive input. There are many ways in which the resulting meta information can be used to enhance security analysis (e.g., semantics for taint sources). In this chapter, we use the resolved semantics to achieve two main goals. First, we use it to understand the general landscape of the types of security and privacy sensitive information that applications are requesting from the user. Second, we leverage it to identify questionable, and potentially malicious, practices used by apps by performing outlier analysis. This analysis identifies applications that request data uncommon for the its category (e.g., a game asking for an SSN) and allows us to triage apps for manual analysis.

Our work is not the first to consider the semantics of user inputs in Android applications. Prior techniques (SUPOR [Hua15], UIPicker [Nan15], and AppsPlayground [Ras13]) have significant limitations that impact the accurate and unambiguous resolution of the semantics of user inputs. We identify three areas of challenges: layout extraction, descriptive-text (label) resolution, and semantics resolution.

**Layout Extraction:** The spatial arrangement of widgets influences the layout’s semantics. While the parent-child and sibling relationships of a layout’s view hierarchy influence the spatial arrangement, it is a poor approximation of proximity due to the flexibility allowed by widgets. Further, code defining custom views may control the types of widgets displayed as well as their position in the layout. Around 48.7% of apps in our study (Section 3.6) use custom views.

UIPicker operates directly on the XML specification of layouts; therefore, it cannot accurately interpret the spatial arrangement of widgets and their proximities to one another.

In contrast, SUPOR extracts layouts by modifying the static rendering engine of the Android Developer Tool (ADT), which provides this positioning. However, neither UIPicker nor SUPOR sufficiently handles custom views. The UIPicker paper does not mention custom views. The static rendering engine of ADT used by SUPOR cannot execute the bytecode of custom views. Therefore, SUPOR renders custom views as the nearest superclass available in the Android framework, causing inaccurate and incomplete layout rendering. Finally, AppsPlayground can handle custom views, but it must dynamically navigate the GUI to reach layouts.

**Descriptive-Text Resolution:** To prompt the user for inputs, developers either use text widgets with a label in the nearby proximity to the input fields (Figure 3.1), or embed descriptive text as attributes on the input widgets (e.g., the hint and text attributes on the EditText widget). While embedded descriptive text is straightforward to resolve, identifying the correct label for an input widget is nontrivial. Even if embedded descriptive text exists for an input widget, the corresponding label widget must be resolved. Some applications use labels to denote the type of information and embedded descriptive text to provide instructions to the user (e.g., “required”).

A general approach to match an input widget with a label widget is to define a distance metric. Both UIPicker and AppsPlayground use sibling relationships in layouts to resolve the associated descriptive text. However, in practice, sibling relationships do not accurately gauge proximity. In contrast, SUPOR performs label resolution by partitioning the space surrounding an input widget into nine areas, calculating a position-based weighted average for each pixel in the input widget to the nearest pixels in labels, and mapping each input widget to the label widget with the lowest weighted average. However, SUPOR suffers from multiple limitations that are sources for incorrect label resolutions as shown in Section 3.5. First, it always resolves labels to an input widget even if the label is *too far away* from the input widget. Second, it depends on the input widget’s area size and predefined weights that bias it towards labels to the left and above, thus affecting the algorithm’s generality.

**Semantics Resolution:** Mobile applications use short text phrases in descriptive text, making semantics resolution of user inputs challenging. Word polysemy limits the use of simple key-phrase matching techniques, and affects 20.9% of input-field resolutions (Section 3.5.3). Therefore, disambiguation is required.

None of the prior techniques resolve the ambiguity of words. SUPOR, UIPicker, and AppsPlayground use key-phrase matching on the input widget’s associated descriptive text,

but cannot differentiate the multiple semantics of that key phrase. SUPOR acknowledges this limitation, and excludes the word “address” from their key-phrase list. Additionally, UIPicker leverages developer-defined variable names and input-widget type attributes (e.g., password) in their resolution. However, such developer-defined names are not tolerant to name-based obfuscation or poor coding practices.

## **3.4 System Approach and Design**

The UiRef resolves the data semantics of user-input widgets by analyzing Android layouts. Figure 3.3 shows UiRef’s architecture. The layout-extraction module instruments APKs to force the rendering of their layouts and exports the rendered layouts for further analysis. The label-resolution module identifies patterns within the placement and orientation of labels with respect to input widgets in the layout and geometrically resolves labels. The semantics-resolution module applies text-analytics techniques on the input widget’s associated descriptive text to determine the expected type of information accepted by the input widgets.

### **3.4.1 Layout Extraction**

The goals of the layout-extraction module are three-fold: (1) identify all layouts in an application; (2) identify the spatial relationships between UI widgets for each layout; and (3) identify the descriptive text displayed to users. The spatial relationships are needed to map label widgets to input widgets, as discussed in Section 3.4.2.

UiRef’s layout-extraction module uses a hybrid technique to extract rendered layouts. UiRef’s technique uses static analysis to identify the layouts used by the application, and dynamic on-device rendering to extract each rendered layout that users eventually interact with. Since on-device rendering allows for the execution of developer code, it allows for the correct rendering of custom views.

UiRef disassembles an APK using ApkTool [Apk], and collects the resource identifiers for each layout present in the APK’s public.xml file. It saves the layout resource identifiers to the application’s assets folder to be used later during on-device rendering. Subsequently, it injects a custom activity into the APK, and rewrites the application’s manifest file to register the injected activity as an entry point. UiRef then reassembles the APK, and sideloads it onto a live device.

When the injected activity is executed, it iterates through the resource identifiers saved in the assets folder, and invokes the `setContentView()` method for each identifier to cause the application to render the layout. It then iterates through the rendered layout's view hierarchy to extract associated metadata, such as the coordinates of each view, visibility attributes, and text strings. This information along with the view hierarchy is exported as XML.

As discussed in Section 3.4.2, UiRef's label-resolution module requires a list of potential labels and a list of input widgets. To construct these lists, UiRef uses class-inheritance information to identify the types of widgets that commonly accept user inputs (i.e., `EditText`, `AbsSpinner`, `CheckedTextView`, `RadioButton`, `CheckBox`, `ToggleButton`, `Switch`, `SwitchCompat`, `RatingBar`), and widgets that display static text (i.e., `TextView`). For example, an `EditText`, or widgets that extend `EditText`, are likely to accept user inputs. Note that this technique differs from SUPOR's use of inheritance information, as SUPOR uses it during the rendering process to determine how to render custom views. In contrast, UiRef does not rely on inheritance information during rendering, as UiRef can render custom views. UiRef extracts inheritance information during layout extraction to avoid performing the Class Hierarchy Analysis offline.

To determine whether a widget accepts a user input, UiRef resolves the closest ancestor in a widget object's class hierarchy that is in Android's framework, and includes the nearest SDK class in the metadata output. For example, if a class `CustomTextView1` extends `android.view.Textview`, UiRef would mark its ancestor as `TextView`. Similarly, if `CustomTextView2` extends `CustomTextView1`, UiRef would also mark its ancestor as `TextView`, because `CustomTextView2` extends `CustomTextView1`, which extends `TextView`. We identify 14 base widget types from Android's documentation, and use Java's `instanceof` operator to find the view object's nearest SDK class.

Note that UiRef executes only the code needed to render layouts, and hence does not suffer from code-coverage limitations. This technique does have two limitations. First, UiRef is limited to statically defined layouts. However, statically defining layouts is a common practice, as it typically requires less effort than dynamic generation. Second, UiRef cannot extract dynamically generated text (e.g., from network connections). However, since we focus on text labels for user inputs and not display content, we believe it is reasonable to assume that labels statically define the text.

---

**Algorithm 1** Resolution of Label to Input Widget

---

```
1: procedure RESOLVELABELS(uifs, labels)
2:   resolved ← init empty list
3:   do
4:     pairs ← GenCandidateSets(uifs, labels)
5:     labels.remove(pairs.labels)
6:     uifs.remove(pairs.uifs)
7:     resolved.append(pairs)
8:   while size(pairs) > 0
9:   return resolved
```

---

### 3.4.2 Label Resolution

The goal of UiRef’s label-resolution module is to identify the label associated with each user-input widget. It operates on the intuition that developers are consistent with the physical arrangement and orientation of labels to user-input widgets. For example, if a developer positions labels to the left of an input widget, then it is expected that other labels in the layout will also be positioned on the left. Therefore, the label-resolution module works by identifying patterns within the placement of labels relative to input widgets.

UiRef’s algorithm for label resolution (Algorithm 1) is iterative, so it correctly resolves labels even if inconsistencies exist. As an example, the “Address” label in the layout of Figure 3.1 is inconsistent as it is placed above the input widget while other labels are placed to the left of the input widgets, and is still successfully resolved by the algorithm. During the first iteration, UiRef resolves the label with the text “First Name” to the EditText with the identifier *@+id/first*, and “Last Name” to the EditText with the identifier *@+id/last*. During the second iteration, UiRef resolves the label with the text “Address” to the EditText with the identifier *@+id/address*.

The label-resolution module accepts the rendered layouts from the layout-extraction module as input, and resolves labels associated with user-input widgets.

**Label-Resolution Algorithm:** UiRef maps labels to input widgets by identifying patterns in their relative placements. Algorithm 1 describes the algorithm. The input is the sets of input widgets (*uifs*) and potential labels (*labels*) identified using class-inheritance information within the layout-extraction module (Section 3.4.1).

Algorithm 1 starts (Line 4) by invoking GenCandidateSets (Algorithm 2) to generate a list of candidate sets of label and input-widget pairs. For each input widget, GenCandidateSets

---

**Algorithm 2** Create Candidate Map

---

```
1: procedure GENCANDIDATESETS(uifs, labels)
2:   candidates  $\leftarrow$  init empty candidate map object
3:   for each i  $\in$  uifs do
4:     for each l  $\in$  labels do
5:       v  $\leftarrow$  calcSmallestVector(i, l)
6:       if v.distance > threshold then
7:         continue
8:       candidates[v].appendNE(i, l)
9:   return GetOptimalSet(candidates)
```

---

creates a set of vectors from the input widget to all potential labels in the layout (Lines 3–8, Algorithm 2). The vectors represent the Euclidean distance (i.e., magnitude) and direction (i.e., angle) between the input widget and label. *calcSmallestVector* (not shown) creates up to three vectors for each input-widget and label pair. Two vectors go from the two closest corners of the input widget to the corresponding corners of the label. The third vector is created if a label is directly above, below or to the sides of the input widget, being anchored at the closest point between the input widget and label.

If a vector’s magnitude is greater than a predetermined threshold, it is not considered as a candidate (Lines 6–7, Algorithm 2). The algorithm then appends the input-widget and label pair to the candidate set under the entry for the corresponding vector if either of the widgets does not already exist (Line 8, Algorithm 2). During implementation, we empirically determine the threshold by taking the average distances of labels to input widgets for 100 randomly sampled applications. The distance threshold can be made independent from the device’s screen size by representing the threshold as a proportion of the screen size. Note that the apps in the validation set were excluded from the datasets used in our evaluation.

After the candidate sets are constructed, *GetOptimalSet* (Algorithm 3) is invoked (Line 9, Algorithm 2) to extract the optimal label to input-widget mapping. The algorithm first retrieves the candidate sets with the largest set size (Line 2, Algorithm 3). If there are multiple such sets, the algorithm prioritizes sets whose labels are either above or to the left of input widgets (i.e., the vector direction is 90 degrees or 180 degrees, respectfully) (Lines 5–8, Algorithm 3). If multiple sets are of equal size and both vector directions are to the left or top of the input widgets, then *GetOptimalSet* selects the set with the smallest vector distance (Lines 9–10, Algorithm 3). The algorithm returns the optimal label to input-widget

mapping.

Once Algorithm 1 receives the optimal mapping of label to input-widget pair, it removes the resolved pairs from the initial lists, and appends the resolved pairs to the resolved set (Lines 5–7, Algorithm 1). This process is repeated until no new labels are resolved.

### 3.4.3 Semantics Resolution of Input Widgets

The goal of semantics resolution is to resolve the types of input data prompted by the associated descriptive text. These types of data correspond to *concepts* in our terminology, which are expressed by terms (i.e., single word or phrase) in layouts. However, the mapping of terms to concepts is not necessarily disjoint. Different terms may represent the same concept (synonymy), and the same term may represent multiple concepts (polysemy). Due to polysemy, strategies of keyword-based resolution cannot be used to resolve concepts.

The semantics-resolution module uses the context surrounding a term to resolve its concept. The module has two main tasks: (1) terminology extraction, and (2) concept resolution.

#### 3.4.3.1 Terminology Extraction

Terminology extraction is used to determine candidate terms that represent concepts in the domain. UiRef requires terminology of security and privacy sensitive concepts that applications use to prompt for inputs. Terms can be a single word (uniterm) or a phrase (multiterm), such as *gender* and *date of birth*. UiRef uses a data-driven technique for defining a list of security and privacy sensitive terms. The list is derived from the text displayed within layouts from the dataset in Section 3.6.

Our terminology-extraction process uses all text displayed in the layouts extracted by the layout-extraction module. We begin by using regular expressions to replace email addresses, URLs, and common phone-number formats (e.g., a@b.com, http://www.b.com, 123-456-7890) by their respective terms. For example, a@b.com is replaced by email\_addr\_example. We also substitute the # symbol with the word “number,” and use regular expressions to remove prompt text, such as “enter your.” We then lemmatize the text, being a common preprocessing step to normalize text. For example, lemmatizing the word “ethnicities” would change it to “ethnicity.”

After preprocessing the text, we mark text that contains only a single word as a potential uniterm. To heuristically remove noise and reduce the manual post-processing efforts

---

**Algorithm 3** Find Optimal Mapping

---

```
1: procedure GETOPTIMALSET(candidates)
2:   maxSets ← GetLargestSets(candidates)
3:   opt ← maxSets[0]
4:   for each s ∈ maxSets[1 :] do
5:     optLorT ← IsLeftOrTop(opt.vec)
6:     sLorT ← IsLeftOrTop(s.vec)
7:     if !optLorT & sLorT then
8:       opt ← s
9:     else if optLorT == sLorT AND opt.vec.dist > s.vec.dist then
10:      opt ← s
11:  return opt
```

---

described below, we remove uniterms that appear only within a single layout. Note that removing uniterms may cause us to miss a few important terms; however, missed terms can be added manually.

Next, we create a candidate set of sensitive multiterms by extracting n-grams of sizes 2-4 for each piece of text. Note that this process uses a sliding window across the words in the text to find the most appropriate groupings of words.

Once the set of n-grams is created, we heuristically refine the set as follows. First, we remove n-grams that appear in less than two separate layouts. Second, we remove n-grams that are substrings of larger n-grams and appear in only the same layouts. For example, we remove the bigram “social security” if the only time that it appears is in the longer term “social security number.” Third, we remove n-grams that start or end in stopwords (e.g., “a,” “the,” “and”). Fourth, we remove n-grams with common verbs occurring in middle positions of n-grams (e.g., “could,” “have,” “would”). Note that the goal of multiterm extraction is not to extract textual prompts, but rather to extract terms that correspond to concepts. For example, we seek to extract the term first name from the following text, “What is your first name?” Finally, if a matching unigram is found when spaces are removed from n-grams, we mark the unigram as a potential synonym of the n-gram (e.g., user name and username).

Once the candidate lists are created, we perform two final manual post-processing steps. First, we read through the lists and filter out n-grams representing verb phrases not caught with our filters, and phrases that do not clearly represent concepts. Second, we scan through the lists of n-grams and uniterms, and mark down potentially sensitive terms along

with their synonyms. For example, in this step, we mark “last name” as a sensitive term, and “surname” as its synonym. We also create a list of potentially ambiguous terms (e.g., name, address, number), which is used later during semantics resolution to determine which terms require disambiguation.

### 3.4.3.2 Concept Resolution

The goal of concept resolution is to determine the semantics of an input widget. For example, UiRef aims to link the concepts *first name*, *last name*, and *postal address* to the corresponding input widgets in Figure 3.1. Recall from Section 3.3 that key-phrase matching alone is not sufficient due to polysemy. Before we can disambiguate terms, we must determine the different meanings (i.e., senses) in which a term appears. Note that an automated technique is required, as all possible senses of the term must be considered when performing disambiguation.

The process of determining these different meanings is known as word-sense induction. The process of resolving the meaning of a specific instance of a term is known as word-sense disambiguation. UiRef performs these two tasks using the Adaptive SkipGram (AdaGram) model [Bar15]. AdaGram extends Mikolov’s SkipGram model [Mik13] by using a non-parametric Bayesian approach over Dirichlet processes to learn multiple word vectors per word. For a single word, a word vector represents a *sense* in which the word appears.

**Word-Sense Induction:** Training an AdaGram model to perform word-sense induction requires flat text documents. To flatten text within layouts, UiRef generates a string from the text within layouts, scanning from the layout’s top-left corner to the bottom-right corner. For each piece of text, UiRef preprocesses the text by performing lemmatization, stripping stopwords, and transforming the text into lowercase. For input widgets with both labels and hints, UiRef outputs the label text before the hint. For example, Figure 3.1 produces the following document: “first name last name address.”

To allow for the disambiguation of multiterms, UiRef generates the second document for the same layout with multiterms treated as one lexical unit. UiRef collapses adjacent words that form multiterms by replacing spaces with underscores (e.g., “first name” becomes “first\_name”) using the list of multiterms from the previous step. For example, Figure 3.1 also produces the following document: “first\_name last\_name address.”

After these two documents are created for each layout, UiRef trains an AdaGram model, which is used to perform word-sense disambiguation. Note that UiRef trains the AdaGram

model using a maximum of 25 potential word senses being learned per word. This number is chosen through experimental exploration to avoid underestimating the number of word senses, as doing so would cause concepts to overlap in the same word sense.

Once the model is trained, we manually resolve the concept to which the word vector refers for each potentially ambiguous word by analyzing the terms that have the closest relation to the word vector. For example, the closest related words for one meaning discovered for “address” are *city, street, first, zip, postal*. Therefore, we mark the semantics for that meaning of “address” as a *postal address*. The closest related words for another meaning of “address” are *port, host username, ip, and pswd*, which we mark as *IP address*.

**Semantics Resolution and Disambiguation:** To resolve the semantics of each input widget, UiRef first lemmatizes the hint text, combines multiterms into one lexical unit, and removes stopwords. Next, UiRef uses the *spotter* to scan the hint’s preprocessed text to search for the sensitive terms constructed in Section 3.4.3.1. If the spotter locates a sensitive term that is unambiguous, UiRef marks the input widget with the term’s associated concept. However, if the spotter finds a potentially ambiguous term, UiRef uses the trained AdaGram model to disambiguate the term using the surrounding context with a window size of 5 (i.e., 5 words before and 5 words after). To disambiguate a word, UiRef predicts the probability of the target term given the context, and returns the concept with the highest probability. If the hint text does not contain any sensitive terms, UiRef repeats this process with the associated label’s text.

For example, UiRef resolves the input widget with the widget identifier *@+id/address* in Figure 3.1 as follows. Since the widget does not contain embedded text (e.g., hint or text attributes), UiRef begins by analyzing the text of the label that it resolves for this widget (i.e., “address”). UiRef’s spotter finds the word “address” in the text and tags it as ambiguous. UiRef disambiguates address by extracting the surrounding context (*first\_name last\_name*), and using the model to predict the meaning of the word. UiRef predicts that “address” refers to the sense that corresponds to a postal address.

### 3.4.4 Outlier Analysis

The goal of outlier analysis is to identify abnormal input requests that do not align with the purpose and functionality of the application. Since different types of applications vary in terms of the information that they request, comparing sensitive input requests across a broad range of applications is not sufficient to identify outliers. For example, a financial

**Table 3.1** Semantic Bucket Excerpt (5/78)

Semantic Bucket	Sensitive Terms
username_or_email_addr	email address, email adress, email id, emailid, gmail address, screenname, username, ...
credit_card_info	credit card number, card number, card code, cvv code, cvv, cvc, credit card expiration, ...
person_name	first name, middle name, last name, full name, middle initial, legal name, credit card holder, ...
phone_number	phone number, telephone number, mobile phone, cell, work phone, home phone, fax number, ...
location_info	city, town, state, zip, zip code, post code, street address, ship address, billing address, ...

application may be expected to request the user’s income while a game would not be expected to do so. Therefore, we perform outlier analysis at the granularity of Google Play’s 42 application categories. Note that we combine all of the 18 subcategories of games into a single “game” category, which results in 25 categories of applications.

To detect outliers for each Google Play category, we perform two tasks. First, we manually collapse sensitive concepts into 75 semantic buckets. For example, first name and last name are grouped in a *person\_name* bucket. Table 3.1 shows 5 of the 75 semantic buckets along with an excerpt of the associated terms. Second, we apply an unsupervised technique, called the Ranking-based Outlier Analysis and Detection (ROAD) algorithm [Sur12], to generate a ranked list based on the likelihood that a record is an outlier.

The ROAD algorithm was chosen over other outlier detection approaches multiple reasons. First, our dataset consists of categorical variables. The ROAD algorithm is based on *k*-modes with the dissimilarity function proposed by Ng et al. [Ng07], which is a natural choice due to the use of the data’s *mode* rather than its *mean*. In contrast, other popular approaches, such as One-class SVM and *k*-means, were designed for continuous variables. Second, we chose the ROAD algorithm instead of other common outlier detection approaches designed for categorical data, such as attribute value frequency (AVF), as the ROAD algorithm produces a multi-dimensional outlier ranking based on the dissimilarity of the data records from the largest clusters and the density of those clusters rather than the occurrence frequency of the attributes.

To detect outliers, the ROAD algorithm operates in two phases. First, *k*-modes is applied to cluster the categorical data records to discover common trends (i.e., the largest-sized clusters). We choose the value of *k* by iteratively running *k*-modes to maximize the silhouette score, which is a metric that measures the similarity of a record to its own cluster compared to records in neighboring clusters.

The distance function, as shown in Equation 3.1, is a modified version of the dissimilarity function proposed by Ng et al. [Ng07]. The distance function is defined as  $d(Z_j, X_i)$ , where

$Z_j$  is the mode of the  $j^{th}$  cluster,  $C_j$ , and  $X_i$  is the categorical data record. The distance between  $Z_j$  and  $X_i$  is the summation of the dissimilarity scores for each of the  $m$  attributes in the categorical data records. The dissimilarity score for the  $r^{th}$  attribute is 1 if the value of  $z_{j,r}$  does not equal the attribute value of  $x_{i,r}$ , and  $1 - \frac{|C_{j,r}|}{|C_j|}$  otherwise, where  $|C_{j,r}|$  is the number of records in cluster  $C_j$  with the attribute value  $x_{i,r}$ , and  $|C_j|$  is the number of records in cluster  $C_j$ .

Note that we modify the distance function by multiplying the dissimilarity score by a probability-based value weighting function. We modify the weighting functions so that rare attribute values have a larger influence on the distance calculation. For example, if most applications do not request the user's social security number (SSN), it is more significant that an application requests the user's SSN than not requesting the user's SSN. Therefore, the attribute value of requesting the user's SSN (i.e., rare values) holds more weight in the distance function than the attribute value of not requesting the user's SSN (i.e., frequent values). The weighting function,  $weight(x_{i,r})$ , is the reciprocal of the square root of the probability that the value of  $x_{i,r}$  occurs in the categorical dataset. The probability,  $P(x_{i,r})$ , is the frequency of the value of  $x_{i,r}$  divided by the total number of categorical data records. The weighting function returns a higher weight for rare occurrences of attribute values.

$$d(Z_j, X_i) = \sum_{r=1}^m \left( \delta(z_{j,r}, x_{i,r}) * weight(x_{i,r}) \right) \quad (3.1)$$

where

$$\delta(z_{j,r}, x_{i,r}) = \begin{cases} 1, & \text{if } z_{j,r} \neq x_{i,r} \\ 1 - \frac{|C_{j,r}|}{|C_j|}, & \text{if } z_{j,r} = x_{i,r} \end{cases}$$

and

$$weight(x_{i,r}) = \frac{1}{\sqrt{P(x_{i,r})}}$$

Second, the ROAD algorithm uses the resulting clusters to rank records by their distance from the largest-sized clusters and the density score of the record. A large cluster is defined as a cluster that has more than  $\alpha\%$  of the data records in the cluster. In our implementation of the ROAD algorithm, we initially set  $\alpha$  to the experimentally determined value of 20% for each category, and decrement  $\alpha$  by a half a percent until a large cluster is found. Note that we chose to start at 20%, as it ensured that smaller sized clusters were not accidentally included when choosing the large cluster. Further, we chose to decrement by a half percent until a

large cluster is found, because it also ensured that only the largest clusters are chosen. To calculate the distance of each data record from the large clusters, we use the same distance function in Equation 3.1. The density score is simply the summation of the probability that the value  $x_{i,r}$  occurs divided by the number of attributes. The records are sorted in descending order based on the cluster distances, and in ascending order by the density score when the cluster distances between the two records are equal. A higher distance from the largest cluster denotes and a lower density score denotes that the record is more-likely an outlier.

## 3.5 Evaluation

In this section, we evaluate the effectiveness of UiRef with respect to its major modules. To evaluate UiRef’s layout extraction, we use a combination of emulators and real devices. We use 12 x86 Android emulators and a single Nexus 4, both running Android 5.1.1. We run applications that contain ARM-based native libraries on the real device, as the x86 emulators do not include the translation library.

Our dataset is based on the October 31, 2014 PlayDrone [Vie14] snapshot (1.4 million applications). We perform proportionate stratified random sampling across the dataset to ensure a representative sample by using the Google Play categories and number of downloads to form our strata. For verification purposes, we use Python’s *langdetect* module to ensure the dataset contains only applications with English descriptions. Our final dataset consists of 50,162 apps.

We use SUPOR [Hua15] as a representative baseline for comparison. Since SUPOR’s source or binary code was not available, we reimplement their approach. When specific implementation details are not clear or ambiguous, we contact the authors for clarifications. We do not compare our results to UIPicker due to the limitations discussed in Section 3.3, and the unavailability of their trained classifiers.

### 3.5.1 Layout-Extraction Performance

To evaluate the effectiveness of UiRef’s layout-extraction technique, we estimate its improvement over SUPOR’s technique for rendering static layouts. As discussed in Section 3.3, one improvement that UiRef provides over SUPOR is the rendering of custom views. To evaluate the improvement for handling custom views, we estimate a lower bound and up-

**Table 3.2** Performance Evaluation Results

	Label Resolution		Semantics Resolution		Disambig.
	UiRef	SUPOR*	UiRef	SUPOR*	UiRef
<b>Acc.</b>	84.0%	63.2%	95.0%	90.2%	82.1%
<b>Raw</b>	630/750	474/750	708/745	672/745	275/335

\* UiRef’s Layout Extraction, and our reimplementaion of SUPOR

per bound: (1) lower bound: the number of layouts and applications that include custom views that programmatically add other views; and (2) upper bound: the number of layouts and applications that include custom views. Note that we do not evaluate other potential limitations of SUPOR’s layout extraction, e.g., the accuracy of ADT’s rendering.

Our dataset consists of the 50,162 applications described earlier. To identify custom views that add other views programmatically, we disassemble the APKs using ApkTool and perform class-hierarchy analysis to identify classes whose inheritance hierarchy contains the View class or any other SDK class that extends the View class (e.g., EditText, LinearLayout). Next, we perform a signature-based search on the custom view’s underlying code for method invocations used to add views, such as ViewGroup→addView(View child). Since our goal is to provide a conservative estimation rather than an exhaustive analysis, we choose a signature-based technique over reachability analysis to reduce computational overhead.

**Results:** In total, around 25.6% of the static layouts (479,337/1,873,737) and 48.7% of the applications (24,436/50,162) contain at least one custom view. Further, 35.1% of applications in the dataset contain at least one custom view whose underlying code dynamically adds views (17,597/50,162), which impacts around 5.5% of static layouts (102,671/1,873,737). Therefore, UiRef provides between a 5.5% to 25.6% improvement over SUPOR for extracting layouts with an average extraction time of 53.7 milliseconds per layout.

### 3.5.2 Label-Resolution Performance

To evaluate UiRef’s label-resolution technique, we manually annotate the label that corresponds to each input widget. We compare our manual annotations with the results from both UiRef and SUPOR’s label resolutions. Note that we use UiRef’s layout-extraction technique when reporting SUPOR’s label-resolution results to allow for a direct comparison between the algorithms, and to reduce potential errors carried over from SUPOR’s layout extraction.

**Table 3.3** Disambiguation Evaluation Results

Ambiguous Term (≤25 widgets)	Example Concepts	Prevalence of Ambiguity During Semantics Resolution (175,101 widgets)			Disambiguation Performance (335 widgets)	
		Requests	Layouts	Apps	Correct	Incorrect
account_number	bank account, utility account	292 (0.2%)	274 (3.8%)	181 (1.2%)	20	3
address	IP address, postal address	3,228 (1.8%)	2,905 (4.1%)	2,310 (14.8%)	22	2
age	person age, product age	334 (0.2%)	297 (0.4%)	198 (1.3%)	21	3
cc	carbon copy, credit card	61 (0.03%)	55 (0.1%)	48 (0.3%)	3	0
cm	person height, product height	1,599 (0.9%)	1,569 (2.1%)	1,553 (9.9%)	4	2
day	birth day, current day	7,297 (4.2%)	5,502 (7.7%)	2,095 (13.4%)	1	5
destination	physical location, file location	165 (0.1%)	158 (0.2%)	141 (0.9%)	16	7
first	first name, first place	325 (0.1%)	307 (0.4%)	254 (1.6%)	22	3
ft	person height, product height	1,537 (0.9%)	1,530 (2.1%)	1,521 (9.7%)	1	2
height	person height, product height	246 (0.1%)	233 (0.3%)	148 (0.9%)	19	2
last	last name, last place	201 (0.1%)	176 (0.2%)	144 (0.9%)	20	2
lb	person weight, product weight	3,056 (1.7%)	1,548 (2.2%)	1,533 (9.8%)	2	4
location	physical location, file location	4,577 (2.6%)	4,545 (6.4%)	2,653 (17.0%)	23	2
month	birth month, expiration month	386 (0.2%)	313 (0.4%)	235 (1.5%)	3	6
name	person name, file name	9,753 (5.6%)	9,497 (13.3%)	7,403 (47.3%)	19	5
number	phone number, card number	1,641 (0.9%)	1,330 (1.9%)	995 (6.4%)	20	3
security_code	CCV code, verification code	145 (0.1%)	136 (0.2%)	109 (0.7%)	22	2
weight	person weight, product weight	285 (0.2%)	273 (0.4%)	181 (1.2%)	19	3
year	birth year, expiration year	1,472 (0.8%)	1,394 (1.9%)	1,174 (7.5%)	18	4
<b>Total</b>		36,600 / 175,101 (20.9%)	25,720 / 71,291 (36.1%)	10,003 / 15,642 (63.9%)	275/335 (82.1%)	60 / 335 (17.9%)

We randomly select 12 applications for each of Google Play’s 42 categories from the 50k dataset. We run UiRef on the 504 applications to extract layouts. Note that SUPOR handles only EditText widgets. Therefore, we remove layouts that do not contain at least one EditText widget to provide a fair comparison to SUPOR (although UiRef resolves 9 base input widget types), reducing our dataset to 280 applications. We then remove layouts whose screenshot fails or does not capture the entire layout, layouts that are duplicates of another layout, contain non-English text, are a fragment such as a search bar, or do not contain any descriptive text or icon. Our final dataset consists of 349 layouts (109 applications) with 750 input widgets where 420 widgets have associated labels, and 472 are manually tagged as containing sensitive data.

**Results:** Table 3.2 shows that UiRef provides a significant 20.8% improvement in accuracy over SUPOR when resolving labels. In total, UiRef correctly resolves the labels for 630/750 input widgets (84.0%) with an average runtime of 16 milliseconds per layout. The majority of UiRef’s incorrect label resolutions are due to applications using TextViews to display non-modifiable data alongside user-input widgets, causing UiRef’s pattern-based resolution algorithm to choose incorrect candidate sets. In future work, we plan to resolve this problem by applying program analysis to differentiate between labels and such TextViews. The rest

of the incorrect label resolutions are due to the multiple labels per input widget (e.g., measurement units), or multiple input widgets per label (tabular arrangements). SUPOR’s errors originate from the lack of maximum distance metric always resulting in a resolved label for an input widget, and from its preference for labels located to the left or above input widgets while the correct label is located below the input widget.

### 3.5.3 Semantics-Resolution Performance

We next evaluate (1) UiRef’s accuracy of resolving semantics; (2) the prevalence of ambiguity during semantics resolution; and (3) the effectiveness of UiRef’s disambiguation.

#### 3.5.3.1 Overall Performance

To evaluate UiRef’s semantics resolution, we manually annotate the 349 layouts in Section 3.5.2 with a semantics label for each input. We remove input widgets from our results whose ambiguity could not be resolved from viewing the screenshot alone (5 input widgets in total). We compare our manual annotations with the results produced by UiRef’s semantics resolution algorithm for input widgets, and SUPOR’s key-phrase matching using our term list. Note that SUPOR ignores the resolution of ambiguous terms, such as “name,” so our re-implementation of SUPOR also ignores ambiguous terms.

**Results:** Table 3.2 shows that UiRef achieves 95.0% accuracy when resolving the semantics of input widgets (4.8% increase in accuracy over SUPOR). UiRef’s incorrect resolutions are due to UiRef not having enough context to disambiguate the term (14/37), the spotter missing sensitive keywords (13/37), insufficient parsing of the text (5/37), incorrect label resolutions (4/37), and incorrectly marking a non-sensitive input request as sensitive (1/37). Although SUPOR has a 90.2% accuracy with this dataset, such result is an overapproximation due to dataset selection, as the dataset has a low number of input widgets with ambiguous terms, and the majority of input widgets with SUPOR’s incorrect label resolutions are resolved using the embedded text attributes (171/192). In Section 3.5.3.3, we measure the prevalence of ambiguity during semantics resolution.

#### 3.5.3.2 Prevalence of Ambiguity

To demonstrate the importance of disambiguation when resolving semantics of input widgets, we measure the impact of ambiguity on semantics resolution. We run UiRef’s

semantics resolution on the 50,162 applications described in Section 3.5 and output input requests where the descriptive text (i.e., hint, label, or text) used for semantics resolution contains one of the 19 ambiguous terms shown in Table 3.3. From the 50,162 applications, there are 175,101 input requests requiring semantics resolution across 71,291 layouts and 15,642 applications.

**Results:** Table 3.3 shows that the resolution of 20.9% (36,600/175,101) of input requests contain an ambiguous term within the descriptive text used when resolving semantics. Further, ambiguity affects the semantics resolution of 36.1% (25,720/71,291) of the layouts and 63.9% (10,003/15,642) of applications. The prevalence of ambiguity clearly demonstrates limitations of key-phrase-based techniques for resolving semantics. For example, SUPOR ignores the resolution of the term “name,” which impacts the resolution of 5.6% (9,753/175,101) of input fields being semantically resolved. Similarly, the term “address” affects 1.8% (3,228/175,101) of semantics resolutions. The pervasiveness of ambiguity when resolving semantics demonstrates the necessity of a disambiguation technique.

### 3.5.3.3 Disambiguation Performance

To evaluate UiRef’s disambiguation technique, we evaluate UiRef’s accuracy on 19 ambiguous terms shown in Table 3.3. For each ambiguous term, we randomly select 25 input requests that contain the ambiguous term as a hint, label, or text attribute of an input widget from the dataset in Section 3.5. Note that for terms that do not appear in at least 25 input requests (e.g., “cc”), we select the maximum number of samples available. We substitute layouts with another randomly selected layout if we cannot manually resolve ambiguity when viewing the screenshot. Further, since certain layouts are duplicated across applications and can skew the evaluation (e.g., over-approximating accuracy), we substitute layouts with another randomly selected layout if it is a duplicate of a previously annotated layout for that term. Our dataset consists of 362 input requests from 296 layouts (250 applications). For each layout, we manually resolve ambiguity by viewing the screenshot and XML file and then compare our manual annotation to UiRef’s prediction.

Note that UiRef does not resolve the semantics of an input widget requesting non-sensitive data (e.g., cement age for the term “age”), occurring with 26 widgets in our dataset. Such result is due to word-sense induction not learning a *sense* for the concept due to under-representation in the dataset. However, unresolved semantics for non-sensitive requests should not affect the results, as UiRef’s goal is to identify sensitive requests. Thus,

we remove these 26 widgets from our dataset, resulting in 335 input requests.

**Results:** Table 3.3 shows that UiRef achieves an 82.1% accuracy for resolving the ambiguity of sensitive terms (275/335). The results demonstrate that UiRef’s disambiguation provides a significant advantage over key-phrase-based techniques, such as SUPOR. In particular, UiRef resolves the semantics of the term “address” with 88.0% accuracy (22/25) where SUPOR ignores the term due to ambiguity. UiRef’s disambiguation also shows a 91.6% accuracy (22/24) when distinguishing between credit-card verification (CCV) codes and passphrases denoted by the term “security code.” In this case, UiRef correctly disambiguates “security code” as CCV code (15 widgets), and as a passphrase (7 widgets).

## 3.6 Security and Privacy Analysis

Our primary motivation for creating UiRef was to classify the security and privacy sensitive inputs provided by users. In this section, we use UiRef to perform a large-scale study of 50,162 Google Play applications from Section 3.5 to understand the scope of questionable security and privacy practices. We ran UiRef on the dataset and triaged applications using outlier detection for manual inspection. We aim to explore two main research questions:

**RQ1:** *What types of security and privacy sensitive information are mobile applications asking for?*

**RQ2:** *What are the security and privacy implications of sensitive input requests?*

Note that our analysis is not intended to be comprehensive. Rather, we intend to broadly highlight various forms of questionable practices by application developers. Our hope is that our initial findings will drive further research in understanding the security and privacy challenges associated with user input requests. Extended results are available on the project website [Uir].

### 3.6.1 Sensitive Information Requests

To answer RQ1, we consolidated the sensitive input requests identified by UiRef for every application. We grouped this information into 9 categories based on high-level semantics (Table 3.4). To further understand the sensitive input requests made by applications, we use the following methodology: (1) We manually identify relevant sensitive concepts extracted by UiRef, (2) for interesting input requests, we view the layout’s screenshot, and (3) if further

**Table 3.4** Consolidated Sensitive Information Requests

Category	Sensitive Information Requests (# Applications)
Account/Contact	username_or_email (11047), passwd (10251), phone_num (6307), twitter_passwd (55), wifi_passwd (17), gmail_passwd (10), social_media_url (8), wifi_ssid (7), ftp_passwd (6), smtp_passwd (5), facebook_passwd (2), nq_account_passwd (1), mint_passwd (1), gritsafe_passwd (1), exchange_passwd (1), adobe_passwd (1)
Personal	person_name (8057), date_of_birth (2285), gender (1237), company_name (499), person_age (178), person_weight (149), job_title (121), person_height (109), school_name (75), education_info (38), marital_status (37), demographic_info (27), native_language (8), citizenship (7), birth_place (7), marriage_date (5), religion (4), political_affiliation (3),
Financial	credit_card_info (4433), loan_info (1974), bank_account_info (156), salary (116), bank_info (78), house_financial_info (57)
Vehicle/Driver	vehicle_info (173), license_plate (66), vehicle_vin (61), insurance_policy_num (52), vehicle_registration (11), license_expiry_date (3)
Device	device_id (106), mac_address (27), serial_num (26), service_provider (19), device_manufac_info (6), sim_card (6),
Health	medication_name (70), prescription_num (27), drug_dosage (23), blood_pressure (22), blood_type (16), heart_rate (14), body_mass_index (11), blood_glucose (6), doctor_email_id (2)
Personal Id #	SSN (52), driver_license_num (51), id_num (20), tax_id (5), passport_num (3), student_id (1), medicaid_num (1)
Family Member	family_member_name (30), family_member_phone (9), guardian_email (3), mother_birth_place (2)
Location/Travel	location_info (6761), flight_num (28)

clarification is required, we analyze the application’s disassembled code. Our analysis results in several interesting findings as follows.

**Finding 1:** *Applications request a wide-range of security and privacy sensitive information.*

Table 3.4 shows the types of sensitive information requested by applications. The most frequent information requests are related to account or contact information (e.g., usernames/email addresses, passwords), which is due to applications requiring account login or registration. Applications also request a substantial amount of personal information (e.g., the person’s name, date of birth, gender, age, race, marital status, religion, and political affiliation). Other requests include sensitive personal identifiers (e.g., SSN - 52 applications, driver’s license number - 51 applications). Finally, applications request a range of financial data (e.g., credit card numbers), vehicular data (e.g., vehicle identification numbers—VIN), location data (e.g., postal addresses), and device data (e.g., IMEI). IMEI represents an interesting use case as the applications are asking the user to input the IMEI rather than retrieving the IMEI in the background. Note that for this study, we determine the sensitivity of the terms based on own experiences, however, UiRef’s approach is generic and can be easily adapted for alternative terms with different sensitivities (see Section 3.7).

**Finding 2:** *Applications directly request third-party passwords defeating the purpose of OAuth-like solutions.* Knowledge of a password leads to full control of users’ accounts. A primary goal of OAuth is to allow third-parties to access a user’s resources without having

direct access to their password. Further, the user can revoke access without changing the password. If the user enters the password directly into the application, the application can still request an OAuth token; however, it eliminates much of OAuth's benefits.

We found 68 applications directly requesting the user's Twitter (55 apps), Gmail (10 apps), Facebook (2 apps), and Adobe (1 app) credentials within internally-defined layouts. Most Twitter password requests (51/55) come from the WinterWell JTwitter library, which requests the user's password within a static Android layout. The remaining 4 applications requesting the user's Twitter password use other third-party libraries to request an OAuth token (e.g., Twitter4J). During our analysis, we found that the JTwitter library also has an option to request OAuth tokens within an embedded WebView. This is still a bad practice, as the application that embeds the WebView can access all of the WebView's data. In fact, Google deprecated support for OAuth requests to Google in embedded WebViews for security concerns [Goa]. The applications requesting Gmail (10), Facebook (2), and Adobe (1) passwords directly request the data for login purposes and are subject to the same problem.

**Finding 3:** *Applications frequently request sensitive financial information.* We found that around 8% of applications (4,433/50,162) are requesting the user's credit card information (e.g., credit card number, security code, and expiration date). Exposing credit card details and other financial information to untrusted third-party applications is high risk, as demonstrated by the numerous data breaches in recent years [Pri]. Further, applications that are not fully compliant to the Payment Card Industry's Data Security Standard [Pci] (PCI DSS) place the user at potential risk for financial loss and fraud. PCI DSS outlines standards for merchants that handle credit card information, such as encrypted storage and transmission. Based on our findings, we believe that there is a need for deeper analysis to identify non-compliance, and explore alternate payment solutions, such as opting for centralized and well-tested billing processors (e.g., Google's In-app Billing service), to reduce associated risks.

### 3.6.2 Identifying Anomalous Input Requests

To explore RQ2, we used outlier analysis to triage applications for manual analysis. We apply an unsupervised technique, called the Ranking-based Outlier Analysis and Detection (ROAD) algorithm [Sur12] at the granularity of Google Play's 25 application categories (games subcategories combined into one category), to generate a ranked list based on the

likelihood that a record is an outlier. Note that we modify the distance function of the ROAD algorithm by using a probability-based value weighting function, so that rare attributes have a larger influence on the distance function. An application is considered to be an outlier based on its distance from the largest cluster (or second largest cluster if the largest cluster requests no sensitive input), where clusters are formed based on the types of sensitive information they request. Interestingly, only the Weather category had sensitive input requests associated with the largest cluster. Section 3.4.4 provides a detailed explanation on ROAD and how we use it to detect outliers in our dataset. Table A.1 (appendix) shows the results of our outlier analysis, where the counts represent the number of applications.

Our approach is to use outlier analysis as a filtering mechanism to identify applications that merit further deeper analysis. We focus our manual analysis on the top-10 outlier applications (based on the distance from the largest cluster) for each of the 25 Google Play categories (250 apps in total). Our deeper manual analysis involves reviewing the screenshots of the application's layouts and its disassembled code to explore how the application is using the data. The following discussion reports our high-level findings.

**Finding 4:** *Applications are making questionable requests asking for too much sensitive data.* Table A.1 (Appendix) shows the contrast between expected and unexpected input requests by outlier apps. When the largest cluster requests a concept (e.g., username) it is not unexpected for the outlier to also request it. However, outlier analysis identifies a number of interesting cases where the requested data does not align with the app's category. For example, in the Communication category, outliers request the user's religion, marital status, and demographics. We inspected these apps and found that some request the data to send to advertisers and disclose this purpose to users, while others request it to search for friends.

In the Personalization category, outlier applications ask for data such as the user's birthday, birth place, and bank account information. Through manual inspection, we found the app (com.Chinese\_I\_Ching\_Horoscope\_20706) requests the user's birthday and birthplace to provide horoscope readings, but send the data to a third-party website to provide the service without notifying the user. The app (psmainapp-ui) requesting bank account information (bank name and credentials) provides a vault for encrypted data storage. Upon manual analysis, we found that the app uses a constant seed for key generation ('sknpyvvn'), which puts the users' data at risk.

Similarly, in the Game category, outliers ask for a wide-range of personal information,

such as the user's name, salary, age, marital status, gender, and SSN, which we discuss further in Findings 5-7.

**Finding 5:** *Applications disclose sensitive input requests to advertisers.* We found 6 game applications requesting the user's zip code, age, salary, gender, marital status, education information, ethnicity, and political affiliation. On further analysis, we found that they were disclosing this information to the Millennial Media advertising network for targeted advertisements. The apps were all developed by the same developer (Brett Plummer), and 3 of them are relatively popular<sup>2</sup>, as they have 100-500k downloads. The apps collect the sensitive data requests in layouts claiming to be used for profile information, which is misleading as it is only used to disclose to advertisers. Further, the privacy policies and layouts do not specify that the sensitive data provided is being disclosed to advertisers.

**Finding 6:** *Applications include third-party libraries that directly request sensitive information.* We found 2 game apps<sup>3</sup> that include a third-party library (i.e., Skillz eSports) that displays layouts to request a wide-range of sensitive information, such as SSNs, passport numbers, names, addresses, credit card data, and phone numbers. Although we could not successfully run the apps to verify the requests due to crashing, we viewed the Skillz eSports's website and found that it requests this data to allow users to withdraw their winnings. The main concern in such cases is that the user may not understand to which entity they are disclosing information. For example, consider the case where the user trusts the main application and is willing to disclose their SSN to that application, but does not trust Skillz eSports with their information. Since the layouts do not denote who (library or application) is collecting the information, users may potentially expose data to third-parties they do not trust.

**Finding 7:** *Applications are displaying untrusted ads in the same windows as sensitive data requests.* We found the same 6 applications discussed in Finding 5 displaying ads in layouts that requests sensitive input. Prior work [RK13] showed that untrusted code can traverse UIs to steal private data in layouts. For example, an untrusted widget loaded within the same view hierarchy can obtain the root view, and then traverse back down the hierarchy to search for input widgets with sensitive data. Our finding demonstrates the need for mechanisms such as LayerCake [RK13].

**Takeaway:** The findings from our study motivate the need to analyze how apps use sensitive data requested through input widgets. These findings were possible due to the availability

---

<sup>2</sup>com.alaskajim.{football, bible2, rockmusic}

<sup>3</sup>com. binarypumpkin.bingo.{easter,usa}

of reliable user input semantics from UiRef. Although our analysis is not comprehensive, we found several concerning practices and sensitive input requests with limited manual analysis, further motivating the need for automated analysis to focus on the apps triaged by UiRef.

### 3.7 Discussion

**Threats to External Validity:** The dataset used in the study may not be representative of the entire market. To mitigate this threat, we use proportionate stratified sampling to select our dataset, using the application categories and popularity to form the strata. Further, UiRef’s outlier detection assumes that the majority of applications are well behaved and not requesting an extraneous amount of data. Although this assumption is acceptable for Google Play applications, this assumption may not be transferable to other markets.

**Threats to Internal Validity:** UiRef extracts the statically defined layouts from applications. Applications may also dynamically generate and modify layouts at runtime, and display content in WebViews, which may result in false negatives. Reconstructing dynamic layouts using static analysis is a challenging problem, left as future work. Applications may also include unreachable or unused layouts in the APKs, such as layouts bundled with libraries or as artifacts from testing. As this characteristic can lead to false positives, identifying reachable layouts from the applications’ entry points may mitigate this problem. Although textual labels are commonly used for internationalization, icons and images may also denote semantics, leading to false negatives. Integrating optical character and image recognition can be conducted in future work.

Automatically extracting a comprehensive lexicon of security and privacy related terms is an open problem and warrants future investigation. Although an incomplete lexicon may result in false negatives, as UiRef uses it to determine types of information considered private, UiRef’s techniques are general and the list can be substituted once a more comprehensive lexicon becomes available. Further, users commonly progress through sequences of layouts to accomplish tasks within applications, such that prior layouts may provide context into the semantics of the current layout. The lack of prior context may lead to imprecision when resolving certain layouts. In future work, we plan to explore leveraging context from prior layouts in workflows to assist in resolving semantics.

### 3.8 Related Work

Although there has been a considerable amount of work that focuses on mobile malware analysis, we limit our discussion to privacy analysis of mobile applications. TaintDroid [Enc10] pioneered the area by using dynamic taint analysis to identify Android applications that leak private information, such as GPS location and device identifiers. PiOS [Ege11] and AndroidLeaks [Gib12] are the initial static analysis duals of TaintDroid for iOS and Android, respectively. Due to the various challenges associated with statically analyzing Android applications, a number of additional static-analysis frameworks [Arz14b; Oct13; Oct12; FWR14; Gor15] have been proposed. Other program information than information flows is also used to study privacy in mobile applications. Han et al. [Han13a] compare API calls within iOS and Android applications, and show iOS applications call significantly more privacy-sensitive APIs than their Android counterparts. In prior work, the privacy semantics of information is unambiguous, as it comes from well-defined APIs. In contrast, UiRef resolves privacy semantics from ambiguous user-input APIs.

The act of sending privacy-sensitive information to the network does not constitute a violation. AppIntent [Yan13] pairs screenshots with potential privacy leaks for human review. Recent research has sought to lessen the need for human review by using natural language processing. WHYPER [Pan13], AutoCog [Qu14], and CHABADA [Gor14] consider the application’s text description to help infer the user’s expectation of security and privacy relevant actions. AsDroid [Hua14] checks the coherence between UI text and event callbacks triggering sensitive behavior. UiRef complements these approaches by providing comprehensive contextual semantics of the UI. Pluto [Dem16] assesses user data exposure by resolving the semantics of data originating from well-structured files (e.g., SQL, XML, JSON). Closest to our work are SUPOR [Hua15] and UIPicker [Nan15] for which we provide a detailed comparison in Section 3.3.

Other work has focused on using program analysis to extract the structure and sequence of GUIs as intermediate representations. GATOR [She15] extracts GUI-layout hierarchies and transition graphs for test-input generation by performing static reference analysis and control-flow analysis. A<sup>3</sup>E [AN13] constructs activity-transition graphs to drive automated application exploration by using static taint tracking to resolve intents that flow to method invocations that launch activities. Test-input generation and automated exploration tools would benefit from UiRef by using it to resolve the semantics of input widgets, and use such

semantics to generate input data.

### **3.9 Conclusion**

While prior studies of Android security and privacy have focused on information from well-defined APIs, they have largely ignored user inputs as a source of sensitive information. In this chapter, we have presented UiRef for resolving the security and privacy semantics of data entered into input widgets. We have introduced novel techniques that achieve overall accuracies of 95.0% at semantics resolution and 82.1% at disambiguation. We have used UiRef to perform a large-scale study of 50,162 apps, and identified concerning practices, including insecure exposure of account passwords and non-consensual input disclosures to third parties. Our findings demonstrate that user-input semantics could provide a unique perspective into improving mobile-app security and privacy.

## CHAPTER

### 4

# POLICYLINT: INVESTIGATING INTERNAL PRIVACY POLICY CONTRADICTIONS ON GOOGLE PLAY

In this chapter <sup>1</sup>, we characterize and measure the impact of self-contradictory policy statements within privacy policies. To do so, we create PolicyLint, a privacy policy analysis tool that identifies self-contradictory policy statements by simultaneously considering negation and varying semantic levels of data objects and entities. To do so, PolicyLint automatically generates ontologies from a large corpus of privacy policies and uses sentence-level natural language processing to capture both positive and negative statements of data collection and sharing. We use PolicyLint to analyze the policies of 11,430 apps and find that 14.2% of policies contain contradictions that may be indicative of misleading statements. We manually verify 510 contradictions, identifying concerning trends that include the use of misleading presentation, attempted redefinition of common understandings of terms, con-

---

<sup>1</sup>Portions of this chapter appear in Andow et al. [And19]

flicts in regulatory definitions (e.g., US and EU), and “laundering” of tracking information facilitated by sharing or collecting data that can derive sensitive information. In doing so, PolicyLint significantly advances automated analysis of privacy policies.

## 4.1 Introduction

Mobile apps collect, manage, and transmit some of the most sensitive information that exists about users — including private communications, fine-grained location, and even health measurements. These apps regularly transmit this information to first or third parties [Enc10; Arz14b; Gra12b]. Such data collection/sharing is often considered (legally) acceptable if it is described in the privacy policy for the app. These policies are sophisticated legal documents that are typically long, vague, and difficult for novices, experts, and algorithms to interpret. Accordingly, it is difficult to determine whether app developers adhere to privacy policies, which can help app markets and other analysts identify privacy violations, or help end users choose more-privacy-friendly apps.

Recent work has begun studying whether or not mobile app behavior matches statements in privacy policies [Sla16; Yu16; Zim17b; Wan18]. However, the prior work fails to account for contradictions within privacy policies, which may lead to incorrect interpretation of sharing and collection practices. Identifying contradictions requires overcoming two main challenges. First, privacy policies refer to information at different semantic granularities. For example, a policy may discuss its practices using broad terms (e.g., “personal information”) in one place in the policy, but later discuss its practices using more specific terms (e.g., “email address”). Prior approaches have tackled this issue by crowdsourcing data object ontologies [Sla16; Wan18],<sup>2</sup> but such crowdsourced information is not complete, accurate, or easily collected. Second, prior approaches have struggled to accurately detect negative statements, relying on bi-grams (e.g., “not share”) [Zim17b] or detecting only verb modifiers [Yu16] while neglecting the more-complicated statements (e.g., “will share X except Y”) that are common in privacy policies. Modeling negative statements is required to determine the correct meaning of a policy statement (i.e., “not sharing” information versus “sharing” information). Fully characterizing contradictions requires simultaneously addressing both challenges.

---

<sup>2</sup>Ontologies are graph data structures that capture relations among entities. For example, “personal information” subsumes “your email address.”

In this chapter, we present PolicyLint for automatically identifying potential contradictions of sharing and collection practices in software privacy policies. Contradictions make policies unclear, confusing both humans and any automated system that rely on interpreting the policy. Considering these uses cases, PolicyLint defines two contradiction groupings. *Logical contradictions* are contradictory policy statements that are more likely to cause harm if users and analysts are not aware of the contradictory statements. One example is a policy that initially claims not to collect personal information, but later in fine print discloses collecting a user’s name and email address for advertisers. *Narrowing definitions* may cause automated techniques that reason over policy statements to make incorrect or inconsistent decisions and may result in vague policies. PolicyLint is the first tool to have the sophistication necessary to reason about both negative statements and statements covering varying levels of specificity, being necessary for uncovering contradictions.

PolicyLint is inspired by other security `lint` tools [Eva94; Eva00; EL01; EL02; Ege13; Joh78], which analyze code for indicators of potential bugs or other programming errors. Like any static approach, not every `lint` finding is necessarily an error. For example, potential bug conditions could be mitigated by an external control or other context that the tool cannot verify. In many cases, only a human can verify the outputs of a `lint` finding. In the case of PolicyLint, we note that privacy policies are complex legal documents that may be intentionally vague, ambiguous, or even deliberately misleading even for human interpretation. Despite these fundamental challenges, PolicyLint condenses long, complicated policies into a small set of candidate issues of interest to human or algorithmic analysis.

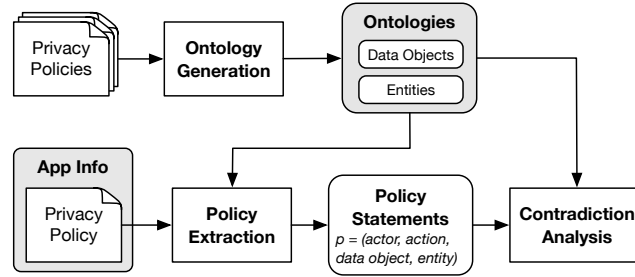
This chapter makes the following contributions:

- **Automated generation of ontologies from privacy policies.** PolicyLint uses an expanded set of Hearst patterns [Hea92] to extract ontologies for both data objects and entities from a large corpus of privacy policies (e.g., “W such as X, Y, and Z”). PolicyLint is more comprehensive and scalable than crowdsourced efforts [Sla16; Wan18]
- **Automated sentence-level extraction of privacy practices.** PolicyLint uses sentence-level NLP and leverages parts-of-speech and type-dependency information to capture data collection and sharing as a four-tuple: (actor, action, data object, entity). For example, “We [actor] share [action] personal information [data object] with advertisers [entity].” Sentence-level NLP is critically important for the correct identification of

negative statements. We also show that prior attempts at analyzing negation would fail on 28.2% of policies.

- **Automated analysis of contradictions in privacy policies.** We formally model nine types of contradictions that result from semantic relationships between terms, providing an algorithmic method to detect contradictory policy statements. Our groupings of *narrowing definitions* and *logical contradictions* lay the foundation for ensuring the soundness of future automated policy tools and identifying potentially misleading policy statements. In a study of 11,430 privacy policies from mobile apps, we are the first to find that logical contradictions and narrowing definitions are *rampant*, affecting 17.7% of policies, with logical contradictions affecting 14.2% of policies.
- **Manual analysis of contradictions to identify trends.** The high ratio of policy contradictions is surprising. We manually review 510 contradictions across 260 policies, finding that many contradictions are indeed indicators of misleading or problematic policies. These contradictions include making broad claims to protect personal information early in a policy, yet later carving out exceptions for data that the authors attempt to redefine as not personal, that could be used to derive sensitive information (e.g., IP addresses and location), or that are considered sensitive by some regulators but not others.

PolicyLint has four main potential use cases. First, policy writers can leverage PolicyLint to reduce the risk of releasing misleading policies. In fact, when we contacted parties responsible for the contradictory policies, several fixed their policies (Section 4.4.4). Second, regulators can use PolicyLint’s definition of logical contradictions to identify deceptive policies. While the FTC has identified contradictory statements as problem areas within privacy policies [Com00], to the best of our knowledge, there is no legal precedent regarding whether regulatory agencies would take action as a result of contradictory policies. However, we believe that some of our findings in Section 4.4 potentially fall under the FTC’s definition of deceptive practices [Ftca]. We envision that regulators could deploy PolicyLint to audit companies’ privacy policies for misleading statements at large-scale. Third, app markets, such as Google Play, can deploy PolicyLint similarly to ensure apps posted in the store do not have misleading statements in the privacy policy. Further, they can also use PolicyLint’s extraction of policy statements to auto-generate privacy labels to display on the markets to nudge users to less privacy-invasive apps. Finally, automated techniques for analyzing



**Figure 4.1** Overview of PolicyLint

privacy policies can use PolicyLint’s techniques of fine-grained policy statement extraction and formalization of logical contradictions and narrowing definitions to help ensure tool soundness.

This chapter is organized as follows. Section 4.2 describes PolicyLint’s design. Section 4.4 reports on our empirical study using PolicyLint. Section 4.5 discusses limitations and future work. Section 4.6 describes related work. Section 4.7 concludes.

## 4.2 PolicyLint

PolicyLint seeks to identify contradictions within individual privacy policies for software. It provides privacy *warnings* based on contradictory sharing and collection statements within policies, but similar to lint tools for software, these warnings require manual verification. PolicyLint identifies “candidate contradictions” within policies. A candidate contradiction is a pair of contradicting policy statements when considered in the most conservative interpretation (i.e., context-insensitive). Candidate contradictions that are validated by analysts are termed as “validated contradictions.” Manual verification is required due to the fundamental problems of ambiguity when interpreting the meaning of natural language sentences (i.e., multiple interpretations of the same sentence).

For example, consider the privacy policy for a popular recipe application (`com.omnilux-trade.allrecipes`). One part of the policy states “We do not collect personally identifiable information from our users.” It is clear from this sentence that the application does not collect any personal information. However, later the policy states, “We may collect your email address in order to send information, respond to inquiries, and other requests or questions.” Such sentence is a clear contradiction to the earlier sentence, as email address

is considered personal information. As discussed in detail in Section 4.4, the cause for this underlying contradiction is that the developer does not consider email address as personal information.

To our knowledge, our work is the first to characterize and automatically analyze contradictions within privacy policies. While PolicyLint is not the first NLP tool to analyze privacy policies, identifying contradictions requires addressing two broad challenges.

- *References to information are expressed at different semantic levels.* Prior work [Sla16; Wan18] uses ontologies to capture subsumptive (i.e., “is-a”) relationships between terms; however, such ontologies are crowdsourced and subsumptive relations are manually defined by the authors, leaving concerns of comprehensiveness and scalability. For example, prior work [Sla16; Wan18] built their ontology using only 50 and 30 policies, respectively. While crowdsourced ontologies could be comprehensive given unlimited time and manpower, crowd-sourcing at large scale is infeasible due to limited resources. Further, existing general-purpose ontologies do not capture all of the specific relations required to reason over data types and entities mentioned within privacy policies.
- *Privacy policies include negative sharing and collection statements.* Most of the prior work [Sla16; Wan18] operate at paragraph level and cannot capture negative sharing statements. Prior work that does capture negative statements [Zim17b; Yu16] misses complex statements (e.g., “will share personal information except your email address”). These works extract coarse-grained policy statement summaries (paragraph-level [Sla16; Wan18], document-level [Zim17b]) and can never precisely model negative statements or the entities involved. Their imprecision may result in incorrectly reasoning about 28.2% of policies due to their negation modeling (Finding 1 in Section 4.4).

We tackle these challenges using two key insights.

**Sentence structure informs semantics:** Sharing and collection statements generally follow a learnable set of templates. PolicyLint uses these templates to extract a four tuple from such statements: (actor, action, data object, entity). For example, “We [actor] share [action] personal information [data object] with advertisers [entity].” The sentence structure also provides greater insight into more complex negative sharing. For example, “We share personal information except your email address with advertisers.” PolicyLint extracts

such semantics from policy statements by building on top of existing parts-of-speech and dependency parsers.

**Privacy policies encode ontologies:** Due to the legal nature of privacy policies, general terms are often defined in terms of examples or their constituent parts. While each policy might not define semantic relationships for all of the terms used in the policy, those relationships should exist in some other policies in our dataset. By processing a large number of privacy policies, PolicyLint automatically generates an ontology specific to policies (one for data objects and one for entities). PolicyLint extracts term definitions using Hearst patterns [Hea92], which we have extended for our domain.

Figure 4.1 depicts the data flow that comprises PolicyLint.

There are three main components of PolicyLint: ontology generation, policy extraction, and contradiction analysis. The following sections describe the design of these components in detail.

### 4.3 Preprocessing Privacy Policies

Privacy policies are commonly made available via a link on Google Play to the developer’s website and hosted in HTML. Most NLP parsers expect plaintext input, therefore PolicyLint begins by converting the HTML privacy policy into plaintext. We now describe how PolicyLint achieves this conversion.

**Removing Non-relevant and Non-displayed Text:** Privacy policies are frequently embedded as main content on a webpage containing navigational elements and other non-relevant text. Additionally, non-displayed text should also be stripped from the HTML, as we want to analyze what is actually displayed to users. PolicyLint extracts the privacy policy portion of the Webpage by iterating over the elements in the HTML document. To remove non-relevant text, PolicyLint strips comment, style, script, nav, and video HTML tags. PolicyLint also strips HTML links containing phrases commonly used for page navigation (e.g., “learn more,” “back to top,” “return to top”). Finally, PolicyLint removes HTML span and div tags using the "display:none" style attribute.

**Converting HTML to Flat Plaintext Documents:** Certain HTML elements, such as pop-up items, result in a non-flat structure. When flattening of the HTML documents, PolicyLint must ensure the plaintext document has a formatting style similar to the text displayed on the webpage (e.g., same paragraph and sentence breaks). PolicyLint handles pop-up

elements by relocating the text within the pop-up element to the end of the document. Pop-up elements often provide additional context, an explanation, clarification, or a definition of a term. Therefore, relocating these elements should not have a significant effect on processing the referencing paragraph. To ensure formatting style is maintained, PolicyLint converts the HTML document to markdown using `html2text`.

**Merging Formatted Lists:** Formatted lists within text can cause NLP parsers to incorrectly detect sentence breaks or incorrectly tag parts-of-speech and typed dependencies. These parsing errors can negatively impact the semantic reasoning of sentences. Therefore, PolicyLint merges the text within list items with the preceding clauses before the list begins. PolicyLint also uses a set of heuristics for nesting list structures to ensure list items propagate to the correct clause.

PolicyLint merges formatted lists in two phases. The first phase occurs before the aforementioned conversion to markdown. In this phase, PolicyLint iterates over HTML elements using list-related HTML tags (i.e., `ol`, `ul`, `li`) to annotate list structures and nesting depth. The second phase occurs after the conversion to markdown. In this phase, PolicyLint searches for paragraphs ending in a colon where the next sentence is a list item (e.g., starts with bullets, roman numerals, formatted numbers, or contains annotations from the first phase). It then forms complete sentences by merging the list item text with the preceding text.

PolicyLint iterates over the paragraphs in the markdown document to find those that end in a colon. For each paragraph that ends in a colon, PolicyLint checks whether the preceding paragraph is a list item. If the line of text is a list item, PolicyLint creates a new paragraph by appending the list item text to the preceding text that ended with the colon. If the list item ends in another colon, PolicyLint repeats the same process above but by prepending the nested list items to the new paragraph created in the last step. PolicyLint then leverages the symbols that denote list items to predict the next list item's expected symbol, which is useful for detecting boundaries of nested lists. For example, if the current list item starts with "(1)," then we would expect the next list item to start with "(2)." If the item symbol matches to expected symbol, PolicyLint merges the list item text as discussed above and continues this process. If the item symbol does not match the expected symbol, PolicyLint stops this process and returns.

**Final Processing:** The final step converts markdown to plaintext. PolicyLint strips markdown formatting such as header tags and bullet points, normalizes Unicode characters,

and strips list item numbering and other format characters. PolicyLint then uses `langid` to determine if the majority of the document is written in English. If not, PolicyLint discards the document. If so, PolicyLint outputs the plaintext document.

### 4.3.1 Ontology Generation

The goal of the ontology generation is to define subsumptive (“is-a”) relationships between terms in privacy policies to allow reasoning over different granularities of language. PolicyLint operates on the intuition that subsumptive relationships are often embedded within the privacy policy text, e.g., an example of the types of data considered to be a specific class of information. The following example identifies that demographic information subsumes age and gender.

**Example 1.** *We may share demographic information, such as your age and gender, with advertisers.*

PolicyLint uses such sentences to automatically discover subsumptive relationships across a large set of privacy policies. It focuses on data objects and the entities receiving data objects.

PolicyLint uses a semi-automated and data-driven approach for ontology generation. It breaks ontology generation into three main parts. First, PolicyLint performs domain adaptation of an existing model of statistical-based named entity recognition (NER). NER is used to label data objects and entities within sentences, capturing not only terms, but also surrounding context in the sentence. Second, PolicyLint learns subsumptive relations for labeled data objects and entities by using a set of 11 lexicosyntactic patterns with enforced named-entity label constraints. Third, PolicyLint takes a set of seed words as input and generates data-object/entity ontologies using the subsumptive relations discovered in the prior step. It iteratively adds relations to the ontology until a fixed point is reached. We now describe this process in detail.

#### 4.3.1.1 NER Domain Adaptation

To identify subsumptive relations for data objects and entities, PolicyLint must identify which sentence tokens represent a data object or entity. For Example 1, we seek to identify “demographic information,” “age,” and “gender” as data objects, and “we” and “advertisers” as entities. PolicyLint uses a statistical-based approach of named-entity recognition (NER)

**Table 4.1** NER Performance: Comparison of spaCy’s stock *en\_core\_web\_lg* model versus our domain adapted model

	Overall		Data Objects		Entities	
	Default	Adapted	Default	Adapted	Default	Adapted
Precision	43.48%	84.12%	-	82.20%	61.22%	86.75%
Recall	8.33%	81.67%	-	79.84%	17.75%	85.21%
F1-Score	13.99%	82.88%	-	81.00%	27.52%	85.97%

to label data objects and entities within sentences. Prior research [Sla16; Wan18; Yu16; Zim17b] proposed keyphrase-based approaches for identifying data objects. However, keyphrase approaches are less versatile in practice: they cannot handle term ambiguity and variability, and they can identify only terms in their pre-defined list. For example, “internet service provider” can be both a data object and entity, which cannot be differentiated by keyphrase-based approaches. In contrast, statistical-based NER both resolves ambiguity and discovers “unseen” terms.

Unfortunately, existing NER models are not trained for our problem domain (data objects and collective terms describing entities, e.g., “advertisers”). Training an NER model from scratch is time-consuming due to the large amount of training data required to achieve reasonable performance. Therefore, PolicyLint takes an existing NER model and updates it using annotated training data from our problem domain. Specifically, PolicyLint adopts spaCy’s NER engine [HM17], which uses deep convolutional neural networks. We adapt the *en\_core\_web\_lg* model to the privacy policy domain.

To perform domain adaptation, we gather 500 sentences as training data. Our training data is selected as follows. First, we randomly select 50 unique sentences from our policy dataset. Second, for each of the 9 lexicosyntactic patterns described in Section 4.3.1.2, we randomly select 50 sentences that contain the pattern (450 in total). We run the existing NER model on the training sentences to prevent the model from “forgetting” old annotations. We then manually annotate the sentences with data objects and entities.

When updating the existing NER model, we perform multiple passes over the annotated training data, shuffling at each epoch, and using minibatch training with a batch size of 4. To perform the domain adaptation, the current model predicts the NER labels for each word in the sentence and adjusts the synaptic weights in the neural network accordingly if the prediction does not match the annotation. We stop making passes over the training data when the loss rate begins to converge. We annotate an additional 100 randomly selected sentences as holdout data for testing the model. Table 4.1 shows the NER model perfor-

**Table 4.2** Lexicosyntactic patterns for subsumptive relations

#	Pattern
H1	<i>X, such as <math>Y_1, Y_2, \dots, Y_n</math></i>
H2	<i>such X as <math>Y_1, Y_2, \dots, Y_n</math></i>
H3	<i>X [or and] other <math>Y_1, Y_2, \dots, Y_n</math></i>
H4	<i>X, including <math>Y_1, Y_2, \dots, Y_n</math></i>
H5	<i>X, especially <math>Y_1, Y_2, \dots, Y_n</math></i>
H'1	<i>X, [e.g. i.e.], <math>Y_1, Y_2, \dots, Y_n</math></i>
H'2	<i>X ([e.g. i.e.], <math>Y_1, Y_2, \dots, Y_n</math>)</i>
H'3	<i>X, for example, <math>Y_1, Y_2, \dots, Y_n</math></i>
H'4	<i>X, which may include <math>Y_1, Y_2, \dots, Y_n</math></i>

\* H\* = Hearst Pattern; H'\* = Custom Pattern

mance before and after domain adaptation for our holdout dataset. PolicyLint achieves 82.2% and 86.8% precision for identifying data objects and entities, respectively.

#### 4.3.1.2 Subsumptive Relation Extraction

PolicyLint uses a set of 9 lexicosyntactic patterns to discover subsumptive relations within sentences, as shown in Table 4.2. The first five are Hearst Patterns [Hea92], and the last four are custom deviations based on observations of text in privacy policies. For each pattern, PolicyLint ensures named-entity labels are consistent across the pattern (i.e., PolicyLint uses Hearst patterns enforcing constraints on named-entity labels). For example, Example 1 is recognized by the pattern “*X, such as  $Y_1, Y_2, \dots, Y_n$* ” where *X* is a noun,  $Y_1, Y_2, \dots, Y_n$  is a conjunctive noun phrase, and the NER labels for *X* and each  $Y_i$  are all data objects. Note that PolicyLint merges noun phrases before applying the lexicosyntactic patterns to ease extraction.

Given the set of extracted relations, PolicyLint normalizes the relations by lemmatizing the text and substituting terms with their synonym. For example, consider that we know “blood sugar levels” is a synonym for “blood glucose level.” Lemmatization turns “blood sugar levels” into “blood sugar level,” and synonym substitution turns it into “blood glucose level.” To identify synonyms, we output non-terminal (i.e., *X* value of the Hearst patterns) data objects and entities in the subsumptive relations. We manually scan through the list and mark synonyms. We repeat the process with the terminal nodes that are included after constructing the ontology. We then output the data objects and entities labeled from all policies and sort the terms by frequency. We mark synonyms for the most frequent terms

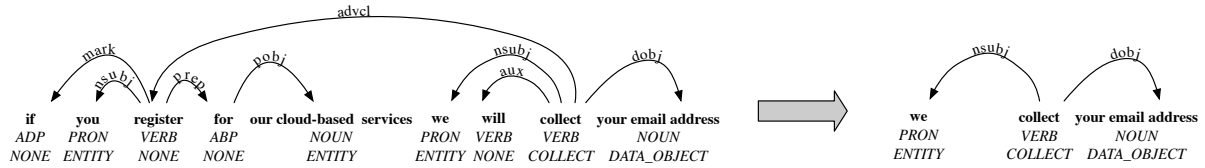
**Table 4.3** Seed terms used for ontology construction

<b>Ontology</b>	<b>Seeds</b>
Data Ontology	information, personal information, non-personal information, information about you, biometric information, financial information, device sensor information, government-issue identification information, vehicle usage information
Entity Ontology	third party

by keyword searching for related terms based on sub-strings and domain knowledge. For example, if “location” appears as a frequent term, we output all data objects that contain the word “location,” read through the list, and mark synonyms (e.g., “geographic location”). Next, we use domain knowledge to identify that “latitude and longitude” is a synonym of “location,” output the terms that contain those words, and manually identify synonyms.

#### **4.3.1.3 Ontology Construction**

PolicyLint generates ontologies by combining the subsumptive relations extracted from policies with a set of seed terms (Table 4.3). For each ontology, PolicyLint iterates through the seeds, selecting relations that contain it. PolicyLint then expands the term list from the relations in that iteration. PolicyLint continues iterating over the relations until no new relations are added to the ontology. If there exists any inconsistent relation where X is subsumed under Y and Y is subsumed under X, PolicyLint uses the relation that has a higher frequency (i.e., appearing in more privacy policies). Once a fixed point is reached, PolicyLint ensures that there is only one root node by creating connections between any nodes that do not contain inward-edges with the root of the ontology (i.e., “information” for the data ontology, and “public” for the entity ontology). Finally, PolicyLint ensures no cycles exist in the ontology by identifying simple cycles in the graph and removing an edge between nodes to break the cycle. PolicyLint chooses which edge to remove by finding the edge that appears least frequently in the subsumptive relations and ensures that the destination node has more than one in-edge to ensure that a new root node is not created.



**Figure 4.2** Transformation of Example 2 from its dependency-based parse tree to its DED tree.

### 4.3.2 Policy Statement Extraction

The goal of policy statement extraction is to extract a concise representation of a policy statement to allow for automated reasoning over such statements. We represent data sharing and collection statements as a tuple (*actor*, *action*, *data object*, *entity*) where the *actor* performs some *action* (i.e., share, collect, not share, not collect) on the *data object*, and the *entity* represents the entity receiving the data object. For example, the statement, “We will share your personal information with advertisers,” can be represented by the tuple of (we, share, personal information, advertisers). PolicyLint extracts complete policy statements from privacy policy text by using patterns of the grammatical structures between data objects, entities, and verbs that represent sharing or collection (for brevity we call these SoC verbs). This section describes these steps.

#### 4.3.2.1 DED Tree Construction

The goal of constructing the data and entity dependency (DED) trees is to extract a concise representation of the grammatical relations between the data objects, entities, and the verbs that represent sharing or collection (SoC verbs). The main intuition behind constructing these trees is to allow PolicyLint to infer semantics of the sentence based on the grammatical relations between the tokens (i.e., *who* collects/shares *what* with *whom*). The DED tree for a sentence is derived from the sentence’s dependency-based parse tree. However, the DED tree removes nodes and paths that are not relevant to the data objects, entities, and SoC verbs, and performs a set of simplifications to generalize the representation. The transformation for Example 2 is shown in Figure 4.2.

**Example 2.** *If you register for our cloud-based services, we will collect your email address.*

To construct DED trees, PolicyLint parses a sentence and using its custom-trained NER model to label data objects and entities within the sentence (Section 4.3.1). PolicyLint

**Table 4.4** SoC verbs used by PolicyLint

Type	Word
Sharing	disclose, distribute, exchange, give, provide, rent, report, sell, send, share, trade, transfer, transmit
Collection	access, check, collect, gather, know, obtain, receive, save, store, use

merges noun phrases and iterates over sentence tokens to label SoC verbs by ensuring the PoS (part-of-speech) tag of the token is a verb and the lemma of the verb is in PolicyLint’s manually curated list of terms (Table 4.4). PolicyLint also labels the pronouns, “we,” “I,” “you,” “me,” and “us,” as entities during this step. PolicyLint then extracts the sentence’s dependency-based parse tree whose nodes are labeled with the data object, entity, and SoC verb labels as discussed above.

**Negated Verbs:** PolicyLint identifies negated verbs by checking for negation modifiers in the dependency-based parse tree. If the verb is negated, PolicyLint labels the node as negative. PolicyLint propagates the negation to descendant verb nodes in three situations. First, if a descendant verb is part of a conjunctive verb phrase with the negated verb, the negation is propagated. For example, “*We do not sell, rent, or trade your personal information,*” means “not sell,” “not rent,” and “not trade.” Second, if the descendant verb has an open clausal complement to the negated verb, the negation is propagated. For example, “*We do not require you to disclose any personal information,*” initially has “require” marked as negative. Since “disclose” is an open clausal complement to “require,” it is marked as negative. Third, if the descendant verb is an adverbial clause modifier to the negated verb, the negation is propagated. For example, “*We do not collect your information to share with advertisers,*” initially has “collect” marked as negative. Since “share” is an adverbial clause modifier to “collect,” it is marked as negative.

**Exception Clauses:** PolicyLint identifies exception clauses by traversing the parse tree and finding terms that represent exceptions to a prior statement: such as “except,” “unless,” “aside/apart from,” “with the exception of,” “besides,” “without,” and “not including.” For each identified exception clause, PolicyLint traverses down the parse tree from the exception clause to identify verb phrases (subject-verb-object) and noun phrases related to that exception. PolicyLint then traverses upward from the exception term to identify the nearest verb node and appends as a node attribute the list of noun phrases and verb phrases

identified in the downward traversal.

In certain cases, the term may not have a subtree. For example, the exception term may be a *marker* that introduces a subordinate clause. In the sentence, “*We will not share your personal information unless consent is given,*” the term “unless” is a marker that introduces the subordinate clause “your consent is given.” For empty sub-trees, PolicyLint attempts the downward traversal from its parent node.

**DED Tree construction:** Finally, PolicyLint constructs the DED tree by computing the paths between labeled nodes on the dependency-based parse tree, copying labels and attributes described above. Note that PolicyLint also copies over all unlabeled subjects and direct objects from the parse tree, as they are needed to extract the information. PolicyLint further simplifies the tree by merging conjuncts of SoC verbs into one node if the coordinating conjunction is “and” or “or.” For example, “*We will not sell, rent, or trade your personal information.*” can be simplified by collapsing “sell,” “rent,” and “trade” into one node. The resulting node’s label is a union of all of the tags of the merged verbs (i.e., {share} + {collect} = {share, collect}). Similarly, PolicyLint repeats the same process for conjuncts of data objects and entities.

PolicyLint then prunes the DED tree by iterating through the nodes labeled as verbs in the graph and performing the following process. First, for a verb node labeled as an SoC verb, PolicyLint ensures that its sub-tree contains at least one other node labeled as an SoC verb, data object, or entity. If the node’s sub-tree does not meet this condition, PolicyLint removes the subtree rooted at the node labeled as an SoC verb. Second, for verb nodes not labeled as SoC verbs, PolicyLint ensures that at least one SoC verb is contained in its sub-tree and that it meets the conditions above for an SoC verb. Similarly, if these conditions are not met, PolicyLint also removes the sub-tree rooted at that non-labeled verb node. For example, this pruning step causes the sub-tree rooted at the verb “register” to be removed in Figure 4.2.

#### 4.3.2.2 SoC Sentence Identification

To identify sentences that describe sharing and collection practices, PolicyLint takes a set of positive examples of sentences as input and then extracts their DED trees to use as known patterns for sharing and collection phrases. To begin, we feed PolicyLint a set of 560 example sentences that describe sharing and collect practices. PolicyLint generates the DED trees from these sentences and learns 82 unique patterns.

## Training Sentence Generation:

**Table 4.5** Sentence Generation Templates

1	<i>ENT may VERB_PRESENT DATA</i>	We may share your personal information.
2	<i>We may VERB_PRESENT DATA PREP ENT</i>	We may share your personal information with advertisers.
3	<i>We may VERB_PRESENT ENT DATA</i>	We may send advertisers your personal information.
4	<i>We may VERB_PRESENT PREP ENT DATA</i>	We may share with advertisers your personal information.
5	<i>DATA may be VERB_PAST PREP ENT</i>	Personal information may be shared with advertisers.
6	<i>DATA may be VERB_PAST</i>	Personal information may be shared.
7	<i>DATA may be VERB_PAST by ENT</i>	Personal information may be shared by advertisers.
8	<i>We may choose to VERB_PRESENT DATA</i>	We may choose to share personal information
9	<i>We may choose to VERB_PRESENT DATA PREP ENT</i>	We may choose to share personal information with advertisers.
10	<i>You may be required by us to VERB_PRESENT DATA</i>	You may be required by us to share personal information.
11	<i>You may be required by us to VERB_PRESENT DATA PREP ENT</i>	You may be required by us to share personal information with advertisers.
12	<i>We are requiring you to VERB_PRESENT DATA</i>	We are requiring you to share personal information.
13	<i>We are requiring you to VERB_PRESENT DATA PREP ENT</i>	We are requiring you to share personal information with advertisers.
14	<i>We require VERB_PRESENT_PARTIC DATA</i>	We require sharing personal information
15	<i>We require VERB_PRESENT_PARTIC DATA PREP ENT</i>	We require sharing personal information with advertisers.
16	<i>We may VERB_PRESENT ENT with DATA</i>	We may provide advertisers with your personal information.

PolicyLint requires a training set of sharing and collection sentences to learn underlying patterns from in order to identify “unseen” sharing and collection sentences. As manually selecting a set of sharing and collection sentences with diverse grammatical structures is tedious, we opted to auto-generate the training sentences instead. Note that auto-generating sentences does not adversely impact the extensibility of PolicyLint, as adding a new pattern is as simple as feeding PolicyLint the new sentence. To identify the templates, we used our domain expertise to identify different sentence compositions that could describe sharing and collection sentences. We identified 16 sentence templates, as shown in Table 4.5.

To fill the templates, we substitute an entity (*ENT*), data object (*DATA*), the correct tense of an SoC verb (*VERB\_PRESENT*, *VERB\_PAST*, *VERB\_PRESENT\_PARTICIPLE*), and a preposition that describes with *whom* the sharing occurs for sharing verbs (*PREP*). We began by identifying the present tense, past tense, and present participle forms of all of the SoC verbs (e.g., “share,” “shared,” “sharing,” respectively). We then identified common prepositions for each of the sharing verbs that describe with *whom* the sharing occurs. For example, for the terms *share*, *trade*, and *exchange* the preposition is “with” and for the terms *sell*, *transfer*, *distribute*, *disclose*, *rent*, *report*, *transmit*, *send*, *give*, *provide* the preposition is “to.”

For each template, we fill the placeholders accordingly. If the template has a placeholder for prepositions (*PREP*) and the verb is a collect verb, we skip the template. We also skip templates for “send,” “give,” and “provide” if it does not contain a placeholder for a preposition (T1, T6, T8, T10, T12, T14), as those sentences did not make sense without specifying to *whom* the data is being sent/given/provided. We set *DATA* to the phrases “your personal information” and “your personal information, demographic information, and financial information.” Similarly, we set *ENT* to the phrases “advertiser” and “advertisers, analytics providers, and our business partners.” Note that we included conjuncts of the *DATA* and *ENT* placeholders to account for deviations in the parse tree due to syntactic ambiguity (i.e., a sentence can have multiple interpretations). Therefore, we generate two sentences for each template: one with a singular *DATA* and *ENT*, and second with the plural *DATA* and plural *ENT*. In total, we generate 560 sentences, which are used by PolicyLint to learn patterns to identify sharing and collection sentences.

**SoC Sentence Classification:** PolicyLint iterates sentences of a given privacy policy. If the sentence contains at least one SoC verb and data object (labeled by NER), PolicyLint constructs the DED tree. PolicyLint then compares the sentence DED tree to the DED trees of the known patterns. A pattern is matched if (1) the sentence DED tree’s label types are equivalent to the pattern DED tree (e.g., {*entity*, *SoC\_verb*, *data*}), and (2) the known pattern DED tree is a subtree of the sentence DED tree.

For a tree  $t_1$  to be a subtree of tree  $t_2$ , (1) the tree structure must be equivalent, (2) the dependency labels on edges between nodes must match, and (3) the following three node conditions must hold. First, for SoC verb nodes to match, they must have a common lemma. For example, a node with the lemmas {*sell*, *rent*} matches a node with lemma {*rent*}. Second, if the node’s part-of-speech is an apposition, the tags, dependency label, and lemmas must

be equal. Third, for all other nodes, the tags and dependencies must be equal.

On sub-tree match, PolicyLint records the nodes in the sub-tree match and continues the process until either (1) each pattern is checked, or (2) the entire DED tree has been covered by prior sub-tree matches. If at least one sub-tree match is found, PolicyLint identifies the sentence as a potential SoC sentence and begins extracting the policy statement tuple.

#### 4.3.2.3 Policy Extraction

The goal of the policy extraction phase is to transform the DED tree into a (*actor, action, data object, entity*) tuple for a policy statement. PolicyLint performs extraction starting with the SoC nodes present in the sub-tree matches. If multiple SoC nodes exist in the sub-tree matches, multiple tuples are generated. However, multiple sub-tree matches over the same SoC node will only result in the generation of one tuple. The SoC determines the action (e.g., collect, not\_collect). The meaning of the action is determined based on whether the node is labeled as positive or negative (e.g., collect vs. not collect), as discussed in Section 4.3.2.1.

**Actor Extraction:** To extract the actor, PolicyLint starts from the matching SoC verb node. The actor is a labeled entity chosen from the (1) subject, (2) prepositional object, or (3) direct object (in that order). However, if the dependency is an open clausal complement or adverbial clause modifier, PolicyLint prioritizes the direct object and prepositional object over the subject. For example, “*We do not require you to disclose any personal information.*” has “disclose” as an open clausal complement to “require.” In this case, the correct actor of this policy statement is the user (i.e., “you”) rather than the vendor (i.e., “we”), which is correctly captured due to PolicyLint dependency-based prioritization rules. If no match is found, PolicyLint traverses up one level in the DED tree and repeats. Finally, if no match is found, PolicyLint assumes that the actor is the implicit first-party.

**Data Object Extraction:** To extract the data objects, PolicyLint starts from the matching SoC verb node. It traverses down the DED tree to extract all nodes labeled as data objects. The traversal continues until another SoC verb is reached. If no data objects are found, and the verb’s subject and direct object are not labeled as a data object, PolicyLint extracts the data objects from the nearest ancestor SoC verb.

**Entity Extraction:** To extract entities, PolicyLint starts from the matching SoC verb node. It traverses down the DED tree extracting all nodes labeled as entities that are not actors. The traversal continues until another SoC verb is reached.

**Exception Clauses:** PolicyLint considers exception clauses if the verb is marked as negative

(e.g., not collect, not share), creating a cloned policy statement with the meaning flipped (e.g., collect, share). We do not handle exception clauses for positive statements. For example, “*We might also share personal information without your consent to carry out your own requests*” still shares personal information.

For negated verbs, there are three cases. First, if the exception clause’s node attribute contains only data objects, PolicyLint replaces the data objects of the new policy with the data objects under the exception clause. For example, “*We will not collect your personal information except for your name and phone number.*” produces policies: (we, not\_collect, personal information, NULL), (we, collect, [name, phone number], NULL). Second, if all noun phrases have an entity label, PolicyLint replaces the entities of the new policies with the entities under the exception attribute. For example, “*We do not share your demographics with advertisers except for AdMob.*” produces policies: (we, not\_share, demographics, advertisers) and (we, share, demographics, AdMob). Third, if the labels are not data objects or entities, PolicyLint removes the initial policy statement. For example, “*We will not collect your personal information without your consent.*” produces the policy: (we, collect, personal information, NULL).

**Policy Expansion:** PolicyLint may extract multiple actors, actions, data objects, and entities when creating policy statements. These complex tuples are expanded. For example, ([we], [share, sell], [location, age], [Google, Facebook]) expands to (we, share, location, Google), (we, share, location, Facebook), (we, share, age, Google), etc.

### 4.3.3 Policy Contradictions

PolicyLint’s components of ontology generation and policy extraction identify the sharing and collection statements in privacy policies. This section formally defines a logic for characterizing different contradictions. It then describes how PolicyLint uses this logic to identify candidate contradictions within privacy policies. We note that contradictions may occur between an application’s privacy policy and the privacy policies of third-party libraries (e.g., advertisement libraries). While our study focuses specifically on contradictions within an individual privacy policy, the formal logic and subsequent analysis tools may also be used to include the privacy policies for third-party libraries with minimal modification.

**Table 4.6** Rules that transform a CPS into an SPS

Rule	Transformation Rules	Rationale
T1	(actor, share, data object, entity) $\implies$ (actor, collect, data object)	Unknown whether sharing occurs at the client side or server side
T2	(actor, share, data object, entity) $\implies$ (entity, collect, data object)	Can observe only client-side behaviors
T3	(actor, not_share, data object, entity) $\implies$ (entity, not_collect, data object)	Can observe only client-side behaviors
T4	(actor, not_share, data object, entity) $\implies$ (actor, collect, data object)	If mention not share, assume implicit collection

### 4.3.3.1 Policy Simplification

PolicyLint simplifies policy statements for its contradiction analysis. We refer to the *(actor, action, data object, entity)* tuple defined in Section 4.3.2 as a Complete Policy Statement (CPS). We simplify CPS statements that discuss sharing data (i.e., *action* is share or not share) by capturing sharing as collection.

**Definition 1** (Simplified Policy Statement). *An SPS is a tuple,  $p = (e, c, d)$ , where  $d$  is the data object discussed by the statement,  $c \in \{collect, not\_collect\}$  represents whether the object is collected or not collected, and  $e$  is the entity receiving the data object.*

To transform a CPS into an SPS, we leverage three main insights. First, policies do not typically disclose whether the sharing of the data occurs at the client side or server side. Therefore, an actor sharing a data object with an entity may imply that the actor is collecting the data and performing the data sharing at the server side. In this case, a new policy statement would need to be generated for allowing the actor to collect the data object (Rule T1, Table 4.6). Second, a data object being shared with an entity may imply that the entity is collecting the information from the mobile device (Rule T2, Table 4.6). Similarly, a policy for stating that the actor does not share a data object with an entity implies that the entity is not collecting the data from the mobile device (Rule T3, Table 4.6). Finally, a policy for stating that the actor does not share a data object implies that the actor collects the data object, because the policy would likely have not mentioned not sharing data that was never collected (Rule T4, Table 4.6).

However, there are two special cases. First, PolicyLint only treats verb lemmas “save” and “store” as positive (“not saving/storing” does not mean “not collecting”). Second, PolicyLint ignores negative statements with verb lemma “use.” This case leads to false

positives, as PolicyLint does not extract the collection purpose. For example, “*We do not use your location for advertising.*” means that your location is not collected for the specific purpose of advertising.

#### 4.3.3.2 Contradiction Types

We model an application’s privacy policy as a set of simplified policy statements  $P$ . Let  $D$  represent the total set of data objects and  $E$  represent the total set of entities, as represented by ontologies for data objects and entities, respectively. A policy statement  $p \in P$  is a tuple,  $p = (e, c, d)$  where  $d \in D$ ,  $e \in E$ , and  $c \in \{collect, not\_collect\}$  (Definition 1).

Language describing policy statements may use different semantic granularities. One policy statement may speak in generalizations over data objects and entities while another statement may discuss specific types. For example, consider the policies  $p_1 = (\text{advertiser}, \text{not\_collect}, \text{demographics})$  and  $p_2 = (\text{Google Admob}, \text{collect}, \text{age})$ . If we want to identify contradictions, we need to know that Google AdMob is an advertiser and age is demographic information. These relationships are commonly referred to as *subsumptive relationships* where a more specific term is subsumed under a more general term (e.g., AdMob is subsumed under advertisers and age is subsumed under demographics).

We use the following notation to describe binary relationships between terms representing data objects and entities.

**Definition 2** (Semantic Equivalence). *Let  $x$  and  $y$  be terms partially ordered by an ontology  $o$ .  $x \equiv_o y$  is true if  $x$  and  $y$  are synonyms, defined with respect to an ontology  $o$ .*

**Definition 3** (Subsumptive Relationship). *Let  $x$  and  $y$  be terms partially ordered by “is-a” relationships in an ontology  $o$ .  $x \sqsubset_o y$  is true if term  $x$  is subsumed under the term  $y$  such that  $x \not\equiv_o y$ . Similarly,  $x \sqsubseteq_o y \implies x \sqsubset_o y \vee x \equiv_o y$ .*

Note that Definitions 2-3 parameterize the operators with an ontology  $o$ . PolicyLint operates on two ontologies: data objects and entities. Therefore, the following discussion parameterizes the operators with  $\delta$  for the data object ontology and  $\varepsilon$  for the entity ontology. For example,  $x \equiv_\delta y$  and  $x \equiv_\varepsilon y$ .

A contradiction occurs if two policy statements suggest that entities both may and may not collect or share a data object. Contradictions can occur at the same or different semantic levels. For example, the simplest form of contradictions is an exact contradiction where a policy states that an entity will both collect and not collect the same data object, e.g.,

**Table 4.7** Contradictions (C) and Narrowing Definitions (N)

Rule	Logic	Example
$C_1$	$e_i \equiv_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(Flurry, collect, IMEI) (Flurry, not_collect, IMEI)
$C_2$	$e_i \equiv_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(Flurry, collect, IMEI) (Flurry, not_collect, Device Info)
$C_3$	$e_i \sqsubset_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(Flurry, collect, IMEI) (Advertiser, not_collect, IMEI)
$C_4$	$e_i \sqsubset_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(Flurry, collect, IMEI) (Advertiser, not_collect, Device Info)
$C_5$	$e_i \supset_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(Advertiser, collect, IMEI) (Flurry, not_collect, Device Info)
$N_1$	$e_i \equiv_\varepsilon e_j \wedge d_k \supset_\delta d_l$	(Flurry, collect, Device Info) (Flurry, not_collect, IMEI)
$N_2$	$e_i \sqsubset_\varepsilon e_j \wedge d_k \supset_\delta d_l$	(Flurry, collect, Device Info) (Advertiser, not_collect, IMEI)
$N_3$	$e_i \supset_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(Advertiser, collect, IMEI) (Flurry, not_collect, IMEI)
$N_4$	$e_i \supset_\varepsilon e_j \wedge d_k \supset_\delta d_l$	(Advertiser, collect, Device Info) (Flurry, not_collect, IMEI)

$$*P = \{(e_i, \text{collect}, d_k), (e_j, \text{not\_collect}, d_l)\}$$

(advertiser, collect, age) and (advertiser, not\_collect, age). Due to subsumptive relationships (Definitions 3), there are 3 relationships between terms ( $x \equiv_o y$ ,  $x \sqsubset_o y$ , and  $x \supset_o y$ ). Each binary relation applies to both entities and data objects. Therefore, there are  $3^2 = 9$  types of contradictions, as shown in Table 4.7.

Contradictions have two primary impacts when analyzing privacy policies. First, all contradictions impact analysis techniques that seek to automatically reason over policies and may result in these techniques making incorrect or inconsistent decisions. For example, unlike firewall rules that have a specific evaluation sequence, privacy policy statements do not have a specific pre-defined sequence of evaluation. Therefore, analysis techniques may make incorrect or inconsistent decisions based on the order in which they evaluate policy statements. Second, contradictions may impact a human analyst's understanding of a privacy policy, such as by containing misleading statements. We define two groupings of contradictions based on whether they may impact a human's comprehension of privacy policies or solely impact automated analysis.

**Logical Contradictions ( $C_{1-5}$ ):** Logical contradictions are contradictory policy statements that are more likely to cause harm if users and analysts are not aware of the contradictory

statements. Logical contradictions may cause difficulties when humans attempt to comprehend or interpret the sharing and collection practices discussed in the policy. They can be characterized as either exact contradictions ( $C_1$ ) or those that discuss *not* collecting broad types of data and later discuss collecting exact or more specific types ( $C_{2-5}$ ). One example is a policy that initially claims to not collect personal information, but later in fine print discloses collecting a user’s name and email address for advertisers. Similar to the goal of software `lint` tools, PolicyLint flags logical contradictions as problems within policies to allow a human analyst to manually inspect the statements and analyze intent. As shown in Section 4.4, these contradictions may lead to the identification of intentionally deceptive policy statements or those that may result in ambiguous interpretations.

**Narrowing Definitions ( $N_{1-4}$ ):** Narrowing definitions are contradictory policy statements where broad information is stated to be collected, and specific data types are stated to not be collected. These statements narrow the scope of the types of data that are collected, such as stating that personal information is collecting while your name is *not* collected. Note that narrowing definitions are not necessarily be an undesirable property of policies, as saying a broad data type is collected does not necessarily imply that every specific subtype is collected, but they may result in vague policies. There may be clearer ways for policy writers to convey this information, such as explicitly stating the exact data types collected and shared. For example, if the app collects your email address, the policy could directly state, “We collect your email address,” instead of including a narrowing definition, such as “We collect personal information. We do not collect your name.” However, policies that narrow the scope of their data sharing and collection practices can be seen as more desirable in contrast to policies that just disclose practices over broad categories of data. Nonetheless, narrowing definitions impact the logic behind analysis techniques, as they must consider prioritization of data objects and entities.

#### 4.3.3.3 Contradiction Identification

PolicyLint uses the contradiction types from Table 4.7 to determine a set of candidate contradictions. It then uses a set of heuristics to reduce the set of candidate contradictions that are potentially low-quality indicators of underlying problems. Next, PolicyLint prepares the contradictions for presentation by collapsing duplicate contradictions, linking other metadata (e.g., download counts of applications), and by using a set of filtering heuristics to allow the regulator or privacy analysts to focus on specific subclasses of candidate

**Table 4.8** First-Party Synonyms

First-Party Synonyms
we, I, us, me, our app, our mobile application, our mobile app, our application, our service, our website, our web site, our site
app, mobile application, mobile app, application, service, company, business, web site, website, site

contradictions. The remainder of this section describes this process.

**Initial Candidate Set Selection:** Given policy statements  $p_1$  and  $p_2$ , PolicyLint ensures that  $p_1.c$  does not equal  $p_2.c$ , as contradictions require opposing signs (positive/negative). PolicyLint then compares entities  $p_1.e$  and  $p_2.e$ , determining if they are equal or have a subsumptive relationship. A subsumptive relationship occurs if there is a path between the entities in the entity ontology. When comparing entities, PolicyLint treats the terms in Table 4.8 as synonyms for the first-party (i.e., “we”). If an entity match is found, PolicyLint then performs the same steps for data objects  $p_1.d$  and  $p_2.d$  using the data object ontology. If a data object match is found, PolicyLint adds the candidate contradiction to the candidate set. Note that PolicyLint ignores entities and data objects in policy statements that are not contained in ontologies, as it cannot reason about those relations. However, if PolicyLint cannot find a direct match for a term in the ontology, it will try to find sub-matches by splitting the term on the coordinating conjunction terms (e.g., “and,” “or”) and checking for their existence in the ontologies. Further, PolicyLint does not identify policy statements as contradictions if they are generated from the same sentence due to the semantics of exception clauses. For example, “*We do not collect your personal information except for your name.*” produces simplified policy statements (we, not\_collect, personal information) and (we, collect, name). While the semantics of this statement is clear, our definition of contradictions would incorrectly identify these statements as a  $C_2$  contradiction. Therefore, PolicyLint ignores same sentence contradictions to reduce the risk false positives.

**Candidate Set Reduction:** PolicyLint uses heuristics to prune candidate contradictions that are likely low-quality indicators of underlying problems. PolicyLint removes contradictions that occur based on potentially poor relations discovered in the ontologies. For example, PolicyLint filters out contradictions that occur between certain data object pairs, such as “usage information” and “personal information.” Contradictions whose entities refer to the user (e.g., “user,” “customer,” “child”) or involve terms for general data objects (e.g., “information,” “content,” “material”) or entities (e.g., “individual,” “public”) are also removed. Finally, PolicyLint removes candidate contradictions where the negative policy

statement may be conditioned with age restrictions or based on user choice by searching for common phrases in the sentences that generated the policy statements (e.g., “under the age of,” “from children,” “you do not need to provide”). Note that some of these reductions may occur during candidate set construction to reduce complexity of the analysis.

**Candidate Set Filtering:** PolicyLint further filters the set of candidate contradictions into subsets based on the data objects involved in the contradiction to allow for targeted exploration during verification. For example, all of the contradictions with statements involving collecting email address but not collecting personal information are placed into one subset (e.g., (\*, collect, email address) and (\*, not\_collect, PII)).

**Contradiction Validation:** Given the filtered subsets of candidate contradictions, the next step is to explore certain subsets and validate candidate contradictions. To validate a candidate contradiction, the analyst reads through the policy statements and sentences that generated them in context of the entire policy and makes a decision.

## 4.4 Privacy Study

Our primary motivation for creating PolicyLint was to analyze contradicting policy statements within privacy policies. In this section, we use PolicyLint to perform a large-scale study on 11,430 privacy policies from top Android applications.

**Dataset Collection:** To select our dataset, we scraped Google Play for the privacy policy links for up to the top 500 free applications across Google Play’s 35 application categories in September 2017. Note that the “top free applications” are based on the ranking provided by Google Play (i.e., “topselling\_free” collection per application category). We downloaded the HTML privacy policies using the Selenium WebDriver in a headless Google Chrome browser to allow for the execution of dynamic content (e.g., JavaScript). We excluded apps that did not have a privacy policy link on Google Play, those whose pages were unreachable at the time of collection, and privacy policies where the majority of the document was not written in English, as discussed in Appendix 4.3. We converted the HTML policies to plaintext documents. Our final dataset consists of 11,430 privacy policies.

### 4.4.1 General Policy Characteristics

PolicyLint extracted sharing and collection policy statements from 91% of the policies in our dataset (10,397/11,430). From those policies, PolicyLint extracted 438,667 policy statements

from 177,169 sentences that PolicyLint identified as a sharing or collection sentence. Of those policy statements, 32,876 had a negative verb and 405,789 had a positive verb. In particular, 60.5% (6,912/11,430) of policies had at least one negative policy statement and 89.6% (10,239/11,430) of policies had at least one positive policy statement.

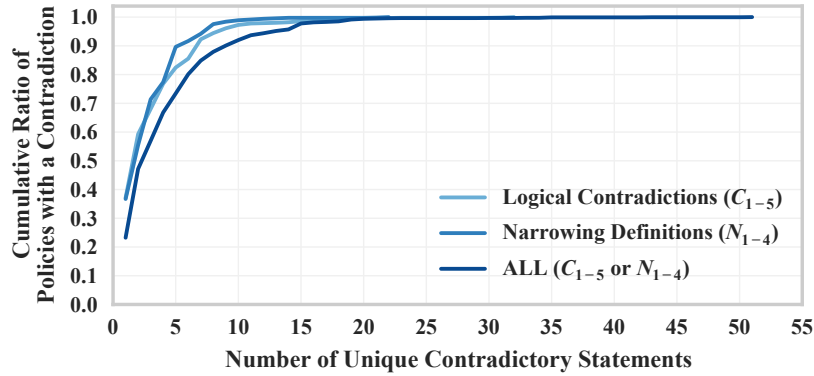
We explored why PolicyLint did not extract policy statements for 9.0% of policies by analyzing a random subset of 100 policies and found that 88.0% (88/100) were due to an insufficient crawling strategy (i.e., privacy policy links pointed at home pages, navigation pages, or 404 error pages). Three links pointed at a PDF privacy policy while PolicyLint only handles HTML. Two policies were not written in English and were not caught by our language classifier. Finally, 7 policies were due to errors extracting policy statements, such as incomplete verbs lists (3), tabular format policies (1), and policies that describing permission uses (3).

**Finding 1:** *Policies frequently contain negative policy statements that discuss broad categories of data.* For the 60.5% of policies with at least one negative policy statements, the data object “personal information” appeared in 67.7% of those policies (4,681/6,912). This demonstrates the importance of handling negative policy statements, as around 41.0% of the policies contain a negative policy statement that claims that a broad type of data (i.e., “personal information”) is not collected. Further, we measured the distance from the negation (i.e., “not”) to the verb that they modify and found that the 28.2% (3,234/11,430) of policies have a distance greater than one word away. This calls into question prior work [Sla16; Wan18] that only assume positive statements when considering sharing and collection statements, as they could be incorrect up to 60.5% of the time when reasoning over sharing and collection statements. Further, approaches that handle negations using bigrams [Zim17b] would have failed to reason about 28.2% of the policies.

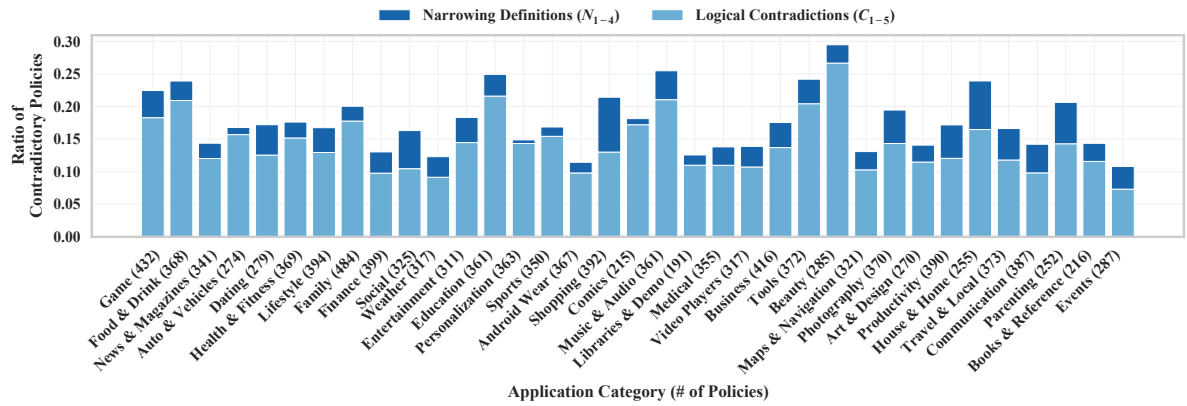
#### 4.4.2 Candidate Contradictions

Based on PolicyLint’s fine-grained policy statement extraction, we found that 59.1% (6,754 / 11,430) of the policies were candidates for contradiction analysis, as they contain at least one positive and one negative policy statement. There were 13,871 and 129,575 policy statements among these that were negative and positive, respectively.

**Finding 2:** *For candidate contradictions, 14.2% of privacy policies contain logical contradictions ( $C_{1-5}$ ).* PolicyLint identified 9,906 logical contradictions across around 14.2% (1,618/11,430) of policies. Therefore, around 14.2% of policies may contain potentially



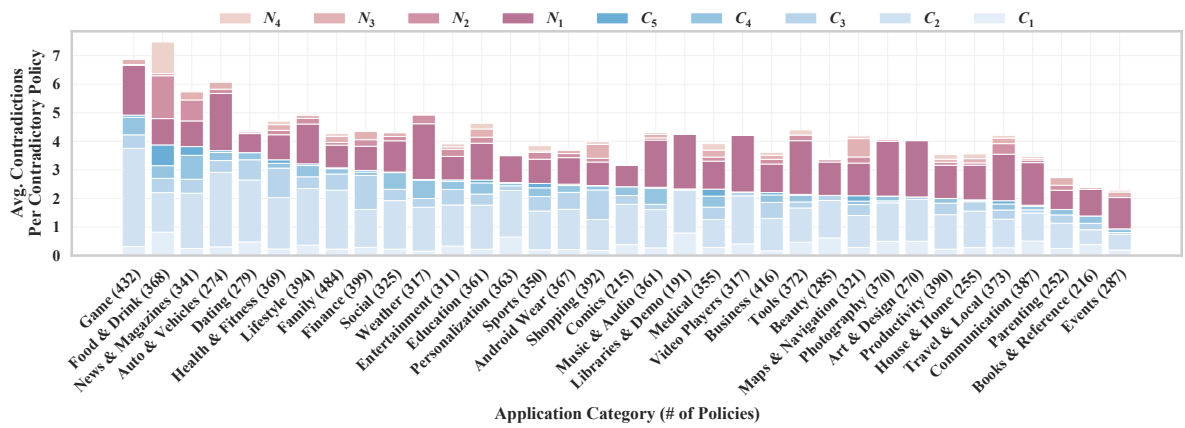
**Figure 4.3** CDF of Contradictory Policy Statements: 50% of contradictory policies have 2 or fewer logical contradictions.



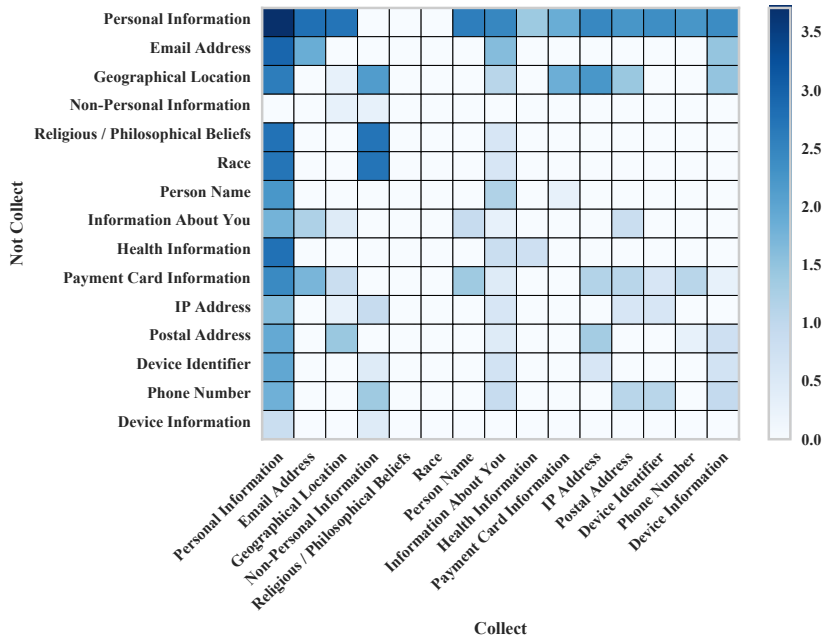
**Figure 4.4** Ratio of Contradictory Policies per Category: 79.7% of contradictory policies have at least one or more logical contradictions ( $C_{1-5}$ ) that may indicate potentially deceptive statements.

misleading statements. Figure 4.3 shows around three-fifths (59.2%) of policies with at least one logical contradiction have 2 or fewer unique contradictions. The relatively low number of candidate contradictions per policy indicates that manual validation is feasible. As roughly 6 in 7 policies are not contradictory, writing policies without logical contradictions is possible.

**Finding 3:** *Contradiction prevalence and frequency does not substantially vary across Google Play app categories.* Figure 4.5 shows the ratio of policies containing candidate contradictions per each Google Play category. The categories with policies most and least prone to contradiction are *Beauty* and *Events*, respectively. However, when analyzing the policies



**Figure 4.5** Average number of unique candidate contradictions per category: Logical contradictions ( $C_{1-5}$ ) and narrowing definitions ( $N_{1-4}$ ) are both widely prevalent across Google Play categories.



**Figure 4.6** Log 10 Frequency of Data Type Pairs in Contradictions: Negative statements that discuss broad categories of data are problematic (i.e.,  $C_{1-5}$ ).

within those categories, we found that their means were skewed by contradictory policies for applications by the same developer. When recomputing the average without the outliers, these categories followed the general trend. Policies with logical contradictions accompany

7.3-to-20.9% of apps across all categories. We find that policies with logical contradictions are not substantially more prevalent in particular categories of apps, but instead occur consistently in apps from every category. We also found that logical contradiction prevalence does not substantially vary by download count as well.

Figure 4.4 displays the average number of candidate contradictions for policies containing one or more contradictions. We found that logical contradiction frequency for contradictory policies did not substantially vary across Google Play categories. Initial analysis indicated contradictory policies for apps in the *Games* category contain around 4.9 logical contradictions on average. Further analysis revealed this is due to policies with 19 unique logical contradictions in 9 apps produced by the same developer and one app that had 31 unique logical contradictions. Excluding the outliers bring the category's average to 3.16 logical contradictions per app, which fits the trend of the rest of the categories. This may indicate that poor policies are linked to problematic developers. Similar analysis on *Food & Drink*, *Auto & Vehicles*, and *News & Magazines* produced similar results. We find that the number of logical contradictions per policy is roughly equivalent across application categories, which means that one app category is not necessarily more contradictory on average than another.

**Finding 4:** *Negative policy statements that discuss broad categories of data are problematic.* Figure 4.6 shows the frequency of the most common data type pairs referred to in contradictory policy statements. The contradicting policy statements in the topmost row are most problematic. This row represents logical contradictions, which are either (1) exact contradictions or (2) discuss not collecting broad types of data and collecting more specific data types ( $C_{1-5}$ ). As we demonstrate in Section 4.4.3, they can lead to a myriad of problems when interpreting the policy including making interpretation ambiguous in certain cases. The leftmost column corresponds to narrowing definitions ( $N_{1-4}$ ), which solely impact automated analysis techniques, as discussed in Section 4.3.3.

**Finding 5:** *For candidate contradictions, 17.7% of privacy policies contain at least one or more logical contradictions ( $C_{1-5}$ ) or narrowing definitions ( $N_{1-4}$ ).* PolicyLint identified 17,986 logical contradictions and narrowing definitions across around 17.7% (2,028/11,430) of policies. Figure 4.3 shows slightly more than half (57.0%) of contradictory policies have 3 or fewer unique logical contradictions and narrowing definitions. As discussed in Section 4.3.3, logical contradictions and narrowing definitions impact approaches that seek to automatically reason over privacy policies. To correctly reason over contradictions, analysis

techniques must include logic to prioritize specificity of data types and entities and be able to identify potentially undecidable cases, such as exact contradictions ( $C_1$ ). No prior works that attempt to reason over policies [Sla16; Yu16; Zim17b; Wan18] operate at the required granularity to identify contradictions or contain the logic to correctly reason over them. Therefore, these approaches could make the incorrect or inconsistent decisions around 17.7% of the time around 1–3 times per policy on average. We performed similar analysis across categories as Finding 4.4.2 and found that logical contradictions and narrowing definitions do not substantially vary across Google Play categories.

### 4.4.3 Deeper Findings

In this section, we describe the findings from validating candidate contradictions. We limit our scope to logical contradictions ( $C_{1-5}$ ), as they may be indicative of misleading policy statements. Due to resource constraints, we did not validate all 9,906 candidate logical contradictions from the 1,618 policies. Instead, we narrow the scope of our study by choosing categories of candidate contradictions to focus on. Our selection and validation methodology are described below.

**Selection Methodology:** To select the categories of candidate contradictions to focus on, we analyzed the data objects involved in the contradictory statements by analyzing Figure 4.6. We limit our scope to logical contradictions ( $C_{1-5}$ ) that discuss not collecting “personal information” and collecting “email address,” “device identifier,” or “personal information.” We also explore two other categories of candidate contradictions in which one type of data can be derived from the other type, which caught our attention when analyzing the heat map. Within each category of candidate contradictions, we chose which specific candidate contradictions to validate by sorting the contradictions based on the app’s popularity and working down the list. We spent around one week validating contradictions where our cutoffs were due to time constraints and attempting to achieve coverage across categories.

**Validation Methodology:** To validate candidate contradiction, one-of-three authors reads through the sentences that generated each policy statement for the candidate contradiction to ensure correctness of policy statement extraction. If there was an error with policy statement extraction, we record the candidate contradiction as a false positive and stop analysis. Next, we locate the sentences within the policy and view the context in which they appear (i.e., section, surrounding sentences) to determine whether the policy statements are contradictory. We try to determine why the contradiction occurred if possible and

record any observations about the policy. If the author was uncertain about their decision, a second author analyzed it. The two authors discuss and resolve conflicts, with no conflicts left unresolved after discussion.

#### 4.4.3.1 Personal Information and Email Addresses

For candidate contradictions with negative statements about “personal information” and with positive statements about “email address,” we found 618 candidate contradictions across 333 policies ( $C_2$ ,  $C_4$ ,  $C_5$ ) We validated 204 candidate contradictions from 120 policies. We found 5 candidate contradictions were false positives due to inaccuracies labeling data objects by the NER model. From the 199 remaining candidate contradictions across 118 policies, we had the following main findings. Note that when considering the findings discussed below, the terms “personally identifiable information” and “personal information” are commonly used synonymously in USA regulations and “personal data” is considered the EU equivalent albeit covering a broader range of information.

**Finding 6:** *Policies are stating that certain types of common personally identifiable information, such as email addresses, as non-personally identifiable.* When validating 14 candidate contradictions, we found 14 policies that explicitly state that they do NOT consider email address as personally identifiable information. 11 of those policies were released by the same developer (OmniDroid) where the most popular app in the set (com.omniluxtrade.allrecipes) has over 1M+ downloads. OmniDroid’s policy explicitly lists email address when defining non-personally identifiable information. The remaining 3 policies belong to another app developer, PlayToddlers. The apps are explicitly targeted towards children from 2-8 years old and have between 500K-1M+ downloads for each app. Their policy states the following sentence verbatim, “When the user provides us with an email address to subscribe to the “PlayNews” mailing list, the user confirms that this address is not a personal data, nor does it contain any personal data.”

The fact that *any* privacy policies are declaring email addresses as non-personal information is surprising, as it goes against the norms of *what* data is considered personal information as defined by regulations (e.g., CalOPPA, GDPR), standards bureaus (NIST), and common sense.

**Finding 7:** *Services that auto-generate template-based policies for app developers are producing contradictory policies.* During our validation process, we noticed that many policies had similar structural compositions and contained a lot of the same language in para-

graphs. When validating 78 candidate contradictions, we found 59 contradictory policies that were automatically generated or used templates. Identical policy statements from various developers suggested that some policies may be generated automatically or acquired from a template. We investigated these cases and identified 59 policies that used 3 unique templates. We check that these were not policies for apps created by the developers or organization. Findings 4.4.3.1 and 4.4.3.3 discuss the problems caused by the templates. This demonstrates that poor policy generators can be a contributing factor for numerous contradictory policies.

**Finding 8:** *Policies use blanket statements affirming that personal information is not collected and contradict themselves by stating that subtypes of personal information are collected, such as email addresses.* When validating 182 candidate contradictions, we found 104 policies broadly make blanket statements that personal information is not collected in one part of the policy and then directly contradict their prior statements by disclosing that they collect email addresses. We found 69 of those policies (127 validated contradictions) state that they do not collect personal information, but later state that they collect email addresses for some purpose. Of those 69 policies, 32 policies define email address as personal information in one part of their policy. Due to the lack of definition of what they consider personal information in the other 37 policies, it is unclear whether they do not consider email address as personal information or are just contradictory.

20 of those policies that explicitly defined email addresses as personal information, but contradicted themselves, are by the same organization (emoji-keyboard.com). The most popular app in that group had 50M+ downloads (emoji.keyboard.emoticonkeyboard). The following two sentences were in the policy verbatim: (1) *“Since we do not collect Personal Information, we may not use your personal information in any way.”*; (2) *“For users that opt in to Emoji Keyboard Cloud, we will collect your email address, basic demographic information and information concerning the words and phrases that you use (“Language Modeling Data”) to enable services such as personalization, prediction synchronization and backup.”* This is clearly a contradictory statement and arguably a misleading practice.

A policy for a particular app with 1M+ downloads (com.picediting.haircolorchanger) appears to have been potentially trying to mislead users by using bold text to highlight desirable properties and then contradicting themselves. For example, the following excerpt was in the policy verbatim including the bold typography: ***“We do not collect any Personal information but it may be collected in a number of ways. We may collect certain information***

*that you voluntarily provide to us which may contain personal information. For example, we may collect your name, email address you provide us when you contact us by e-mail or use our services...*” The use of bold typography and general presentation of these policy statements could potentially be considered as attempting to deceive the reader, who may not perform a close read of the text in fine-print. This finding validates PolicyLint’s value in flagging problematic areas in policies to aid in the identification of deceptive statements.

**Finding 9:** *Policies consider hashed email addresses as pseudonymized non-personal information and share it with advertisers.* When validating three candidate contradictions, we found two policies discuss sharing hashed email addresses with third parties, such as advertisers. One candidate contradiction was a false positive due to misclassifying a sentence discussing opt-out choices as a sharing or collection sentence. The other policy belonged to an app named Tango (com.sgiggle.production). Tango is a messaging and video call app, which has over 100M+ downloads on Google Play and according to their website has 390M+ users globally. Their policy states the following sentences verbatim, *“For example, we may tell our advertisers the number of users our app receives or share anonymous identifiers (such as device advertising identifiers or hashed email addresses) with advertisers and business partners.”* Tango explicitly states that they consider hashed email addresses as anonymous identifiers. It is arguable whether hashing is sufficient for pseudonymization as defined by GDPR, as it is likely that advertisers are using hashed email addresses to identify individuals.

#### **4.4.3.2 Personal Information and Device Identifiers**

For the candidate contradictions with negative statements about “personal information” and with positive statements about “device identifiers,” we found 234 candidate contradictions across 155 policies. We investigated this group of candidate contradictions as there are differing regulations across countries on whether device identifiers are considered personal information. For example, various court cases within the US (Robinson v. Disney Online, Ellis v. Cartoon Network, Eichenberger v. ESPN) ruled that device identifiers are not personal information. However, the GDPR defines device identifiers as personal information. Therefore, our goal was to check whether policies were complying to the stricter GDPR definition of personal information or to the US definition, as this could hint towards problems with complying to regulations across country boundaries. In total, we validated 10 candidate contradictions across 9 policies.

**Finding 10:** *Policies are considering device identifiers as non-personal information, which raises concerns regarding globalization of their policies.* When validating 10 candidate contradictions, we found 9 policies that state that they do not collect personal information, but later state that they collect device identifiers. We find that classification of device identifiers varies across policies. We found 4 policies that explicitly describe device identifiers as non-personal information. The most popular app is Tango (com.sgiggle.production), which boasts of 390M+ global users on their website. It is likely a safe assumption that some of those users are in the EU, which is subject to GDPR. As their current policy still contains this statement, it may hint that they may not be GDPR compliant.

To reduce the threats to the validity of our claims, we re-requested the 9 policies using a proxy to route the traffic through an EU country (Germany) to ensure that an EU-specific policy was not served based on the origin of the request. We requested the English version of the policy where applicable and found similarly problematic statements in regard to not treating device identifiers as personal information.

#### **4.4.3.3 Personal Information and Personal Information**

We found 5100 candidate contradictions across 1061 policies where the data type of both the negative statement and positive statement is “personal information.” We validate 254 candidate contradictions across 153 policies.

**Finding 11:** *Policies directly contradict themselves.* When validating the 254 candidate contradictions, we found that the 153 policies directly contradicted themselves on their data practices on “personal information.” For example, the policy for an application with 1M+ downloads, states: “We may collect personal information from our users in order to provide you with a personalized, useful and efficient experience.” However, later in the policy they state, “We do not collect Personal Information, and we employ administrative, physical and electronic measures designed to protect your Non-Personal Information from unauthorized access and use.” These scenarios are clearly problematic, as the policies state both cases and it makes it difficult, if not, impossible to determine their actual data sharing and collection practices.

#### **4.4.3.4 Derived Data**

In this section, we explore cases of candidate contradictions where the negative statements discuss data that can be derived from the data discussed in the positive statement. In

particular, we explore two cases: (1) coarse location from IP address; and (2) postal address from precise location.

For “coarse location from IP,” we found 170 candidate contradictions from 167 policies that represented collecting IP address and not collecting location. We narrowed down the candidate contradictions by removing statements that discuss precise location, as IP address does not provide a precise location. This filtering resulted in 18 candidate contradictions from 18 different policies. We validated 15 candidate contradictions across 15 different policies for this case. We note that 3 candidate contradictions from 3 policies were false positives due to incorrect negation handling.

For “postal address from precise location,” we found 27 candidate contradictions across 20 policies. Note that we remove candidate contradictions that discuss coarse location, as they are not precise enough to derive postal addresses. We validated 22 candidate contradictions across 17 applications, as 5 candidate contradictions were false positives due to sentence misclassification (4) or errors handling negations (1).

**Finding 12:** *Policies state that they do not collect certain data types, but state that they collect other data types in which the original can be derived.* When validating the 15 candidate contradictions for “coarse location from IP,” we found that all 15 policies were stating that they do not collect location information, but state that they automatically collect IP addresses. As coarse location information can generally be derived from the user’s IP address, it can be argued that the organization is technically collecting the user’s location information. Interestingly, two of the policies discuss that if users disable location services, then location will not be collected. It is highly unlikely that companies cease IP address collection based on device privacy settings. However, as IP address collection typically occurs passively server-side, we cannot claim with 100% certainty that the companies still collect IP addresses when location services are disabled.

When validating 20 candidate contradictions for “postal address from precise location,” we found that 15 policies discussed not collecting postal addresses, but then state that they collect locations. Similar to the above case, postal addresses can be derived from location data (i.e., latitude and longitudes). Again, the argument can be made that they are collecting data precise enough to be considered a postal address, which causes a contradiction. For the other two candidate contradictions from two policies, it was not clear whether it was actually a contradiction, as they state that they do not collect addresses from the user’s address book, which is more specific than a general statement about not collecting addresses.

#### 4.4.4 Notification to Vendors

```
Subject: Contradictory Privacy Policy Statements in <APP_NAME>
To Whom It May Concern:
We are a team of security researchers from the WSPR Lab in the Department of Computer Science at North Carolina State University. We created a tool to analyze privacy policies and found that the privacy policy for your "<APP_NAME>" application (<PACKAGE>) that we downloaded in September 2017 may contain the following potential contradictory statements:
(1) The following statements claim that personal information is both collected and not collected.
(A) <CONTRADICTIONARY_SENTENCE_1>
(B) <CONTRADICTIONARY_SENTENCE_2>
...
Do you believe that these are contradictory statements? Any additional information that you may have to clarify this policy would be extremely helpful. If you have any questions or concerns, please feel free to contact us, preferably by February 11th.
Thank you for your time!
Best Regards,
<EMAIL_SIGNATURE>
```

**Figure 4.7** Email Template

For the 510 contradictions that were validated across 260 policies, we contacted each vendor via the email address listed for the privacy policy’s corresponding app on Google Play. We disclosed the exact statements that we found to be contradictory and explained our rationale. We asked whether they consider the statements to be contradictory and requested clarifications on their policy. Figure 4.7 shows a template of the email. Overall, 244 emails were successfully delivered, as 16 email addresses were either invalid or unreachable. In total, we received a 4.5% response rate (11/244), which is relatively substantial when considering that responding to our emails could raise liability concerns. All of the responses were received within a week or less after sending the initial emails. We waited an additional two weeks but did not receive additional responses before submission. The remainder of this section discusses the responses.

**Fixed Policy:** Three vendors agreed with our findings and updated their policy to remove the contradiction. One vendor stated that there was an “error” in their privacy policy and updated it accordingly. We confirmed that the policy was updated. The remaining two vendors stated that the self-contradictory portion of the policy was a leftover remnant from a prior update and should have been removed. They clarified that they do not collect email addresses.

**Disagreed with our Findings:** One vendor explicitly disagreed with our findings. Their policy states that they do not collect personal information, but also states that they collect email addresses. The vendor responded by claiming that email addresses are only personal information if it contains identifiable information, such as your name (e.g., john.doe@gmail.com), but are not personal information if it does not contain identifiable information (e.g., wxyz@gmail.com). They state that they explicitly tell users to not submit email addresses that contain personal information and thus their policy is not contradictory. This interpretation is surprising, because it goes against the definition of email addresses as personal information, as defined by regulations (e.g., CalOPPA, GDPR) and standards bureaus (NIST).

**Claimed Outdated Policy:** Four vendors responded that they did not find that language in their current policy and that we analyzed an older version of their policy. One of the vendors sent us their updated policy for their mobile app. However, the policy has a link to refer to their full privacy policy, which links to the policy that we had analyzed. We requested clarifications on their “full policy,” but received no response. We analyzed the remaining three policies and found that they had the contradicting language removed in their updated policy.

**No Comment:** Three vendors responded without providing a comment or clarification of their policy. One developer simply replied back, “Thanks for the observation.” The other responded that their app was removed from Google Play. The last responded that they use a template for their policy.

## 4.5 Limitations

PolicyLint provides a set of techniques to extract a concise structured representation of both collection and sharing statements from the unstructured natural language text in privacy policies. In so doing, it provides unprecedented analysis depth, and our findings in Section 4.4 demonstrate the utility and value of such analysis. However, extracting structured information from unstructured natural language text continues to be an open and an active area of NLP research, and there are currently no perfect techniques for this challenging problem. PolicyLint is thus limited by the current state of NLP techniques, such as the limitations of NLP parsers and named-entity recognition. Its performance also depends on its verb lists and policy statement patterns, which may be incomplete despite

our best efforts, reducing overall recall. We note that as a `Lint` tool, our goal was always to provide high precision at a potential cost to high recall. We note that PolicyLint achieves a 97.3% precision (496/510) based on the 14 false positives identified during our validation of contradictions.

Another limitation is that PolicyLint cannot extract the conditions or purposes behind collection and sharing statements. Extracting such information would allow for a more holistic analysis, but doing so would require advances in decades-old NLP problems, including semantic role labeling, coreference resolution, and natural language understanding.

Finally, our analysis focused on policies of Android apps. While we cannot claim that our findings will certainly generalize to policies from other domains (e.g., iOS, web), we hypothesize that self-contradictions likely occur across-the-board, as policies are written for all platforms in largely the same ways to describe data types collected by all platforms.

## 4.6 Related Work

Recent research has increasingly focused on automated analysis of privacy policies. Systems have used NLP for deriving answers to a limited number of binary questions [ZB14] from privacy policies, applied topic modeling to reduce ambiguity in privacy policies [SR09], and used data mining [Zae13] or deep learning [Har18] models to extract summaries from policies of what and how information is used. Other related works [Sla16; Wan18] have used *crowdsourced* ontologies for policy analysis. These methods are often limited by lack of accuracy, completeness, and collection complexity. Prior research has also attempted to infer negative statements in privacy policies with limited success. Zimmeck et al. [Zim17b] and Yu et al. [Yu16] rely on keyword-based techniques of using bi-grams and verb modifiers, respectively, to detect the negative statements. In contrast to all these previous approaches, our work provides a more comprehensive analysis with an automatically constructed ontology and accounting for negations and exceptions in text.

Prior works [Cra16; BR13; Bre15; Yu16] identified the possibility of conflicting policy statements. However, we are the first to characterize and automatically analyze self-contradictions resulting from the interaction of varying semantic levels with negative statements. Cranor et al. [Cra16] analyzed the structured table-based model privacy forms of financial institutions and incidentally discovered cases of self-contradictory statements during their analysis. In contrast, we identify contradictions in the unstructured natural

language text of privacy policies. Breaux et al. [BR13; Bre15] provide a formal language to detect conflicts in privacy policies for multi-tier applications manually. Yu et al. [Yu16] developed a system that verifies the consistency of an app’s privacy policy with the policies of the application’s libraries. In contrast, we characterize nine different types of contradictory statements, discuss their impact on automated analysis, and perform the first large-scale empirical study of self-contradictory policy statements to measure the scope of the problem. Further, unlike Breaux et al. [BR13; Bre15], we do not rely on manual analysis to analyze privacy policies.

Other research has proposed techniques to automatically extract ontological relations of privacy-sensitive data from natural language text. Hosseini et al. [Hos18] define a set of heuristics for inferring ontological fragments from compound noun phrases that describe information types. Evans et al. [Eva17] manually identify patterns in constituency-based parse trees that describe hyponym relationships and use a crowd-sourced lexicon of information types to prune false positives. Differing from prior works, we provide a clear methodology for constructing a usable ontology from the extracted ontological relationships, which does not rely on a pre-established lexicon to reduce false positives. Further, we construct both a data object ontology and an entity ontology while prior works focus only on extracting the ontological relationships that describe data types. However, our work can leverage the patterns that these prior works identify to determine hyponymy and other ontological relationships to improve completeness of extraction.

Analyzing the usability and effectiveness of privacy policies is another well-researched focus area. Research has shown that privacy policies are hard to comprehend by users [MC08] and proposals have been made to simplify their understanding [Nor15; Ram14]. Cranor et al. [Cra16] performed a large-scale study of privacy notices of US financial institutions to highlight a number of concerning practices. Their policy analysis relies on standardized models used for such notices; in contrast, privacy policies in mobile apps follow no such standards making the analysis more challenging. Other approaches [Lin14; Liu14; Rao16] have attempted to bridge the gap between users’ privacy expectations and app policies. SPARCLE [Bro06] follows an alternative approach of privacy compliance by deriving machine-readable operational directives from policies written in natural language. While standardizing privacy policy specification has been attempted [Cra02] with limited success [Mar14], mobile apps’ privacy policies have generally failed to adhere to any standards. The concerning findings in our study highlight the need to renew standardization

discussion.

## **4.7 Conclusion**

This chapter introduced PolicyLint, a privacy policy analysis tool that uses natural language processing to identify contradictory sharing and collection practices within privacy policies. PolicyLint reasons about contradictory policy statements that occur at different semantic levels of granularity by auto-generating domain ontologies. We ran PolicyLint on 11,430 privacy policies from popular apps on Google Play and find that around 17.7% of the policies contain logical contradictions and narrowing definitions, with 14.2% containing logical contradictions. Upon deeper inspection, we found a myriad of concerning issues with privacy policies, including misleading presentations and re-defining common terms. PolicyLint’s fine-grained extraction techniques and formalization of narrowing definitions and logical contradictions lays the foundation to help ensure the soundness of future automated policy analysis works and identify potentially deceptive policies.

## CHAPTER

# 5

# ACTIONS SPEAK LOUDER THAN WORDS: ENTITY-SENSITIVE PRIVACY POLICY AND DATA FLOW ANALYSIS WITH POLICHECK

In this chapter, we propose POLICHECK, which provides an entity-sensitive and negation-sensitive flow-to-policy consistency model. POLICHECK's entity-sensitive flow-to-policy consistency model addresses the fundamental weakness of prior works that do not differentiate the entity (e.g., first-party vs. third-party) receiving the privacy-sensitive data. We use POLICHECK to study 13,796 applications and their privacy policies and find that up to 42.4% of applications either incorrectly disclose or omit disclosing their privacy-sensitive data flows. Our results also demonstrate the significance of considering the entity: without considering entity, prior approaches would falsely classify up to 38.4% of applications as having privacy-sensitive data flows consistent with their privacy policies. These false clas-

sifications include data flows to third-parties that are omitted (e.g., the policy states only the first-party collects the data type), incorrect (e.g., the policy states the third-party does not collect the data type), and ambiguous (e.g., the policy has conflicting statements about the data type collection). By defining a novel automated, entity-sensitive flow-to-policy consistency analysis, Polichack provides the highest-precision method to date to determine if applications properly disclose their privacy-sensitive behaviors.

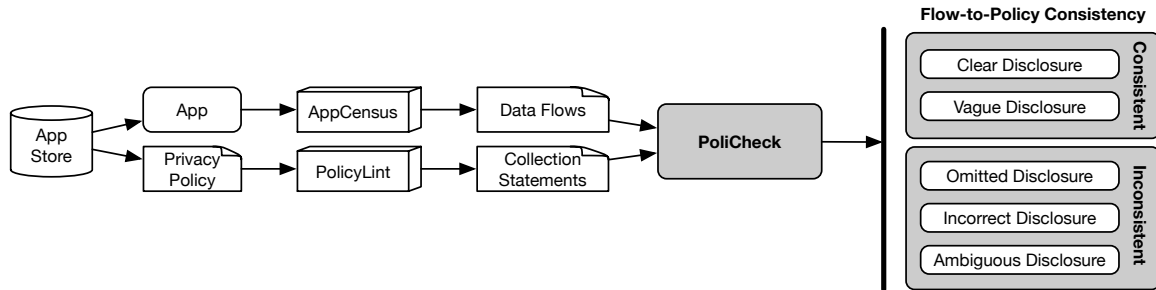
## 5.1 Introduction

Privacy is a long-standing open research challenge for mobile applications. The literature has proposed various program analysis tools for Android [Enc10; Enc11; Arz14b; FWR14] and iOS [Ege11] applications, often citing private-information disclosure as motivations. Subsequent empirical studies [Gra12b; Han13a; Ren16; Rey18; Raz18; Rea19] have demonstrated pervasive and continual disclosure of privacy-sensitive information such as device identifiers and geographic location.

Broadly speaking, the concept of privacy is only vaguely defined and frequently debated. Privacy resides at the intersection of technical, cultural, and legal considerations. In the case of mobile applications, data collection and sharing are often considered (legally) acceptable if it is disclosed in the privacy policy for the application. While there have been several manual analyses of application privacy policies [Bow17; Bow19], it is hard to computationally reason about what privacy policies say, and therefore how applications adhere to them.

Recently, a promising thread of research has begun studying privacy policies and mobile applications [Sla16; Yu16; Zim17b; Wan18]. The goal of these studies is to help application developers adhere to privacy policies, help application stores identify privacy violations, and help end users choose more-privacy-friendly applications. Conceptually, these studies use a combination of static program analysis and natural language processing (NLP) to perform an analysis of *flow-to-policy consistency*. Simply, flow-to-policy consistency analysis determines whether an app’s behavior is consistent with what is declared in the privacy policy.

While such prior studies have led to promising results, the techniques have a fundamental weakness: they do not differentiate the *entity* (e.g., first-party vs. third-party) receiving the privacy-sensitive data. For example, consider the following sentence from a popular



**Figure 5.1** POLICHECK determines the consistency of a mobile application’s data flows to its privacy policy.

Android application with over 10 million downloads:

*When you launch any of our applications, we collect information regarding your device type, operating system and version, carrier provider, IP address, Media Access Control (MAC) address, International Equipment Mobile ID (IMEI), whether you are using a point package, the game version, the device’s geo-location, language settings, and unique device ID.*

This policy statement indicates that the app (the first-party) collects different device identifiers; however, there is no mention of third-parties collecting this information. In actuality, dynamic analysis found that the application sends the IMEI, Android ID, and Ad ID to Tapjoy and the Android ID and Ad ID to Flurry (two third-party advertising providers). Without considering the entity receiving the privacy-sensitive data, prior work would incorrectly classify these data flows as being consistent with the policy.

In addition to not considering the entity, prior studies either do not consider negative statements (e.g., not\_collect) [Sla16; Wan18], or they are insensitive to the semantic granularity of data types (e.g., an IMEI is a type of device identifier) [Yu16; Zim17b]. In Chapter 4, we demonstrated that over 14% of privacy policies contain contradictions, many of which require natural language processing that is sensitive to both negation and semantic granularity. Not considering negations and contradictions within privacy policies leads to both false positives and false negatives when measuring flow-to-policy consistency.

In this chapter, we propose POLICHECK, which uses an *entity-sensitive* flow-to-policy consistency model to determine if an application’s privacy policy discloses relevant data flows. We formally define five types of disclosures (including non-disclosures) that are sensitive to the semantic granularity of both the data type and the entity receiving the data.

We use POLICHECK to study the flow-to-policy consistency of 13,796 Android applications observed to send privacy sensitive values to servers during dynamic analysis (45,603 data flows). We found several significant flow-to-policy inconsistencies in popular real-world applications that impact tens-of-millions of users, such as not disclosing data sharing with advertisers and analytics providers in privacy policies. In general, we found that applications almost never clearly disclose their privacy-sensitive data flows to third parties. In fact, 40.4% of data flows involving third-party entities are broadly discussed using the term “third-party,” leaving it up to guesswork to determine where the data is flowing. Furthermore, we found 5.2% of applications state that they do not share or collect a specific type of information within their privacy policy, but dynamic analysis shows the opposite. Our results from our empirical study and case studies highlight the poor state of privacy policies for Android applications, which demonstrates the need for action from regulatory agencies and application markets.

This chapter makes the following main contributions:

- *We formally define an entity-sensitive flow-to-policy consistency model for mobile applications.* This model includes two types of consistencies and three types of inconsistencies. By considering entities, the model avoids significant *miss-classifications* that result from prior approaches.
- *We design and implement the POLICHECK tool for analyzing the flow-to-policy consistency of Android applications.* POLICHECK builds on top of our work in Chapter 4 for privacy policy analysis and AppCensus [Appb] for dynamic analysis of Android applications. In doing so, we bridge the gap between the low-level data types and DNS domains used by program analysis tools and the often higher-level concepts present in privacy policies.
- *We study and characterize the flow-to-policy consistency of 13,796 Android applications.* Our characterization differentiates first-party and third-party collection and demonstrates the importance of an entity-sensitive consistency model. We show that our entity-sensitive consistency finds significant flow-to-policy inconsistencies that involve sharing data to third-party entities, impacting tens-of-millions of users.

The remainder of this chapter proceeds as follows. Section 5.2 uses examples to provide the high-level intuition behind POLICHECK. Section 5.3 formally defines the different types of flow-to-policy consistencies and inconsistencies. Section 5.4 describes the design of

POLICHECK. Section 5.5 presents our empirical study. Section 5.6 discusses limitations and assumptions. Section 5.7 overviews related work. Section 5.8 concludes.

## 5.2 Flow-to-Policy Consistency

This section motivates POLICHECK’s functionality through five examples that POLICHECK identified. We simultaneously provide a high-level intuition of its functionality by walking through how a human analyst might approach the task. In doing so, we also exemplify the limitations of prior work. This section does not cover every corner case. Sections 5.3 and 5.4 describe POLICHECK in detail.

As shown in Figure 5.1, POLICHECK seeks to determine if the privacy policies for mobile applications disclose their privacy-sensitive data flows to different network entities, as required by various regulations [Law03; PEU16]. We define a data flow as a type of privacy-sensitive data (e.g., IMEI, location, email address) and the entity receiving the data (e.g., Facebook, TapJoy, AdMob). If an application’s privacy policy appropriately discusses the sharing or collection of the specific data type to or by a specific entity for a given data flow, we refer to the data flow as being *consistent* with the privacy policy. To ensure sufficient evidence of sharing or collection by an entity, we scope data flows to network transmission identified during dynamic analysis. In contrast, static analysis may over-approximate data flows (e.g., some ad libraries collect geographic location based on an application developer’s server-side configuration).

### 5.2.1 Clear Disclosures

A data flow has a *clear disclosure* when there exists a statement within the privacy policy that explicitly discusses the exact entity of the data flow receives the exact type of data of the data flow, *and* there is no other policy statement that contradicts it. For illustrative purposes, consider the “Dr. Panda Town: Vacation” (`com.drpanda.town.holiday`) game application with over 1 million downloads on Google Play. This application is built on top of the Unity third-party game engine. For analytics purposes, it obtains the device’s advertising identifier and sends it to `cdp.cloud.unity3d.com` (i.e., Unity).

To determine if this data flow is disclosed by the privacy policy, the first step is to resolve `cdp.cloud.unity3d.com` to the entity “Unity” by matching the root domain (`unity3d.com`) to a list of known analytics providers. For each policy statement, we look for a direct

positive statement match between the flow’s data type and entity and the policy statement’s data type and entity. In this case, we identify the following statement, “*Unity collects the following information through our Games: unique device ID and AD ID.*” We then look for policy statements that contradict the statement by extracting all negative statements that discuss the flow’s data type and entity at any semantic granularity (e.g., analytics providers collecting device information). In this specific case, we do not find any policy statements that contradict the statement extracted above. Therefore, we label this case as a clear disclosure.

### 5.2.2 Vague Disclosures

A data flow has a *vague disclosure* when the only statements within a privacy policy that match a data flow use broad terms for the data type or entity. Similar to clear disclosures, a statement is a vague disclosure only if a contradictory policy statement does not exist. We differentiate vague disclosures from clear disclosures, because there is a risk that the language used to disclose the data flow is so broad that it encapsulates a wide-range of data flows, making it difficult to determine if third-party sharing or collection occurs. Vague disclosures are equivalent to Slavin et al. [Sla16] and Wang et al.’s [Wan18] definition of weak violations.

As an example, consider the popular “Elite Killer: SWAT” (`com.yx.sniper`) game application on Google Play with over 10 million downloads and a 4.3 star rating. For monetization purposes, this application uses the TapJoy advertising provider to deliver advertisements within the application. When requesting advertisements from TapJoy, the application obtains the user’s Android advertising identifier and transmits it to `ws.tapjoyads.com`.

Similar to the previous example, we resolve `ws.tapjoyads.com` to “TapJoy” through a substring match of the root domain in our list of known advertisers. However, rather than identifying only direct matches, we look for policy statements that are not negated that match at any semantic granularity for the flow’s data type and entity. In this case, we identify the following statement, “*A device identifier and in-game or user session activity may be shared with the advertiser.*” This statement matches the data flow, because TapJoy is an advertiser and the Android advertising identifier is a type of device identifier. Next, we look for matching policy statements that are negated. Since we do not find any policy statements that contradict this statement, we label this data flow as a vague disclosure. Finally, we calculate a vagueness score for the resolved policy statement to allow a ranked

ordering, which is based on a normalized ontological distance between the flow’s data type and entity and the policy statements data type and entity.

### 5.2.3 Omitted Disclosure

A data flow has an *omitted disclosure* when there are no policy statements that discuss it. Omitted disclosures are similar to Wang et al.’s [Wan18] definition of strong violations. However, as we demonstrate in the following example, prior definitions do not consider both data type and entity, and therefore may incorrectly classify an omitted disclosure as being flow-to-policy consistent.

Consider the application “Flash Emoji Keyboard & Themes” (`com.xime.latin.lite`) on Google Play, which currently has over 50 million downloads and a 4.1 star rating. This application uses the Avazu advertising provider to serve advertisements within the application for monetization purposes. When requesting advertisements from Avazu, this application obtains the user’s Android identifier, IMEI, and phone number and transmits that information to Avazu servers (`api.c.avazunativeads.com`).

Similar to the previous cases, we look for policy statements that describe the data flow at any semantic granularity, but with both positive and negative actions. For these data flows, we do not find any policy statements that match both data type and entity.

This example application demonstrates the need for considering both the data type and entity. If we only considered the data type, we would identify the following policy statement: “*When you access our Services, we automatically record and upload information from your device including, but not limited to attributes such as the operating system, hardware version, device settings, battery and signal strength, device identifiers...*” This policy statement indicates application itself is collecting device identifiers. However, it does not disclose a data flow to the advertiser. Therefore, the Android identifier, IMEI, and phone number data flows to the advertiser lack flow-to-policy consistency. Prior works [Sla16; Wan18; Zim17b] that do not consider entities when reasoning over privacy policies would have incorrectly identified these data flows as consistent.

### 5.2.4 Incorrect Disclosure

A data flow has an *incorrect disclosure* if a policy statement indicates that the flow will *not* occur (i.e., a negative sharing or collection statement) and there is not a contradicting

positive statement. However, we must be careful when determining if a positive policy statement contradicts the negative statement. In Chapter 4, we identified a class of *narrowing definitions* (labeled  $N_1$  to  $N_4$  in Table 5.1) that use a negative statement when referring to a *more specific* data type or entity. A human reading these policy statements would not view them as contradicting; rather, the negative statement would be viewed as providing an exception to a broad sharing or collection practice. Therefore, we still classify a data flow as an incorrect disclosure if there is a corresponding positive statement that matches the narrowing definitions relationship.

For example, consider the “Furby BOOM!” (`com.hasbro.FurbyBoom`) game application, which has over 10 million downloads on Google Play. This application is built on top of the Unity third-party game engine. To provide statistics to Unity for optimization purposes of their game platform, the application obtains and sends the device’s IMEI to Unity (`stats.unity3d.com`).

Similar to the above cases, we find all relevant policy statements that describe the data flow at any semantic granularity. In this case, we only find one policy statement, “*Our Apps do not send the device ID or IP address to us or to any third-party, and our App does not make further use of this information.*” As the device’s IMEI is a type of device identifier and Unity is a third-party, the application is inconsistent with its own policy, as it is stating that the data flow should not exist. Note that prior works [Sla16; Wan18] would have incorrectly identified this data flow as consistent with the policy, as they do not handle negative statements.

### 5.2.5 Ambiguous Disclosure

A data flow has an *ambiguous disclosure* if the flow matches two or more contradictory policy statements where it is not clear if the flow will or will not occur. As mentioned above, our PolicyLint tool identified different types of relationships between positive and negative sharing statements, as described in Chapter 4. We classify a data flow as having an ambiguous disclosure if there exist two policy statements that have a *logical contradiction* relationship ( $C_1$  to  $C_5$  in Table 5.1), but not a narrowing definition relationship ( $N_1$  to  $N_4$  in Table 5.1), as described above. Furthermore, we have expanded PolicyLint’s classification of conflicting policy statements with a new class of *flow-sensitive contradiction* relationships ( $C_6$  to  $C_{12}$  in Table 5.1), which Section 5.3 explains in more detail. Data flows matching two or more policy statements with a flow-sensitive contradiction relationship are also classified as ambiguous disclosures.

For example, consider the “Flip Diving” (`com.motionvolt.flipdiving`) game application, which has over 50 million downloads on Google Play. This application uses the AdColony advertising provider to serve advertisements for monetization purposes. When requesting advertisements from AdColony, the application obtains the user’s Android advertising identifier and transmits it to `androidads23.adcolony.com`.

Similar to the above cases, we find all relevant policy statements that describe the data flow at any semantic granularity. In this case, we only find two relevant policy statements, “*On our apps, these third-party advertising companies will collect and use your data to provide you with targeted advertising that is relevant to you and your preferences with your consent.*” and “*We don’t give or sell your data to third parties for them to market to you.*” As their policy states that they both do and do not give your data to third-parties for advertising/marketing, the policy is ambiguous. Note that prior works [Sla16; Wan18] would have incorrectly identified this data flow as consistent, as they do not capture negative sharing or collection statements, nor do they identify conflicting statements.

## 5.3 Consistency Model

In this section, we provide the core logic model for our definition of flow-to-policy consistency, as motivated in Section 5.2. We begin with a formal specification of data flows and privacy policy statements. We then introduce four ontological operations required for reasoning over data flows. Finally, we formalize the two types of flow-to-policy consistencies (clear disclosures and vague disclosures) and the three types of inconsistencies (omitted disclosures, incorrect disclosures, and ambiguous disclosures).

### 5.3.1 Data Flow and Policy Statements

We model an application  $a$  as a tuple,  $a = (F, P)$ , where  $F$  is a set of data flows observed for the application and  $P$  is a set of sharing and collection policy statements extracted from the application’s privacy policy. Let  $D$  represent the total set of data object types and  $E$  represent the total set of entities. Then, a data flow is represented by the following definition.

**Definition 4** (Data Flow). *A data flow  $f \in F$  is a tuple  $f = (e, d)$  where  $d \in D$  is the data object type that is sent to an entity  $e \in E$ .*

For example, an application that sends the device’s advertising identifier to AdMob can be concisely represented by the data flow tuple (AdMob, advertising identifier).

Similar to Chapter 4, we represent a sharing and collection policy statement as a tuple (*actor*, *action*, *data type*, *entity*) where the *actor* performs an *action* on a *data type*, and an *entity* receives a data object of that type. We consider four actions: *share*, *not share*, *collect*, and *not collect*. For example, the statement, “*We will share your personal information with advertisers*” is represented as (we, share, personal information, advertisers). As our analysis can only observe client-side behaviors, we adopt the simplified policy statement form described in Chapter 4, which transforms the 4-tuple into a more compact 3-tuple. Intuitively, these transformation rules remove the actor and sharing actions and only considers the entities who may possibly receive (i.e., collect) the data type based on the policy statement (e.g., sharing data implies the actor also collects it). Therefore, we represent a policy statement as follows.

**Definition 5** (Policy Statement). *A policy statement  $p \in P$  is a tuple,  $p = (e, c, d)$ , where data type  $d \in D$  is either collected or not collected,  $c = \{\text{collect}, \text{not\_collect}\}$ , by an entity  $e \in E$ .*

For example, the above 4-tuple (we, share, personal information, advertisers) is represented as two 3-tuples: (we, collect, personal information) and (advertisers, collect, personal information).

### 5.3.2 Ontological Operations

Privacy policies may disclose data flows using terms with a different semantic granularity than the actual data flow. For example, a privacy policy may specify (advertiser, collect, device identifier) to disclose the data flow (AdMob, advertising identifier). To match the policy statement to the data flow, an analysis tool must know that AdMob is an advertiser and an advertising identifier is a type of device identifier. These relationships are commonly referred to as subsumptive relationships, where a more specific term is subsumed under a more general term (e.g., AdMob is subsumed under advertisers, and advertising identifier is subsumed under device identifier). Such relationships are often encoded into an *ontology*, which is a graph where terms are nodes and edges are labeled with the relationship between those terms (e.g., “is-a”).

Our analysis uses two different ontologies: data type and entity. While ontologies can represent several different types of relationships, our ontologies are limited to “is-a” rela-

tionships. We use the following notation to describe binary relationships between terms in a given ontology, which expands on the operators defined in Chapter 4. The operators are parameterized with an ontology  $o$ , which represents either the data type ontology ( $\delta$ ) or entity ontology ( $\epsilon$ ) For example,  $x \equiv_{\delta} y$  and  $x \equiv_{\epsilon} y$ .

**Definition 6** (Semantic Equivalence). *Let  $x$  and  $y$  be terms partially ordered by an ontology  $o$ .  $x \equiv_o y$  is true if  $x$  and  $y$  are synonyms, defined with respect to an ontology  $o$ .*

**Definition 7** (Subsumptive Relationship). *Let  $x$  and  $y$  be terms partially ordered by “is-a” relationships in an ontology  $o$ .  $x \sqsubset_o y$  is true if term  $x$  is subsumed under the term  $y$  and  $x \not\equiv_o y$  (e.g., “ $x$  is-a  $y$ ” or “ $x$  is-a... is-a  $y$ ”). Similarly,  $x \sqsubseteq_o y \implies x \sqsubset_o y \vee x \equiv_o y$ .*

In addition to these two ontological operators, we identify a third type of ontological operator that impacts flow-to-policy consistency analysis. We define a *semantic approximation* operator that identifies terms that have common descendants in the ontology, but are not direct descendants of one other. For example, consider we have the data flow (Flurry, advertising identifier) and the policy statements: (advertiser, not\_collect, identifiers) and (analytic provider, collect, identifier). As Flurry is both an advertiser and analytics provider (common descendant), the policy becomes ambiguous when considering whether the data flow is disclosed by the policy. We define semantic approximation as follows.

**Definition 8** (Semantic Approximation). *Let  $x$  and  $y$  be terms partially ordered by “is-a” relationships in an ontology  $o$ .  $x \approx_o y$  is true if  $\exists z$  such that  $z \sqsubset_o x \wedge z \sqsubset_o y \wedge x \not\equiv_o y \wedge y \not\equiv_o x$ .*

Finally, when discussing vague disclosures, it is useful to characterize the vagueness using a metric. To help define this metric, we define the following to operations to determine a distance between two terms in a given ontology.

**Definition 9** (Ontological Distance). *Let  $x$  and  $y$  be terms partially ordered by “is-a” relationships in an ontology  $o$ , and  $x \sqsubseteq_o y$ . The ontological distance  $\Delta_o(x, y)$  is the shortest path between  $x$  and  $y$ .*

**Definition 10** (Normalized Ontological Distance). *Let  $x$  and  $y$  be terms partially ordered by “is-a” relationships in an ontology  $o$ , and  $x \sqsubseteq_o y$ . The normalized ontological distance  $\hat{\Delta}_o(x, y)$  is the length of shortest path between  $x$  and  $y$  divided by the length of the shortest path between  $x$  and the root node ( $\top$ ) that goes through through  $y$ . More specifically,*

$$\hat{\Delta}_o(x, y) = \frac{\Delta_o(x, y)}{\Delta_o(x, y) + \Delta_o(y, \top)}$$

### 5.3.3 Consistency

Section 5.2 informally defined five types of disclosures used to describe flow-to-policy consistency and inconsistency. This section formally defines a consistency model via the logical relationships between terms in data flows and policy statements. A key part of the informal definitions in Section 5.2 is the interpretation of situations with conflicting policy statements, that is, contradictions and narrowing definitions. The existence of such policy statement conflicts requires flow-to-policy consistency analysis to consider the policy as a whole, rather than looking for the existence of any sharing or collection statement, as done in prior work [Sla16; Wan18; Zim17b].

In Chapter 4, we introduced five types of *logical contradictions* ( $C_1$  to  $C_5$  in Table 5.1) and four types of *narrowing definitions* ( $N_1$  to  $N_4$  in Table 5.1). Logical contradictions are a pair of policy statements that are either exact contradictions ( $C_1$ ) or those that discuss not collecting broad types of data, but also discuss collecting exact or more specific types ( $C_2$  to  $C_5$ ). Narrowing definitions are a pair of policy statements where broad data objects are stated to be collected, and specific data objects are stated to not be collected ( $N_1$  to  $N_4$ ).

We expand on the contradictions in Chapter 4 and define a third pairing of conflicting policy statements called *flow-sensitive contradictions* ( $C_6$  to  $C_{12}$  in Table 5.1). Flow-sensitive contradictions are a pair of policy statements with opposing actions, such that at least one of the data types or entities are semantically approximate to the other. Similar to logical contradictions, flow-sensitive contradictions result in an ambiguous policy when reasoning whether a specific data flow is disclosed. For example, consider we have the data flow (Flurry, advertising identifier) and the policy statements (analytic provider, collect, advertising identifier) and (advertiser, not\_collect, advertising identifier). Since Flurry is both an advertiser and analytic provider, the policy is ambiguous with respect to this data flow.

Before defining our flow-to-policy consistency model, we define three filters on the set policy statements in a policy. These filters simplify the notation used to formally describe the five disclosure types. The following discussion assumes the analysis of an individual application, and each disclosure is described with respect to a specific data flow  $f$ . Applications may have multiple data flows. Furthermore, each application has a set of policy statements  $P$ .

**Definition 11** (Contradicting Policy Statements). *Let  $P$  be a set of policy statements (Definition 5).  $P_C$  is the set of policy statements  $p \in P$  for which there exists a  $p' \in P$  such that  $p$  and*

**Table 5.1** Types of conflicting policy statements in a privacy policy: logical contradictions ( $C_{1-5}$ ), flow-sensitive contradictions ( $C_{6-12}$ ), and narrowing definitions ( $N_{1-4}$ ).

Rule	Logic	Example*
$C_1$	$e_i \equiv_{\varepsilon} e_j \wedge d_k \equiv_{\delta} d_l$	(Flurry, collect, IMEI) (Flurry, not_collect, IMEI)
$C_2$	$e_i \equiv_{\varepsilon} e_j \wedge d_k \sqsubset_{\delta} d_l$	(Flurry, collect, IMEI) (Flurry, not_collect, Device Info)
$C_3$	$e_i \sqsubset_{\varepsilon} e_j \wedge d_k \equiv_{\delta} d_l$	(Flurry, collect, IMEI) (Advertiser, not_collect, IMEI)
$C_4$	$e_i \sqsubset_{\varepsilon} e_j \wedge d_k \sqsubset_{\delta} d_l$	(Flurry, collect, IMEI) (Advertiser, not_collect, Device Info)
$C_5$	$e_i \sqsupset_{\varepsilon} e_j \wedge d_k \sqsubset_{\delta} d_l$	(Advertiser, collect, IMEI) (Flurry, not_collect, Device Info)
$C_6$	$e_i \equiv_{\varepsilon} e_j \wedge d_k \approx_{\delta} d_l$	(Flurry, collect, Device Info) (Flurry, not_collect, Track Info)
$C_7$	$e_i \sqsubset_{\varepsilon} e_j \wedge d_k \approx_{\delta} d_l$	(Flurry, collect, Device Info) (Advertiser, not_collect, Track Info)
$C_8$	$e_i \sqsupset_{\varepsilon} e_j \wedge d_k \approx_{\delta} d_l$	(Advertiser, collect, Device Info) (Flurry, not_collect, Track Info)
$C_9$	$e_i \approx_{\varepsilon} e_j \wedge d_k \equiv_{\delta} d_l$	(Analytic, collect, IMEI) (Advertiser, not_collect, IMEI)
$C_{10}$	$e_i \approx_{\varepsilon} e_j \wedge d_k \sqsubset_{\delta} d_l$	(Analytic, collect, IMEI) (Advertiser, not_collect, Device Info)
$C_{11}$	$e_i \approx_{\varepsilon} e_j \wedge d_k \sqsupset_{\delta} d_l$	(Analytic, collect, Device Info) (Advertiser, not_collect, IMEI)
$C_{12}$	$e_i \approx_{\varepsilon} e_j \wedge d_k \approx_{\delta} d_l$	(Analytic, collect, Device Info) (Advertiser, not_collect, Track Info)
$N_1$	$e_i \equiv_{\varepsilon} e_j \wedge d_k \sqsupset_{\delta} d_l$	(Flurry, collect, Device Info) (Flurry, not_collect, IMEI)
$N_2$	$e_i \sqsubset_{\varepsilon} e_j \wedge d_k \sqsupset_{\delta} d_l$	(Flurry, collect, Device Info) (Advertiser, not_collect, IMEI)
$N_3$	$e_i \sqsupset_{\varepsilon} e_j \wedge d_k \equiv_{\delta} d_l$	(Advertiser, collect, IMEI) (Flurry, not_collect, IMEI)
$N_4$	$e_i \sqsupset_{\varepsilon} e_j \wedge d_k \sqsupset_{\delta} d_l$	(Advertiser, collect, Device Info) (Flurry, not_collect, IMEI)

\* $P = \{(e_i, \text{collect}, d_k), (e_j, \text{not\_collect}, d_l)\}$ ,  $f = (\text{Flurry}, \text{IMEI})$

$p'$  have a logical contradiction ( $C_{1-5}$ ) or a flow-sensitive contradiction ( $C_{6-12}$ ).

**Definition 12** (Narrowing Definition Policy Statements). *Let  $P$  be a set of policy statements (Definition 5).  $P_N$  is the set of policy statements  $p \in P$  for which there exists a  $p' \in P$  such that  $p$  and  $p'$  have a narrowing definition ( $N_{1-4}$ ).*

**Definition 13** (Flow-Relevant Policy Statements). *Let  $P$  be a set of policy statements (Definition 5) and  $f$  be a data flow (Definition 4).  $P_f$  is the set of policy statements in  $P$  that are relevant to the data flow  $f$ . More specifically,  $P_f = \{p \mid p \in P \wedge f.d \sqsubseteq_\delta p.d \wedge f.e \sqsubseteq_\epsilon p.e\}$*

### 5.3.3.1 Flow-to-Policy Consistency

Using the above definitions, we can now define flow-to-policy consistency. As discussed in Section 5.2, there are two types of consistency: clear disclosures and vague disclosures. We now formally define these three concepts, as well as a vagueness metric to help quantify the significance of vague disclosures.

**Definition 14** (Flow-to-Policy Consistency). *A data flow  $f$  is consistent with an application's privacy policy  $P$  if and only if  $\exists p \in P_f$  such that  $p.c = \text{collect} \wedge \nexists p' \in P_f$  such that  $p'.c = \text{not\_collect}$ .*

**Definition 15** (Clear Disclosure). *An application's privacy policy has a clear disclosure of a data flow  $f$  if there exists a collect policy that uses terms of the same semantic granularity for both data type and entity, and there does not exist a conflicting not\_collect policy for the data type and entity. More specifically, there is a clear disclosure of  $f$  if and only if  $\exists p \in P_f$  such that  $p.c = \text{collect} \wedge f.d \equiv_\delta p.d \wedge f.e \equiv_\epsilon p.e$  and  $\nexists p' \in P_f$  such that  $p'.c = \text{not\_collect}$ .*

**Definition 16** (Vague Disclosure). *An application's privacy policy has a vague disclosure of a data flow  $f$  if there does not exist clear disclosure, but there does exist a collect policy using a broader semantic granularity for either the data type or entity, and there does not exist a conflicting not\_collect policy for the data type and entity. More specifically, there is a vague disclosure of  $f$  if and only if  $\nexists p \in P_f$  such that  $p.c = \text{collect} \wedge f.d \equiv_\delta p.d \wedge f.e \equiv_\epsilon p.e$  and  $\exists p' \in P_f$  such that  $p'.c = \text{collect} \wedge f.d \sqsubseteq_\delta p'.d \wedge f.e \sqsubseteq_\epsilon p'.e$  and  $\nexists p'' \in P_f$  such that  $p''.c = \text{not\_collect}$ .*

A data flow with a vague disclosure is not necessarily bad. However, if the terms in the matching policy statement are too broad, the disclosure may not be meaningful to the

user. For example, the policy statement (third-party, collect, personal data) is considerably more vague than (AdMob, collect, advertising identifier) to describe the data flow (AdMob, advertising identifier). As vagueness is subjective, we do not seek a binary classification (e.g., weak violations [Sla16]). Instead, we provide a quantitative metric [0.0-1.0] to rank and compare statements in term of vagueness. A higher value indicates greater vagueness.

Our metric calculates a tuple for vagueness of a flow  $f$ 's disclosure via a policy statement  $p$  using the ontological distances, with values for both data type and entity. Since the magnitude of ontological distances can vary, we normalize the ontological distance to allow ranked comparisons.

**Definition 17** (Vagueness Metric). *The vagueness of a flow  $f$  by a policy statement  $p$  is represented by  $(\hat{\Delta}_e(f.e, p.e), \hat{\Delta}_d(f.d, p.d))$ .*

### 5.3.3.2 Flow-to-Policy Inconsistency

A data flow is inconsistent with the privacy policy if it does not satisfy the above consistency conditions. We define three types of disclosures that represent flow-to-policy inconsistency: omitted disclosures, incorrect disclosures, and ambiguous disclosures.

**Definition 18** (Omitted Disclosures). *An application's privacy policy has an omitted disclosure of a data flow  $f$  if it does not include either collect or not\_collect statements at any semantic granularity for the flow's data type and entity. More specifically, there is an omitted disclosure of  $f$  if and only if  $P_f = \emptyset$ .*

**Definition 19** (Incorrect Disclosure). *An application's privacy policy has an incorrect disclosure of a data flow  $f$  when the policy states that it does not collect or share the data type. More specifically, there is an incorrect disclosure of  $f$  if and only if  $\forall p \in P_f, p.c = \text{not\_collect}$  or  $(P_f \cap P_N \neq \emptyset \wedge P_f \cap P_C = \emptyset)$ .*

Note that incorrect disclosures include narrowing definitions, because they represent an unambiguous case where a policy has relevant flows with both collect and not\_collect statements. Since narrowing definitions have not\_collect statements for the more specific type, a matching data flow represents an incorrect disclosure.

**Definition 20** (Ambiguous Disclosure). *An application's privacy policy has an ambiguous disclosure of a data flow  $f$  when it contains contradicting statements about the data flow. More specifically, there is an ambiguous disclosure of  $f$  if and only if  $P_f \cap P_C \neq \emptyset$ .*

## 5.4 Design

The core contribution of this chapter is enhancing flow-to-policy consistency analysis with the knowledge of which entities collect information. Determining the type of disclosure (Section 5.3) for each observed flow requires both dynamic analysis of applications and natural language processing of application privacy policies. We chose dynamic analysis over static analysis, because it provides (1) evidence that the flow occurs (some ad libraries use server-side configuration to determine what data types to collect), and (2) the network destination of the flow. As dynamic analysis of Android analysis is received significant treatment in literature, we build upon AppCensus [Appb], which is the latest state-of-the-art for dynamically performing privacy analysis. Similarly for processing privacy policies, we build on top of our PolicyLint tool (as described in Chapter 4), which enhances prior approaches by extracting entities, as well as negative statements. POLICHECK’s implementation primarily consists of the logic described in Section 5.3. However, combining AppCensus and our PolicyLint tool into a system required additional design effort. The remainder of this section describes these components.

**Data Flow Extraction:** AppCensus [Appb] identifies privacy sensitive data flows in Android applications using the approach proposed by Reyes et al. [Rey18]. In particular, Reyes et al. instrument the Android operating system to log access to sensitive resources and use the Android VPN API to intercept and log network traffic (including installing a root certificate to decrypt TLS traffic). They exercise the application with Monkey [And19] and collect both the system and network logs. Next, they identify the privacy-sensitive data values from the system logs in the network traffic logs by using value-matching along with a set of heuristics to detect encodings of the data, such as base64 or hashing algorithms. The data flows reported by AppCensus are a tuple, (destination domain/IP address, data type). For example, the data flow discussed in Section 5.2.1 is represented as (`cdp.cloud.unity3d.com`, advertising identifier).

**Domain-to-Entity Mapping:** While data flows are represented as a type of data being transmitted to a domain or IP address, privacy policies discuss data flows using terms for entities instead of domains (e.g., `cdp.cloud.unity3d.com` could be referred to as “Unity” within the privacy policy). Therefore, POLICHECK must map domain names to entities so that data flows conform to Definition 4. Note that for the data flows that had only IP addresses without domain names, we first perform a reverse-DNS search to try to resolve the IP address

as a domain name. If we could not resolve a domain name, we discard the data flow from the data set.

We curated a list of 144 advertisers and 40 analytics providers on the Google Play store from AppBrain.com. This list included the primary website for each organization. We manually produced a supplementary set of terms based on organization names to search our set of domain names with. After obtaining a list of potential domain names for each organization by keyword matching our search terms, we manually culled incorrect or irrelevant domain names. This resulted in a set of domain names for the top analytics and advertising organizations on Google Play.

**First-Party Entity Classification:** Determining which network domain is the first-party of a given application requires careful consideration. First, we check if reversing the second-level domain name of the network destination domain matches the beginning of the application's package name. For example, if the flow (advertising identifier, analytics.mobile.walmart.com) occurs in the app com.walmart.android, we mark the flow as a first-party flow, as reversing walmart.com results in com.walmart, which matches the beginning of the package name. Similarly, we check if the second-level domain name of the link to the application's privacy policy matches the root domain of the destination domain. For example, the privacy policy of the Walmart application is located at <https://corporate.walmart.com/privacy-security/walmart-privacy-policy> and the destination domain is analytics.mobile.walmart.com. Since the second-level domain name of the privacy policy (walmart.com) matches the second-level domain name domain of the destination domain is walmart.com, we mark the flow as first-party.

**Sharing and Collection Statement Extraction:** POLICHECK uses the policy statements output by our PolicyLint tool. As described in Chapter 4, PolicyLint uses sentence-level natural language processing to extract sharing and collection statements from privacy policies while capturing the entities and data types involved along with whether the statement is negated. PolicyLint outputs policy statements as defined by the form in Definition 5.

**Data Type and Entity Ontology Extension:** We extended our PolicyLint tool's ontologies to include all of the data types involved in data flows and all of the entities identified when constructing the domain-to-entity mapping. We begin by pruning PolicyLint's ontologies to remove nodes and edges that did not reach a data type or entity node in the data flows. For all missing data types and entities, we manually added them to their corresponding ontology and added edges. Finally, we extended PolicyLint's synonym list by searching policy

statements using keywords for entity names and data types. For example, extended the synonym list for “advertising network” by searching the policy statements for “advertising.” **Consistency Analysis:** POLICHECK uses the data flows and policy statements from the prior steps to perform flow-to-policy consistency analysis. To do so, we implemented the consistency model logic defined in Section 5.3.

## 5.5 Flow-to-policy Consistency Characterization

Our primary motivation for creating POLICHECK was to analyze whether applications are disclosing their privacy-sensitive data flows in their privacy policies, especially for third-party sharing. In this section, we use POLICHECK to perform a large-scale study of analyzing the consistency of 45,603 data flows from 13,796 unique Android applications and their corresponding privacy policies.

**Dataset Selection:** To select our dataset, we began by scraping the top 100 free applications (“topselling\_free” collection) across Google Play’s 35 application categories in February 2019. We enhanced the dataset with an additional 42,129 randomly selected Android applications from AppCensus. For each application, we downloaded the data flows from AppCensus and downloaded the HTML privacy policies from the developer’s website via the link on Google Play. We excluded applications that did not have any data flows reported by AppCensus (23,488 apps). We also excluded applications whose privacy policies were not successfully downloaded (e.g., 404 errors, links to homepages) or were not written in English based on Python’s `langdetect` module (6,039 apps). We also excluded data flows that did not map to nodes in our entity ontology, which resulted in a final data set of 13,796 applications with 45,603 data flows.

### 5.5.1 Consistency Analysis

We extracted sharing and collection statements from 94.4% (13,021/ 13,796) of the privacy policies using our PolicyLint tool (as described in Chapter 4). From those policies, 218,257 policy statements were extracted from 48,831 sentences that were identified as a sharing or collection sentence, such that 7,526 had negations and 210,731 were positive. 31.2% (4,299/ 13,796) of policies had at least one negative policy statement and 92.6% (12,779/ 13,796) of policies had at least one positive policy statement. Overall, the policy statements discussed

**Table 5.2** Data Flows and Apps for each Disclosure Type

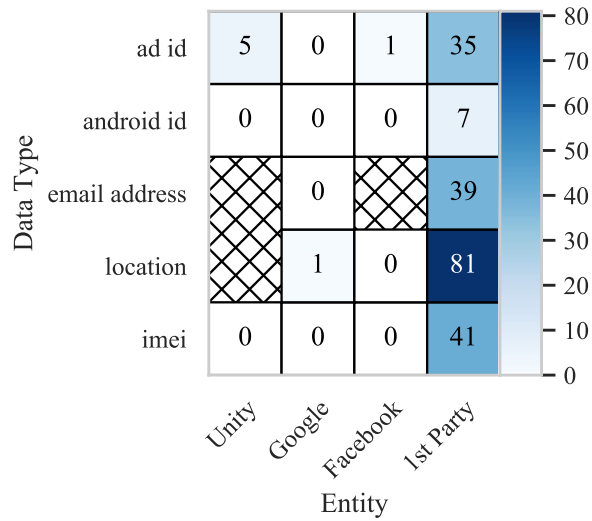
		<b>Clear</b>	<b>Vague</b>	<b>Omitted</b>	<b>Incorrect</b>	<b>Ambiguous</b>
<b>First</b>	<i>Flows</i>	216	2,211	208	18	358
	<i>Apps</i>	206	1,600	154	11	224
<b>Third</b>	<i>Flows</i>	7	23,367	14,201	1,912	3,105
	<i>Apps</i>	7	6,833	5,076	708	892
<b>Total</b>	<i>Flows</i>	223	25,578	14,409	1,930	3,463
	<i>Apps</i>	213	7,626	5,155	719	1,101

34 distinct granularities of data types and 52 distinct granularities of entities. In total, there were 412 unique policy statement tuples.

For the 45,603 data flows across the 13,796 applications, there were 13 unique data types transmitted to 2,243 unique domains. The 2,243 domains were resolved to 112 unique entities. In total, there were 364 unique data flow tuples. Overall, advertising identifiers were the most frequently transmitted data type, which accounted for 26,628 data flows to 100 unique entities across 11,585 applications. Across all of the flows, Unity was the most common data recipient, which accounted for 6,270 data flows containing 6 unique data types across 4,381 applications.

We ran POLICHECK on the dataset and the raw statistics from analysis are listed in Table 5.2. We find that 42.4% of applications contain at least one omitted disclosure or incorrect disclosure, which that the data flow is not disclosed by the policy or is in direct conflict with statements in the policy. The remainder of this section discusses the findings made possible by POLICHECK.

Note that, in this study, we consider device identifiers to be personally identifiable information (PII). First, they are classified as PII under GDPR [PEU16], as they are generally only used for tracking and attribution. Second, other regulations [USC16] have also classified them as PII due to their ability to track users over a long period of time across multiple applications and services. Ad IDs were initially introduced as a pseudonymous identifier to be used to track users for targeted advertising instead of collecting persistent identifiers, such as Android IDs, IMEI, and email addresses. However, we find that 74.7% of entities that receive Ad IDs also receive a persistent identifier, which is PII. Mixing persistent identifiers with Ad IDs nullifies the originally intended properties for Ad IDs, as it loses the property of non-persistence tracking can be bridged across resets. We find that 11,589 applications send this unique identifier to third parties, which is then linkable to users' email addresses, other device identifiers, and other sensitive information.

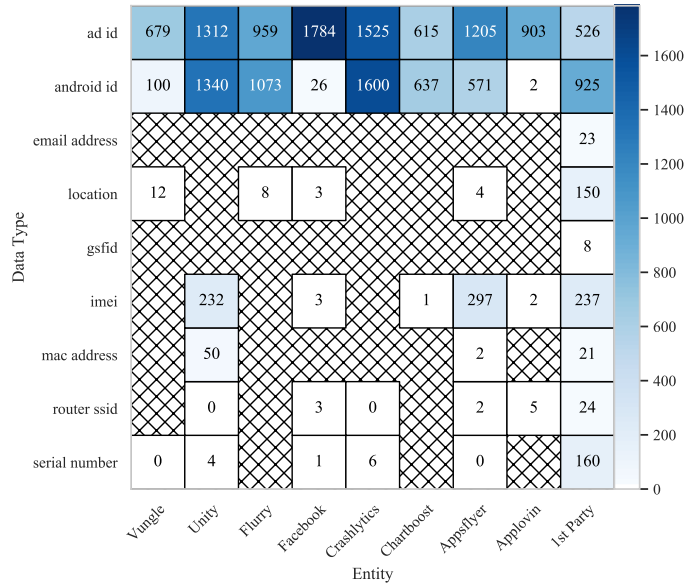


**Figure 5.2** Policies are more clear about the first-party receiving a specific data type than a third-party doing so.

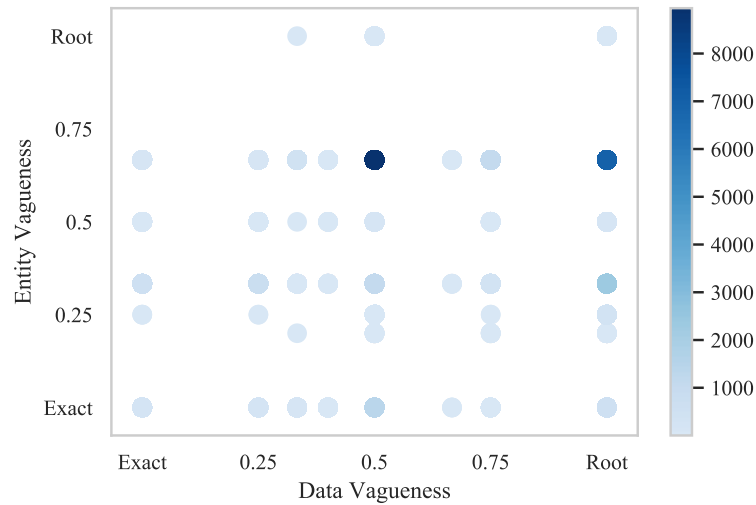
**Finding 1:** Only 0.5% of data flows were explicitly discussed by sentences within the privacy policy in terms of the exact entity and exact data type. In total, only 223 data flows were classified as clear disclosures. Figure 5.2 shows the number of clear disclosures for each of the data flow tuples. The hatched sections denote that there were no transmissions of that specific data type to that entity in the entire dataset. While third parties account for 42,592 of the total data flows, only 7 data flows were classified as clear disclosures. As we discuss in Finding 2, this is likely due to the fact that policies are being vague about third-party disclosures.

In contrast, applications were more likely to clearly disclose their first-party data flows (216 data flows across 206 apps). The most commonly disclosed first-party flow involved location data, which was clearly discussed for 81 data flows. However, as there were over 282 instances of first parties collecting location data, this only accounts for 28.7% (81/ 282) of first-party data flows involving location. Similarly, IMEIs are the second most frequent clear disclosure with 41 data flows, but still only accounted for 12.1% (41/339) of total first-party data flows. The low rate of clear disclosure indicates that privacy policies are not explicitly discussing the types of data that they collect and with whom they share it.

**Finding 2:** 49.5% of applications are disclosing their third-party sharing practices using vague terms. In total, 54.9% (23,367/ 42,592) of third-party flows were disclosed using vague terms to refer to the entity, the data type, or both. Figure 5.3 shows the number of vague



**Figure 5.3** Policies often discuss the sharing of Android and advertising IDs in vague terms.



**Figure 5.4** Policies are likely to use vague terms to describe both data types and entities.

disclosures for each of the data flow tuples with the 8 most common third-party entities. The hatched sections denote that there were no transmissions of that specific data type to that entity in the entire dataset. Ad IDs and Android IDs accounted for 50.2% (21,363/ 42,592) of the vague disclosures for third-party flows. Ad IDs and Android IDs were disclosed 40.7% of the time by the policy statement (third-party, collect, personally identifiable information)

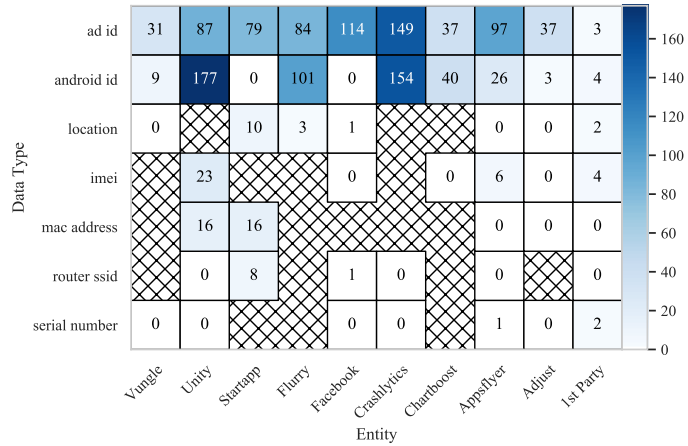
and 25.2% of the time by (third-party, collect, information).

Based on these policy statements, the vagueness of these policy statements does not provide transparency to the wide-range of advertisers and analytics providers that this information is being sent to. As shown in Figure 5.3, Crashlytics and Unity3d were the most frequent entities of data flows that were classified as vague disclosures. Crashlytics is an analytics provider owned by Google. Unity3d provides a game engine to developers, but also provides advertisements and analytics. In particular, data flows to Crashlytics and Unity3d accounted for 7.4% of third-party vague disclosures (3,131/ 42,592). These entities were discussed as third parties in 80.7% (2,528/ 3,131) and 72.9% (2,142/ 2,938) of the time, respectively.

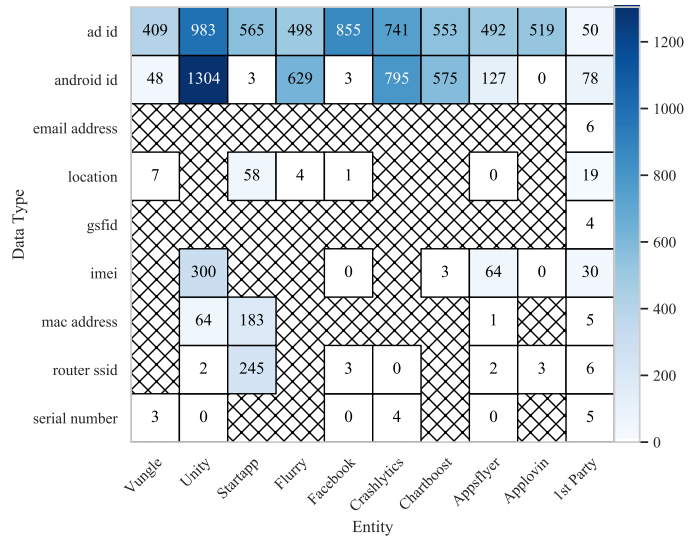
Figure 5.4 shows a graphical representation of the frequency of policy consistency vagueness. Note the root node of the data vagueness is the term “information” while the root node of the entity vagueness is “anyone.” The data vagueness score of around 0.5 generally represents terms such as “personally identifiable information,” “device information,” or “user information.” The entity vagueness score of around 0.67 generally represents the term “third-party” while 0.5 represents terms such as “advertising network” or “analytic provider.” Therefore, in general, third-party data flows are most frequently described in vague terms for both entities and data objects. As the corners of the figure are relatively sparse, it means that if the policy is discussing the entity vaguely then they are likely discussing the data type vaguely. Further, the fact that “third-party” is the most commonly used term to discuss entities, it raises concerns that applications are not complying to the GDPR’s mandate on specificity of disclosures.

**Finding 3:** *11.6% of applications are disclosing their first-party collection practices using broad terms.* In total, 73.4% (2,211/ 3,011) of first-party flows were disclosed using vague terms to refer to the data type. The right column in Figure 5.3 shows the distribution of vague disclosures for first parties. Android IDs accounted for 41.8% (925/ 2,211) of the first-party vague disclosures. Similar to the case for third parties, these flows were most commonly disclosed as the policy tuple (we, collect, personally identifiable information). Surprisingly, they were only disclosed by the terms “device identifiers” or “identifiers” in 20.8% of the flows (192 / 925). A similar trend follows for first-party collection of Ad IDs and IMEIs.

**Finding 4:** *719 applications make incorrect statements about their data practices.* POLICHECK identified that 719 applications contained incorrect disclosures. These applications con-



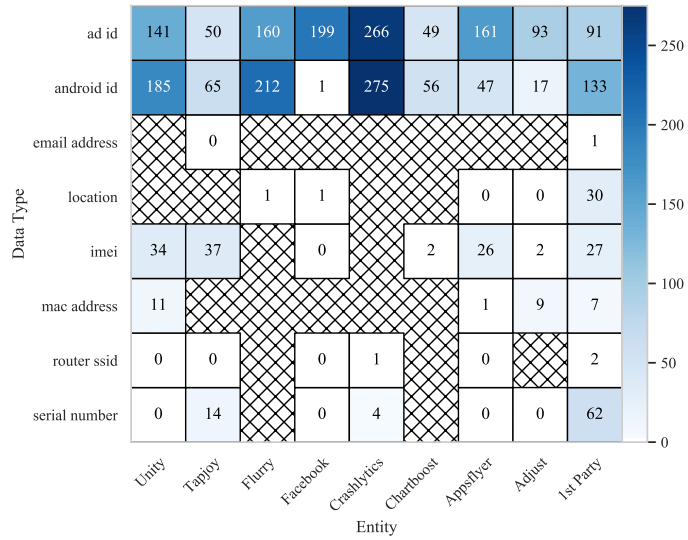
**Figure 5.5** POLICHECK identified 1,912 incorrect disclosures across 719 applications



**Figure 5.6** Policies often do not disclose when they share Android and advertising IDs with third parties. This may be due to the perception that such collection is implied when they disclose that they use an advertiser.

sisted of 4.2% (1,930/ 45,603) of the data flows. Figure 5.5 shows that the most frequent incorrect disclosures involved sharing Ad IDs and Android IDs with third parties. This may be due to the perception that such collection is implied when they disclose that they use an advertiser.

**Finding 5:** POLICHECK found that 31.1% (14,409/45,603) of data flows identified as omitted



**Figure 5.7** Privacy policies are often contradictory when they discuss the sharing of Android and advertising IDs.

*disclosures.* Of the 14,409 omitted disclosures, 208 were first-party flows and 14,201 involved third parties. As shown in Figure 5.6 only 6.9% (208 / 3,011) of first-party data flows were not disclosed. The three most frequently omitted data type for first-party flows were Android IDs (78/208), Ad IDs (50/208), and the device’s IMEI (30/208).

For third-party flows, Figure 5.6 shows that sharing both Android IDs and Ad IDs with Crashlytics and Unity3d accounted for 27.8% (3,168/11,398) and 24.7% (2,810/11,398) of omitted disclosures, respectively. Further, sharing AD IDs with Facebook accounts for 15.8% (1,798/11,398) of third-party omitted disclosures. It is surprising that Crashlytics collects Android IDs, as they are a subsidiary of Google, and using persistent hardware identifiers is against Google’s outline for the best practices on collecting unique identifiers.

The significant number of omitted disclosures raises the two following questions. First, do developers understand the types of data that are actually being collected when they include a third-party SDK into their application? Second, do developers know that they are responsible for disclosing such behaviors in their privacy policy? We leave the exploration of these questions as future work. Note that in comparison to other disclosure types, POLICHECK has lower precision for detecting omitted disclosures. We discuss this in detail in Section 5.5.2 and provide a case study that they may also be used as indicators for policies that are difficult to comprehend.

**Finding 6:** *8.0% of applications have ambiguous privacy policies.* In total, 7.6% (3,463/45,603) of data flows were classified as ambiguous disclosures, which occurred across 8.0% (1,101/13,796) of applications. As shown in Figure 5.7, Android IDs and Ad IDs are involved in 88.8% (3,074/3,463) of ambiguous disclosures. In total,  $C_1$  contradictions were the most common type whose policy statements both state that they do and do not collect information at the same semantic granularity, which accounted for 1,618 types of ambiguous disclosures. For example, on such example is a children’s application called “MiraPhone - Kids Phone 4-in-1 apk” (com.gokids.chydofofon). This application collects the user’s Android ID, but the privacy policy explicitly states “We DO NOT collect your unique identifier [sic],” and also states “Anonymous identifiers, we use anonymous identifiers when you interact with services, such as advertising services and others.” These two statements are contradictory policy statements and it is unclear what the correct interpretation of the policy should be.

## 5.5.2 Evaluation

In this section, we present our evaluation of POLICHECK and additional findings from our evaluation. First, we perform a sensitivity analysis on POLICHECK’s consistency model and show that POLICHECK’s entity-sensitive model vastly outperforms entity-insensitive models. Second, we manually validate a random selection of 153 data flows across 151 applications and show that POLICHECK has a 90.8% precision. The remainder of this section describes these experiments.

### 5.5.2.1 Sensitivity Analysis

To measure the impact of entities in flow-to-policy consistency, we simulated the error rate of entity-insensitive consistency models (i.e., models that do not consider entities) by running consistency analysis in the following three configurations: (1) without entities and without negations (negation-insensitive and entity-insensitive); (2) without entities (entity-insensitive); and (3) without negations (negation-insensitive). Based on the output of the entity-insensitive consistency analysis, we aim to measure the potential error rate. First, we measure the frequency in which third-party data flows are reasoned over using policy statements with semantically unrelated entities (i.e.,  $f.e \not\subseteq_e p.e$ ). This first metric measures when unrelated policy statements are used to reason whether a flow is consistent. Second, we measure the frequency in which third-party data flows are classified as consistent in the

**Table 5.3** Sensitivity Analysis of Flow-to-Policy Consistency Models for Classifying Disclosure Types: Entity-insensitive analysis results in the frequent misclassification of data flows.

	PoliCheck	(-, -, d)		(-, c, d)		(e, -, d)	
		✓	X	✓	X	✓	X
<b>Clear</b>	223	223	3,180	216	1,856	223	39
<b>Vague</b>	25,578	22,964	17,149	18,122	9,852	25,578	5,354
<b>Omitted</b>	14,409	2,087	0	2,087	0	14,409	0
<b>Incorrect</b>	1,930	0	0	558	5,081	0	0
<b>Ambiguous</b>	3,463	0	0	2,298	5,533	0	0

\* (-, -, d): negation-insensitive and entity-insensitive

\* (-, c, d): negation-sensitive and entity-insensitive

\* (e, -, d): negation-insensitive and entity-sensitive

entity-insensitive consistency models that would have been classified as consistent in the inconsistent in entity-sensitive consistency models. This second metric measures when unrelated policy statements cause entity-insensitive models to falsely claim that a data flow is consistent when it is in fact inconsistent. Further, we also measure how these different configurations of consistency models impact the classification of disclosure types.

**Finding 7:** *Ignoring entities during consistency analysis may result in incorrectly classifying up to 37.1% of inconsistent third-party flows as consistent.* We first ran analysis simulating negation-insensitive and entity-insensitive consistency models, such as Slavin et al. [Sla16] and Wang et al. [Wan18]’s models. We found that 53.9% (22,959/ 42,592) of third-party flows were falsely resolved to policy statements that discuss semantically unrelated entities. Of those resolved statements, 39.8% (16,931/ 42,592) referenced first parties and 14.2% (6,028/ 42,592) references a semantically unrelated third-party. In terms of consistency, 37.1% (15,807/ 42,592) of third-party flows were falsely marked as consistent across 38.4% (5,304/ 13,796) of applications. Of those results, 23.0% (9,779/ 42,592) were due to first-party policy statements and 14.2% (6,028/ 42,592) due to third-party policy statements with a semantically unrelated entity. This means that negation-insensitive and entity-insensitive models falsely mark 23.0% inconsistencies as consistent.

We next ran consistency analysis simulation entity-insensitive consistency models, such as Zimmeck et al. [Zim17b]. We found 55.8% (23,775/ 42,592) of third-party flows were improperly resolved to policy statements that discuss semantically unrelated entities. Of those resolved statements, 41.6% (17,698/ 42,592) resolved to policy statements referencing first parties and 14.3% (6,077/ 42,592) resolved to third parties. In terms of consistency,

30.5% (13,014/ 42,592) of third-party data flows are falsely marked as consistent across 32.2% (4,445/ 13,796) of applications. Of those results, 16.3% (6,937/ 42,592) were due to first-party policy statements and 14.3% (6,077/ 42,592) due to third-party policy statements with a semantically unrelated entity. This means that entity-insensitive models falsely mark 30.5% of inconsistencies as consistent.

**Finding 8:** *Entity-insensitive analysis results in the frequent misclassification of disclosure types.* Table 5.3 shows results of our sensitivity analysis for classifying each disclosure type. Overall, entity-insensitive consistency models have the worst performance at classifying disclosure types, as they significantly overestimate the number of clear disclosures and vague disclosures. Negation-insensitive consistency models cannot detect incorrect disclosures or ambiguous disclosures, which correspond to 4.2% and 7.6% of data flows, respectively. With negation-insensitive consistency models, the incorrect disclosures or ambiguous disclosures are wrongly classified as either clear disclosures or vague disclosures, which is concerning as these models would state that the data flow is consistent with the policy. While consistency models that are negation-sensitive and entity-insensitive ( $-$ ,  $c$ ,  $d$ ) can theoretically identify incorrect disclosures and ambiguous disclosures, the results show that their identification of these disclosure types are imprecise due to not considering entities. The results from this analysis demonstrate both the importance of entity-sensitive and negation-sensitive analysis at classifying disclosure types and the unprecedented view that POLICHECK’s flow-to-policy consistency model provides on privacy disclosures.

### 5.5.2.2 POLICHECK’s Performance

To evaluate the precision of POLICHECK, we manually validate a subset of data flows. Note that we do not evaluate ambiguous disclosures, as attempting to resolve ambiguity injects annotator bias into the evaluation. For the remaining disclosure types, we randomly select up to 5 data flows for each data type, such that the proportion of first-party to third-party data flows are proportionate to the disclosure type’s population. Our data set consists of 180 data flows across 166 applications.

**Validation Methodology:** For validation, one-of-three authors began by reading through the sentences that were extracted from each privacy policy to ensure correctness of policy statement extraction. If there was an error with policy statement extraction, we record the disclosure as a false positive and stopped analysis. For clear disclosures, vague disclosures, and incorrect disclosures, we locate the sentences in the policy and ensure that the sentence

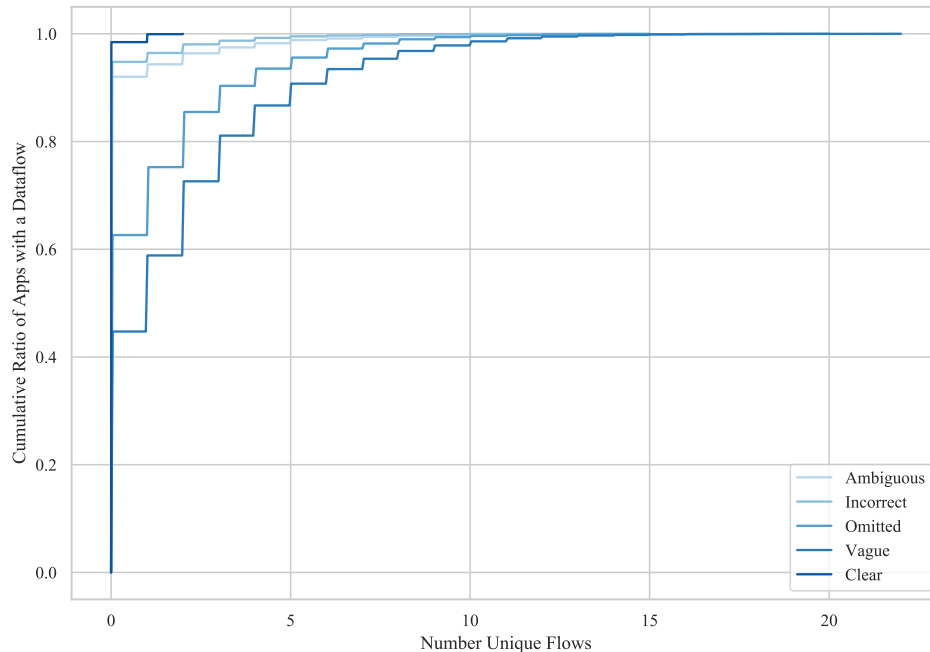
retains the same meaning in the context of the rest of the policy. For omitted disclosures, we read through the rest of the policy to determine if any statements disclose the data flow. If it is not apparent and there is any uncertainty, we mark the flow as “uncertain” to avoid bias. Note that we marked 27 flows as uncertain, resulting in 153 data flows across 151 applications

**Results:** POLICHECK achieves an overall 90.8% precision (139/153) for performing flow-to-policy consistency analysis. For identifying consistencies (i.e., clear disclosures and vague disclosures), POLICHECK had 86 true positives and only 4 false positives. For incorrect disclosures, POLICHECK had 35 true positives and 3 false positives.

For omitted disclosures, POLICHECK had 18 true positives and 7 false positives, which was primarily due to incomplete policy statement extraction. The main reason for incomplete policy extraction was that the information describing the sharing and collection practices was distributed across multiple sentences and sections of the policy. The policy did not make declarative statements on their collection and sharing practices. Understanding an entire document is beyond the current limits of NLP, but this stratification also leads to an important observation. The policies for the omitted disclosure false positives were generally more difficult to read than other policies, and often required a great deal of mental effort to understand them. Therefore, these omissions can potentially be used as an indicator for poor privacy policy interpretability. We explored this direction by analyzing a select number of applications with the greatest number of omitted disclosures in our data set.

*Case Study: Omitted disclosures may also indicate confusing language in privacy policies:* A popular game application with over 100M+ downloads called ‘Ant Smasher by Best Cool & Fun Games,’ (com.bestcoolfungames.antsmasher) had 17 unique omitted disclosures. The application has an E rating, which means that it is marketed towards children, but yet it shares Ad IDs, Android IDs, and location data with advertisers and analytics providers. When validating these data flows, we found the following policy statement, which potentially discloses these practices albeit vaguely.

*For instance, whenever you access and start to interact with our Apps, we are able to identify your IP address, system configuration, browser type and other sorts of information arising from your device. We may aggregate that data in order to improve our Apps and other services we provide, but we will not exploit it commercially or disclose it without your consent, except for third-party service*



**Figure 5.8** The majority of applications have less than five data flows for each disclosure type, but a small percentage have significantly more.

*providers in order to enable the existence of our Apps and the provision of our services.*

First, they never explicitly mention the data types, but it could arguably fall under the vague umbrella phrase, “other sorts of information arising from your device.” Second, the language is unclear and potentially deceptive, because the policy initially implies that device data is not sent to third parties for commercial reasons. However, it adds an exception for enabling the “existence of their application,” which may be interpreted as the revenue from selling user data to advertisers. While our POLICHECK classified this policy as containing omitted disclosures, it is unclear whether this is actually the case and requires interpretation by a legal expert. The language that this policy uses is significantly difficult to interpret and that its behaviors should be disclosed more clearly to end-users.

### 5.5.3 Additional Case Studies

The examples in Section 5.2 provide additional real-world case-studies that demonstrate both POLICHECK’s utility, the significance of our findings, and the importance of an entity-

sensitive consistency model. In this section, we provide additional case studies from our analysis of the most inconsistent applications for each consistency type (i.e., the applications in the long tail in Figure 5.8). We analyzed each data flow, the policy statements extracted, and the privacy policy itself to validate the findings. The remainder of this section presents concrete examples.

### 5.5.3.1 Omitted Disclosures

We investigated applications in our dataset with high numbers of omitted data flows. “Survival Island Games - Survivor Craft Adventure” (`com.gamefirst.chibisurvivor`) is a game with over 500K installs on Google play. We found that the app collects the user’s location data, Android ID, and MAC address to share with advertisers and analytics providers. Its privacy policy does not discuss any details regarding data sharing. Omitted disclosures are grave concerns, especially in cases like the one above, which involves tracking the user’s physical location along with persistent identifiers.

While validating omitted disclosures, we found another application called “Cloudventure: Arcade + Editor” (`at.hakkon.pufpuf.android`) that has an omitted disclosure of Ad ID being shared with AdColony. The privacy policy is copied below in entirety, which shows the potential deceptiveness of their policy.

*Okay guys listen up, I'm forced to write this privacy policy or Google will take this APP from the store.*

*- There is an option in the app to share your level with your friends. This is made by making a screenshot of your screen and is the reason why camera permission is needed.*

*- Also this is a game so you don't want the screen to go dark while playing, right? That's why I need the phone state permission. (`android.permission.READ_PHONE_STATE`)*

*That's it, this app is not evil and I'm not selling your data to some crazy marketing company to get you filled up with spam.*

### 5.5.3.2 Incorrect Disclosures

Three of the top five applications that had the greatest number of incorrect disclosures were released by the same publisher, “Nazara Games.” Their games on Google Play have

over 57 million total downloads for 33 applications (eight of which occur in our dataset). They publish games with an E rating, which may be used by targeted towards children, but still collect a wide-array of privacy-sensitive data. From the 8 applications in our dataset, we found 95 flows originating from Nazara Games applications, of which, 75 had incorrect disclosures. Their application “Chhota Bheem Speed Racing” (`com.nazara.tinylabproductions.chhotabheem-22002`) has over 10M+ downloads and has 15 incorrect disclosures detected by POLICHECK. These flows included location data, Android IDs, Ad IDs, IMEIs, router SSIDs, and other serial numbers. Nazara Games’ applications sent this data to 14 distinct advertisers and analytics providers, such as Flurry, ironSource, and Unity3d Ads. As some of their applications targeted towards children, the mass collection and sharing of this sensitive user data is egregious. Even more so when considering that they’re sending this information is likely being used to target ads towards children.

In Nazara Games’ privacy policies, that they do not sell or rent personal information unless the user gives consent.

*Nazara does not sell or rent your Personal Information to third-parties for marketing purposes without your consent.*

As some of these applications are for children, verifiable consent is required from the child’s legal guardian according to regulations [Law03]. As discussed by prior work [Rey18], clicking a button likely does not constitute verifiable consent. For the applications that are not targeted towards children, it is unclear if consent is explicitly request or implicitly through acceptance of the policy. We leave it as future work to analyze how applications are requesting consent.

### **5.5.3.3 Ambiguous Disclosures**

“Roller Coaster Tycoon 4” (`com.atari.mobile.rct4m`) is a popular game from Atari which has over 10M downloads. We found that this application has 15 ambiguous disclosures due to their sharing of Ad IDs, Android IDs, and IMEI with advertisers and analytics providers, such as TapJoy, ironSource, and AdColony. Atari does not consider device information to be PII. However, various regulations [PEU16; Law03] identify such information as PII, as they can be to identify users over a long span of time across different applications and services. The main source of ambiguous disclosures was due to statements regarding allowing business partners to collect device identifiers, but then stating that third parties will not collect device identifiers without consent.

The “Bowmasters” game application (`com.miniclip.bowmasters`) has over 50M downloads and 12 unique ambiguous disclosures. Their policy states “We don’t give or sell your data to third parties for them to market to you,” but later it states, “On our apps, these third-party advertising companies will collect and use your data to provide you with targeted advertising.” As serving targeted advertisements is a form of marketing, this policy contradicts itself and is ambiguous in terms of the flow.

## 5.6 Limitations

POLICHECK provides an entity-sensitive and negation-sensitive flow-to-policy consistency model that formally specifies two types of consistencies and three types of inconsistencies. In so doing, it provides unprecedented analysis depth of flow-to-policy consistency, and our findings in Section 5.2 and Section 5.5 demonstrate the utility and value of such analysis. However, POLICHECK inherits the technical limitations of the tools that it is built upon. As POLICHECK uses our PolicyLint tool for policy statement extraction, it inherits PolicyLint’s NLP limitations, as discussed in Section 4. Similarly, as PolicyLint uses AppCensus [Appb] for data flow analysis, it also inherits AppCensus’ limitations, such as achieving code coverage and limitations in analysis scope in terms of the types of privacy-sensitive data tracked during analysis. Further, POLICHECK also relies on the completeness of its ontologies and its resolution of domains and IP addresses to entities (e.g., first-party vs. third-party). The current implementation of POLICHECK disregards data flows to IP addresses that are not resolvable using a reverse-DNS search or domains that resolve to domains for web hosting services (e.g., Amazon Web Services), as we cannot determine the entity behind the domain. Improving the completeness of ontologies and performance of entity resolution would allow POLICHECK to reason over a more exhaustive set of data flows.

In addition to POLICHECK’s technical limitations, POLICHECK introduces several open questions regarding the interpretation of regulations and enforceability of inconsistency types. First, POLICHECK assumes that applications are required to disclose their third-party data flows directly within their own privacy policy. POLICHECK does not currently follow the links embedded within an application’s privacy policy (if any) to the privacy policies of other companies with which the application partners. To the best of our knowledge, there is no legal precedent that rules on whether solely providing a link to a third-party privacy policy without discussing that information is shared with third-parties is sufficient for disclosure.

However, we believe our assumption is reasonable due to guidelines released by the FTC on disclosing third-party data flows [Ftcb] and language within the CALOPPA [Law03] that discusses that “operators” need to disclose whether third-parties collect information when the user is using their service. Second, POLICHECK assumes that all types of inconsistencies would be of interest to regulatory agencies and potentially actionable items. In the past, the FTC has taken action in response to incorrect disclosures [FTCb] and omitted disclosures [FTCa]. However, to the best of our knowledge, there has been no legal precedent regarding whether regulatory agencies would take action as a result of ambiguous disclosures. Nonetheless, we believe that ambiguous disclosures may potentially fall under the FTC’s definition of deceptive practices [Ftca].

## 5.7 Related Work

In recent years, there has been an increased focus on analyzing flow-to-policy inconsistencies in mobile applications. The works differ in how app behavioral flows and privacy policies are analyzed. While [Sla16; Zim17b; Yu16] use Android’s application program interface (API) calls to evaluate privacy breaches, Wang et al. [Wan18] extended the taint sources to include sensitive data entered through an app’s UI. For policy analysis, Zimmeck et al. [Zim17b] and Yu et al. [Yu16] rely on keyword-based approaches, of using bi-grams and verb modifiers respectively, to infer the privacy policies, while Slavin et al. [Sla16] and Wang et al. [Wan18] use crowdsourced ontologies for policy analysis. POLICHECK makes significant advancement over all these prior works by considering DNS domains of data-receiving entity for comprehensive entity-sensitive analysis. Accuracy of the analysis is further improved by considering entities and positivity/negativity of statements, while also accounting for different semantic granularities and internal contradictions. Our empirical results (Section 5.5) further demonstrate the effectiveness of these capabilities.

Other recent research has focused on analyzing specific application categories, such as those designed for families, for compliance and privacy violations [Rey17; Rey18; Oko19; Han19]. Similar to POLICHECK, they use dynamic analysis to identify sensitive flows along with the entities receiving the data. However, their policy analysis is either manual [Rey17; Rey18; Han19] or semi-automatic based on keyword searches [Oko19]. While these approaches potentially worked for a category of apps with explicit requirements, they are severely limited in precision and scale for broader categories as targeted by our work. In

contrast, POLICHECK utilizes an automated, comprehensive policy analysis that improves precision by considering additional capabilities, such as semantic granularities and contradictions within policies.

Numerous works focus on the automated analysis of privacy policies. Privee [ZB14] uses natural language processing for deriving answers to a limited set of binary questions from the privacy policies, while Hermes [SR09] applies topic modeling to reduce ambiguity in privacy policies. PrivacyCheck [Zae13] use data mining models to analyze the privacy policies to automatically extract their graphical summaries representing what information is used and how. A more recent work, Polisis [Har18], provides an automated policy analysis tool that uses deep learning to infer types of data collected and the reason for collection. While it provides alternate approaches for comprehensive policy analysis, it is limited in its handling of negations and exclusions in text. In Chapter 4, we showed that a considerable number of policies include negations and exclusions that would be missed by prior works. Our policy analysis is built on top of our PolicyLint tool and hence improves precision over prior art. Moreover, none of the works focus on the evaluation of app behavior, which is a core component for our entity-sensitive flow-to-policy analysis.

There is rich body of work to understand [Bar10; Fel11b; Pen12; Wij15] and bridge [Ros13; Zha13] the gap between application behaviors and users' understanding of these behaviors. POLICHECK differs from these works in its focus of analyzing privacy policy to behavior inconsistencies.

## 5.8 Conclusion

Privacy threats from mobile applications are arguably a greater risk than malware for most smartphone users. While the last decade has produced many static and dynamic analysis to detect when mobile applications send privacy-sensitive data to the network, such data flows are not privacy leaks if they are disclosed in a privacy policy. Recently, several efforts have sought to more fully automate the detection of privacy leaks by contrasting data flows with the application's privacy policy. However, these works have a fundamental limitation: they do not consider the entity receiving the data (e.g., first-party vs. third-party). In this chapter, we proposed POLICHECK and an entity-sensitive flow-to-policy consistency model. We used POLICHECK to study 13,796 applications, comparing their data flows to their policy statements. We find significant evidence of omitted, incorrect, and ambiguous disclosures,

many of which are only possible to identify by considering the entity. As such, POLICHECK provides the highest-precision method to date to determine if apps properly disclose their privacy-sensitive behaviors.

## CHAPTER

# 6

## CONCLUSION AND FUTURE DIRECTIONS

In this dissertation, we characterized and identified privacy risks associated with disclosing privacy-sensitive user data to applications. In Chapter 3, we characterized the scope of privacy-sensitive user data sources to understand what types of data applications are requesting from users. Our analysis uncovered several concerning developer practices, including insecure exposure of account passwords and privacy violations due to non-consensual disclosures of privacy-sensitive user input to third parties. In Chapter 4, we characterized the semantics of sharing and collection statements within privacy policies to understand how privacy practices are being disclosed. We demonstrated the importance of holistic analysis of privacy policies through our identification and formalization of self-contradictory policy statements. We performed a large-scale study on the privacy policies for 11,430 Android applications and found that around 14.2% have potentially deceptive and ambiguous privacy policies due to self-contradictory statements. In Chapter 5, we combine insights gained from our prior studies to provide a formal specification for an

entity-sensitive and negation-sensitive flow-to-policy consistency model. We performed a large-scale study on 13,796 Android applications and found that up to 42.4% of applications either incorrectly disclose or omit disclosing their privacy-sensitive data flows. We further demonstrated that entity-insensitive and negation-insensitive approaches would falsely classify up to 38.4% of applications as having privacy-sensitive data flows consistent with their privacy policies.

The findings from our empirical studies highlight significant privacy risks associated with exposing privacy-sensitive user data to applications and provide concrete examples of such cases that impact tens-to-hundreds of millions of users. Our results show the poor state of privacy disclosures for Android applications, which provides evidence that the self-regulation of privacy policy disclosures is insufficient and requires additional oversight and auditing from application markets and regulatory agencies. In addition, our findings introduce several future directions for research, including:

- **Evaluating Flow-to-Policy Consistency with Privacy-sensitive User Input Data:** We demonstrate in Chapter 3 that applications are collecting a wide-range of privacy-sensitive user input. When studying flow-to-policy consistency in Chapter 5, we only considered data flows reported by AppCensus [Appb], which limits analysis to data obtained through the operating system’s well-defined API calls. Therefore, an open research question is on whether applications are disclosing their collection and sharing of privacy-sensitive user input. Tracking where privacy-sensitive user input flows requires additional research and significant engineering efforts to incorporate the output from UiRef into analysis techniques for tracking information flows. For static analysis, UiRef’s resolved user inputs would need to be linked to variables in the code using the resource identifiers of the input widgets, as proposed by prior efforts [Hua15]. Dynamic analysis would require providing the correct examples of user inputs into each input widget, identifying when such information is transmitted from the device, and ensuring code coverage during application exploration. While prior work has explored this area [Ras13], paradigm changes in the design of applications (e.g., fragments) may introduce additional research challenges. After the data flows are extracted, POLICHECK could consume the output and report flow-to-policy consistency results for user input.
- **Assessing Requests for Consent:** In Chapter 5, we found several cases of privacy policy statements that discuss only collecting or sharing privacy-sensitive user data if

they have the user's consent. Obtaining user consent is an important aspect assessing privacy-sensitive user data requests and is discussed in various regulations, such as GDPR [PEU16] and COPPA [USC16]. Our findings introduce several future research directions into how applications are requesting and obtaining consent from users. A few potential research questions are as follows: (1) Are applications requesting consent clearly and in an easily comprehensible manner? (2) Do users understand the privacy impact of granting consent? (3) Are applications ensuring that data transfer only occurs after consent is given?

- **Enhancing Privacy Policy Analysis:** While we advance the state-of-the-art in reasoning over privacy policies, there are several natural language processing and information extraction tasks that would allow for a deeper and more holistic analysis. First, coreference resolution would increase completeness of information extraction by allowing entities and data type mentions to be tracked over multiple sentences. For example, consider the sentences, “We collect device identifiers. It may be shared with advertisers.” In this case, coreference resolution is required to resolve that the word “it” refers to “device identifiers” in the prior sentence. Second, additional natural language processing techniques, such as semantic role labeling, can be used to extract the purpose of collection and sharing to allow for more precise reasoning involving the purpose of collection. Future research can link the purpose behind collection to data flows to determine whether privacy violations occur. Finally, additional information extraction techniques are needed to extract definitions from privacy policies to construct application-specific data and entity ontologies. Application-specific data ontologies would allow reasoning over terms that specific privacy policies define to group data objects. Application-specific entity ontologies would allow reasoning over organizational-specific relationships, such a subsidiaries and parent organizations.
- **Enforcing Privacy Guarantees:** Our studies in Chapter 3 and Chapter 5 identify several cases of privacy violations involving how applications are handling privacy-sensitive user data. Many of these data flows could be prevent by integrating context-aware privacy-oriented protection mechanisms within the operating systems. For example, operating systems, such as Android and iOS, could prevent data flows that are not disclosed by the privacy policy by refusing to return privacy-sensitive user data to the application or by blocking transmission. Furthermore, we found several

cases of privacy violations involving applications targeted towards children. Similarly, operating systems could implement global parental controls that prevent applications from tracking minors and by streamlining consent requests to the guardian's device.

- **Assessing Privacy Risks of other Ecosystems:** The studies in this dissertation were primarily focused on Android applications and their corresponding privacy policies. While we cannot claim that our findings will certainly generalize to policies from other domains (e.g., iOS, web, IoT), we hypothesize that similar privacy violations likely occur across-the-board, as policies are written for all platforms in largely the same ways to describe data types collected by all platforms. Future research is needed to analyze the privacy risks for third-party applications for these other platforms and domains.
- **Usability of Privacy Policy Authoring:** In Chapter 4, we reached out to developers that had self-contradictory policies and several responded positively stating that they fixed their policy. This demonstrates that some of the findings are not intentionally deceptive or malicious, but rather mistakes made by developers when writing their privacy policy. Future research is needed to identify the challenges that developers face when writing privacy policies, such as whether they understand that the third-party SDKs that they are including collect privacy-sensitive data and that they are responsible for disclosing these behaviors within their privacy policy. Additionally, automated privacy policy generation tools, such as AutoPPG [Yu15], could also potentially be used to help developers create clear, easily comprehensible, non-contradictory privacy policies that disclose all of their data flows.

## BIBLIOGRAPHY

- [Appa] *2017 Retrospective: A Monumental Year for the App Economy*. <https://www.appannie.com/en/insights/market-data/app-annie-2017-retrospective/>. 2018.
- [Aaf13] Aafer, Y. et al. “DroidApiMiner: Mining API-level Features for Robust Malware Detection in Android”. *Proceedings of the International Conference on Security and Privacy in Communication Systems (SecureComm)*. 2013.
- [And17] Andow, B. et al. “UiRef: Analysis of Sensitive User Inputs in Android Applications”. *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 2017.
- [And19] Andow, B. et al. “PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play”. *Proceedings of the USENIX Security Symposium*. 2019.
- [And19] Android Studio. *UI/Application Exerciser Monkey*. <https://developer.android.com/studio/test/monkey.html>. Accessed: May 15, 2019. 2019.
- [Apk] *ApkTool*. <http://ibotpeaches.github.io/Apktool>. 2017.
- [Appb] *AppCensus AppSearch*. <https://search.appcensus.io/>.
- [Arp12] Arp, D. et al. “DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket”. *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS)*. 2012.
- [Arz14a] Arzt, S. et al. “A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks”. *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS)*. 2014.
- [Arz14b] Arzt, S. et al. “FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps”. *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 2014.
- [Au12] Au, K. W. Y. et al. “PScout: Analyzing the Android Permission Specification”. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2012.

- [Avd15] Avdiienko, V. et al. "Mining Apps for Abnormal Usage of Sensitive Data". *Proceedings of the IEEE International Conference on Software Engineering (ICSE)*. 2015.
- [AN13] Azim, T. & Neamtiu, I. "Targeted and Depth-first Exploration for Systematic Testing of Android Apps". *Proceedings of the ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA)*. 2013.
- [Bac16a] Backes, M. et al. "On Demystifying the Android Application Framework: Re-Visiting Android Permission Specification Analysis". *Proceedings of the USENIX Security Symposium*. 2016.
- [Bac16b] Backes, M. et al. "Reliable Third-Party Library Detection in Android and Its Security Applications". *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2016.
- [Bar10] Barrera, D. et al. "A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android". *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2010.
- [Bar15] Bartunov, S. et al. "Breaking Sticks and Ambiguities with Adaptive Skip-gram". *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2015.
- [Bas16] Bashir, M. A. et al. "Tracing Information Flows Between Ad Exchanges Using Retargeted Ads". *Proceedings of the USENIX Security Symposium*. 2016.
- [BB17] Bhatia, J. & Breaux, T. D. "A Data Purpose Case Study of Privacy Policies". *Proceedings of the IEEE International Requirements Engineering Conference (RE)*. 2017.
- [BB18] Bhatia, J. & Breaux, T. D. "Semantic Incompleteness in Privacy Policy Goals". *Proceedings of the IEEE International Requirements Engineering Conference (RE)*. 2018.
- [Bha16] Bhatia, J. et al. "A Theory of Vagueness and Privacy Risk Perception". *Proceedings of the IEEE International Requirements Engineering Conference (RE)*. 2016.
- [Bho14] Bhoraskar, R. et al. "Brahmastra: Driving Apps to Test the Security of Third-Party Components". *Proceedings of the USENIX Security Symposium*. 2014.

- [Bow17] Bowers, J. et al. “Regulators, Mount Up! Analysis of Privacy Policies for Mobile Money Services”. *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*. 2017.
- [Bow19] Bowers, J. et al. “Characterizing Security and Privacy Practices in Emerging Digital Credit Applications”. *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 2019.
- [BR13] Breaux, T. D. & Rao, A. “Formal Analysis of Privacy Requirements Specifications for Multi-tier Applications”. *Proceedings of the IEEE International Requirements Engineering Conference (RE)*. 2013.
- [BS04] Breaux, T. D. & Schaub, F. “Scaling Requirements Extraction to the Crowd: Experiments with Privacy Policies”. *Proceedings of the IEEE International Requirements Engineering Conference (RE)*. 2004.
- [Bre15] Breaux, T. D. et al. “Detecting Repurposing and Over-Collection in Multi-party Privacy Requirements Specifications”. *Proceedings of the IEEE International Requirements Engineering Conference (RE)*. 2015.
- [Bro06] Brodie, C. A. et al. “An Empirical Study of Natural Language Parsing of Privacy Policy Rules Using the SPARCLE Policy Workbench”. *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*. 2006.
- [Bug13] Bugiel, S. et al. “Flexible and Fine-Grained Mandatory Access Control on Android for Diverse Security and Privacy Policies”. *Proceedings of the USENIX Security Symposium*. 2013.
- [Cab18] Cabañas, J. G. et al. “Unveiling and Quantifying Facebook Exploitation of Sensitive Personal Data for Advertising Purposes”. *Proceedings of the USENIX Security Symposium*. 2018.
- [Cha13] Chakradeo, S. et al. “MAST: Triage for Market-scale Mobile Malware Analysis”. *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 2013.
- [Com00] Commission, F. T. *Privacy Online: Fair Information Practices in the Electronic Marketplace: A Federal Trade Commission Report to Congress*. 2000.
- [Con17] Continella, A. et al. “Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis”. *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS)*. 2017.

- [Cos12] Costante, E. et al. “A Machine Learning Solution to Assess Privacy Policy Completeness”. *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*. 2012.
- [Cos13] Costante, E. et al. “What Websites Know About You: Privacy Policy Analysis Using Information Extraction”. *Proceedings of the International Workshop on Data Privacy Management and Autonomous Spontaneous Security (DPM)*. 2013.
- [Cra02] Cranor, L. F. et al. “The Platform for Privacy Preferences 1.0 (P3P1.0) Specification”. *W3C Recommendation 16* (2002).
- [Cra06] Cranor, L. F. et al. “User Interfaces for Privacy Agents”. *ACM Transactions on Computer-Human Interaction (TOCHI)* **13.2** (2006).
- [Cra16] Cranor, L. F. et al. “A Large-Scale Evaluation of US Financial Institutions’ Standardized Privacy Notices”. *ACM Transactions on the Web* (2016).
- [Dav12] Davis, B. et al. “I-ARM-Droid: A Rewriting Framework for In-App Reference Monitors for Android Applications”. *Proceedings of IEEE Mobile Security Technologies Workshop (MoST)*. 2012.
- [Deg18] Degeling, M. et al. “We Value Your Privacy... Now Take Some Cookies: Measuring the GDPR’s Impact on Web Privacy”. *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS)*. 2018.
- [Dem16] Demetriou, S. et al. “Free for All! Assessing User Data Exposure to Advertising Libraries on Android”. *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*. 2016.
- [Mas] *Driving apps pulled from Google Play for reportedly installing Android malware over 560,000 times*. <https://mashable.com/article/apps-malware-google-play-store-breach>. 2018.
- [Ege11] Egele, M. et al. “PiOS: Detecting Privacy Leaks in iOS Applications”. *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*. 2011.
- [Ege13] Egele, M. et al. “An Empirical Study of Cryptographic Misuse in Android Applications”. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2013.
- [Eij19] Eijk, R. van et al. “The Impact of User Location on Cookie Notices (Inside and Outside of the European Union)”. *Workshop on Technology and Consumer Protection (ConPro)*. 2019.

- [Enc09] Enck, W. et al. "On Lightweight Mobile Phone Application Certification". *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2009.
- [Enc10] Enck, W. et al. "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones". *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2010.
- [Enc11] Enck, W. et al. "A Study of Android Application Security". *Proceedings of the USENIX Security Symposium*. 2011.
- [Eri] *Ericsson Mobility Report*. <https://www.ericsson.com/assets/local/mobility-report/documents/2018/ericsson-mobility-report-june-2018.pdf>. 2018.
- [Eva00] Evans, D. "Annotation-Assisted Lightweight Static Checking". *The First International Workshop on Automated Program Analysis, Testing and Verification*. 2000.
- [EL01] Evans, D. & Larochelle, D. "Statically Detecting Likely Buffer Overflow Vulnerabilities". *Proceedings of the USENIX Security Symposium*. 2001.
- [EL02] Evans, D. & Larochelle, D. "Improving Security Using Extensible Lightweight Static Analysis". *IEEE Software* (2002).
- [Eva94] Evans, D. et al. "LCLint: A Tool for Using Specifications to Check Code". *Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*. 1994.
- [Eva17] Evans, M. C. et al. "An Evaluation of Constituency-based Hyponymy Extraction from Privacy Policies". *Proceedings of the IEEE International Requirements Engineering Conference (RE)*. 2017.
- [Fah12] Fahl, S. et al. "Why Eve and Mallory love Android: An analysis of Android SSL (in)security". *Proceedings of the ACM conference on Computer and Communications Security (CCS)*. 2012.
- [Ftca] *Federal Trade Commission Act: Section 5: Unfair or Deceptive Acts or Practices*. <https://www.federalreserve.gov/boarddocs/supmanual/cch/ftca.pdf>.
- [Fel11a] Felt, A. P. et al. "A Survey of Mobile Malware in the Wild". *Proceedings of the ACM Workshop on Security and Privacy in Mobile Devices (SPSM)*. 2011.
- [Fel11b] Felt, A. P. et al. "Android Permissions Demystified". *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2011.

- [Fen15] Feng, H. et al. “LinkDroid: Reducing Unregulated Aggregation of App Usage Behaviors”. *Proceedings of the USENIX Security Symposium*. 2015.
- [FWR14] Fengguo Wei Sankardas Roy, X. O. & Robby. “Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps”. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2014.
- [Fra16] Fratantonio, Y. et al. “Triggerscope: Towards Detecting Logic Bombs in Android Applications”. *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2016.
- [FTCa] FTC. *In the Matter of Goldenshores Technologies, LLC, and Erik M. Geidl, FTC No. 132 3087*. <https://www.ftc.gov/enforcement/cases-proceedings/132-3087/goldenshores-technologies-llc-erik-m-geidl-matter>.
- [FTCb] FTC. *In the Matter of Snapchat, Inc., FTC No. 132 3078*. <https://www.ftc.gov/enforcement/cases-proceedings/132-3078/snapchat-inc-matter>.
- [Gar] *Gartner Says Worldwide Sales of Smartphones Recorded First Ever Decline During the Fourth Quarter of 2017*. <https://www.gartner.com/en/newsroom/press-releases/2018-02-22-gartner-says-worldwide-sales-of-smartphones-recorded-first-ever-decline-during-the-fourth-quarter-of-2017>. 2018.
- [Gib12] Gibler, C. et al. “AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale”. *Proceedings of the International Conference on Trust and Trustworthy Computing (TRUST)*. 2012.
- [Glu16] Gluck, J. et al. “How Short is Too Short? Implications of Length and Framing on the Effectiveness of Privacy Notices”. *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*. 2016.
- [Anda] *Google Android Annual Security Report 2016*. [https://source.android.com/security/reports/Google\\_Android\\_Security\\_2016\\_Report\\_Final.pdf](https://source.android.com/security/reports/Google_Android_Security_2016_Report_Final.pdf). 2016.
- [Andb] *Google Android Annual Security Report 2017*. [https://source.android.com/security/reports/Google\\_Android\\_Security\\_2017\\_Report\\_Final.pdf](https://source.android.com/security/reports/Google_Android_Security_2017_Report_Final.pdf). 2017.
- [Gor15] Gordon, M. I. et al. “Information Flow Analysis of Android Applications in Droid-Safe”. *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS)*. 2015.
- [Gor14] Gorla, A. et al. “Checking App Behavior Against App Descriptions”. *Proceedings of the International Conference on Software Engineering (ICSE)*. 2014.

- [Gra12a] Grace, M. et al. “RiskRanker: Scalable and Accurate Zero-day Android Malware Detection”. *Proceedings of the International Conference on Mobile Systems, Applications and Services (MobiSys)*. 2012.
- [Gra12b] Grace, M. et al. “Unsafe Exposure Analysis of Mobile In-App Advertisements”. *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 2012.
- [Tec] *Half a million Android users tricked into downloading malware from Google Play*. <https://techcrunch.com/2018/11/20/half-a-million-android-users-tricked-into-downloading-malware-from-google-play/>. 2018.
- [Han19] Han, C. et al. “Do You Get What You Pay For? Comparing the Privacy Behaviors of Free vs. Paid Apps”. *Workshop on Technology and Consumer Protection (ConPro)*. 2019.
- [Han13a] Han, J. et al. “Comparing Mobile Privacy Protection through Cross-Platform Applications”. *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS)*. 2013.
- [Han13b] Han, J. et al. “Launching Generic Attacks on iOS with Approved Third-party Applications”. *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS)*. 2013.
- [Har18] Harkous, H. et al. “Polisis: Automated Analysis and Presentation of Privacy Policies Using Deep Learning”. *Proceedings of the USENIX Security Symposium*. 2018.
- [Has18] Hassan, W. U. et al. “Analysis of Privacy Protections in Fitness Tracking Social Networks -or- You can run, but can you hide?” *Proceedings of the USENIX Security Symposium*. 2018.
- [Hea92] Hearst, M. A. “Automatic Acquisition of Hyponyms from Large Text Corpora”. *Proceedings of the Conference on Computational Linguistics (COLING)*. 1992.
- [Heu14] Heuser, S. et al. “ASM: A Programmable Interface for Extending Android Security”. *Proceedings of the USENIX Security Symposium*. 2014.
- [Sym] *Hidden App Malware Found on Google Play*. <https://www.symantec.com/blogs/threat-intelligence/hidden-app-malware-google-play>. 2018.
- [HM17] Honnibal, M. & Montani, I. “spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks, and Incremental Parsing”. *To appear* (2017).

- [Hos18] Hosseini, M. B. et al. “Inferring Ontology Fragments from Semantic Role Typing of Lexical Variants”. *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*. 2018.
- [Hua14] Huang, J. et al. “AsDroid: Detecting Stealthy Behaviors in Android Applications by User Interface and Program Behavior Contradiction”. *Proceedings of the International Conference on Software Engineering (ICSE)*. 2014.
- [Hua15] Huang, J. et al. “SUPOR: Precise and Scalable Sensitive User Input Detection for Android Apps”. *Proceedings of the USENIX Security Symposium*. 2015.
- [Appc] *iOS Developers Ship 29% Fewer Apps in 2017, the First Ever Decline – And More Trends to Watch*. <https://blog.appfigures.com/ios-developers-ship-less-apps-for-first-time/>. 2018.
- [JP04] Jensen, C. & Potts, C. “Privacy Policies as Decision-Making Tools: An Evaluation of Online Privacy Notices”. *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems (CHI)*. 2004.
- [Jia13] Jia, L. et al. “Run-time Enforcement of Information-flow Properties on Android”. *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*. 2013.
- [Joh78] Johnson, S. C. “Lint, a C Program Checker”. *Computer Science Technical Report*. 1978.
- [Kel09] Kelley, P. G. et al. “A “Nutrition Label” for Privacy”. *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*. 2009.
- [Law03] Law, C. S. *California Online Privacy Protection Act (CalOPPA)*. 2003.
- [Les86] Lesk, M. “Automatic Sense Disambiguation using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone”. *Proceedings of the International Conference on Systems Documentation (SIGDOC)*. 1986.
- [Li15] Li, L. et al. “IccTA: Detecting Inter-Component Privacy Leaks in Android Apps”. *Proceedings of the IEEE International Conference on Software Engineering (ICSE)*. 2015.
- [Lin14] Lin, J. et al. “Modeling Users’ Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings”. *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*. 2014.

- [Liu14] Liu, F. et al. “A Step Towards Usable Privacy Policy: Automatic Alignment of Privacy Statements”. *Proceedings of the International Conference on Computational Linguistics (COLING)*. 2014.
- [Ma16] Ma, Z. et al. “LibRadar: Fast and Accurate Detection of Third-Party Libraries in Android Apps”. *Proceedings of the International Conference on Software Engineering (ICSE)*. 2016.
- [Mar14] Marella, A. et al. “Assessing Privacy Awareness from Browser Plugins” (2014).
- [Ftcb] *Marketing Your Mobile App: Get It Right from the Start*. <https://www.ftc.gov/tips-advice/business-center/guidance/marketing-your-mobile-app-get-it-right-start>.
- [MC08] McDonald, A. M. & Cranor, L. F. “The Cost of Reading Privacy Policies”. *I/S Journal of Law and Policy for the Information Society (ISJLP)* 4 (2008).
- [Mik13] Mikolov, T. et al. “Distributed Representations of Words and Phrases and their Compositionality”. *Advances in Neural Information Processing Systems (NIPS)*. 2013.
- [Mil06] Milne, G. R. et al. “A Longitudinal Assessment of Online Privacy Notice Readability”. *Journal of Public Policy & Marketing (JPP&M)* 25.2 (2006).
- [Pew] *Mobile Fact Sheet*. <http://www.pewinternet.org/fact-sheet/mobile/>. 2018.
- [Goa] *Modernizing OAuth interactions in Native Apps for Better Usability and Security*. <https://developers.googleblog.com/2016/08/modernizing-oauth-interactions-in-native-apps.html>. 2016.
- [NE13] Nadkarni, A. & Enck, W. “Preventing Accidental Data Disclosure in Modern Operating Systems”. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2013.
- [Nad16] Nadkarni, A. et al. “Practical DIFC Enforcement on Android”. *Proceedings of the USENIX Security Symposium*. 2016.
- [Nan15] Nan, Y. et al. “UIPicker: User-Input Privacy Identification in Mobile Applications”. *Proceedings of the USENIX Security Symposium*. 2015.
- [Nan18] Nan, Y. et al. “Finding Clues for Your Secrets: Semantics-Driven, Learning-Based Privacy Discovery in Mobile Apps”. *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS)*. 2018.

- [Mal] *New Android Trojan malware discovered in Google Play*. <https://blog.malwarebytes.com/cybercrime/2017/11/new-trojan-malware-discovered-google-play/>. 2017.
- [Ng07] Ng, M. K. et al. "On the Impact of Dissimilarity Measure in k-Modes Clustering Algorithm". *Proceedings of the IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2007.
- [Ngu19] Nguyen, D. C. et al. "Short Text, Large Effect: Measuring the Impact of User Reviews on Android App Security & Privacy". *Proceedings of the IEEE Symposium on Security and Privacy*. 2019.
- [Nor15] Norton, T. B. "Crowdsourcing Privacy Policy Interpretation". *Proceedings of the Research Conference on Communications, Information, and Internet Policy (TPRC)* (2015).
- [Appd] *Number of Android apps on Google Play*. <https://www.appbrain.com/stats/number-of-android-apps>. 2018.
- [OM12] Oberheide, J. & Miller, C. "Dissecting the Android Bouncer". *SummerCon2012* (2012).
- [Oct12] Octeau, D. et al. "Retargeting Android Applications to Java Bytecode". *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. 2012.
- [Oct13] Octeau, D. et al. "Effective Inter-component Communication Mapping in Android with EPPIC: An Essential Step Towards Holistic Security Analysis". *Proceedings of the USENIX Security Symposium*. 2013.
- [Oko19] Okoyomon, E. et al. "On the Ridiculousness of Notice and Consent: Contradictions in App Privacy Policies". *Workshop on Technology and Consumer Protection (ConPro)*. 2019.
- [Olt18] Oltrogge, M. et al. "The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators". *Proceedings of the IEEE Symposium on Security and Privacy*. 2018.
- [Pan18] Pan, X. et al. "FlowCog: Context-aware Semantics Extraction and Analysis of Information Flow Leaks in Android Apps". *Proceedings of the USENIX Security Symposium*. 2018.
- [Pan13] Pandita, R. et al. "WHYPER: Towards Automating Risk Assessment of Mobile Applications". *Proceedings of the USENIX Security Symposium*. 2013.

- [PEU16] Parliament, E. & European Union, C. of the. *General Data Protection Regulation (EU) 2016/679 (“GDPR”)*. 2016.
- [Pci] *PCI Security Standards Council*. <https://www.pcisecuritystandards.org/>. 2016.
- [Pea12] Pearce, P. et al. “AdDroid: Privilege Separation for Applications and Advertisers in Android”. *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS)*. 2012.
- [Pen12] Peng, H. et al. “Using Probabilistic Generative Models for Ranking Risks of Android Apps”. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2012.
- [Goo] *Privacy Policy Guidance*. <https://developers.google.com/actions/policies/privacy-policy-guide>. 2018.
- [Pri] *Privacy Rights Clearinghouse Data Breaches*. <https://www.privacyrights.org/data-breaches?TITLE=>. 2016.
- [Qu14] Qu, Z. et al. “AutoCog: Measuring the Description-to-permission Fidelity in Android Applications”. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2014.
- [Ram14] Ramanath, R. et al. “Identifying Relevant Text Fragments to Help Crowdsource Privacy Policy Annotations”. *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing (HCOMP)*. 2014.
- [Rao16] Rao, A. et al. “Expecting the Unexpected: Understanding Mismatched Privacy Expectations Online”. *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*. 2016.
- [Ras13] Rastogi, V. et al. “AppsPlayground: Automatic Large-scale Dynamic Analysis of Android Applications”. *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)*. 2013.
- [Raz15] Razaghpanah, A. et al. “Haystack: A Multi-Purpose Mobile Vantage Point in User Space”. *arXiv preprint arXiv:1510.01419* (2015).
- [Raz18] Razaghpanah, A. et al. “Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem”. *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2018.

- [Rea19] Reardon, J. et al. “50 Ways to Leak Your Data: An Exploration of Apps’ Circumvention of the Android Permission System”. *Proceedings of the USENIX Security Symposium*. 2019.
- [Rei15] Reidenberg, J. R. et al. “Disagreeable Privacy Policies: Mismatches between Meaning and Users’ Understanding”. *Berkeley Technology Law Journal (BTLJ)* **30** (2015), p. 39.
- [Ren16] Ren, J. et al. “ReCon: Revealing and Controlling Privacy Leaks in Mobile Network Traffic”. *Proceedings of the International Conference on Mobile Systems, Applications and Services (MobiSys)*. 2016.
- [Ren18] Ren, J. et al. “Bug fixes, Improvements, ... and Privacy Leaks: A Longitudinal Study of PII Leaks Across Android App Versions”. *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS)*. 2018.
- [Rey17] Reyes, I. et al. ““Is Our Children’s Apps Learning?” Automatically Detecting COPPA Violations”. *Workshop on Technology and Consumer Protection (Con-Pro)*. 2017.
- [Rey18] Reyes, I. et al. ““Won’t Somebody Think of the Children?” Examining COPPA Compliance at Scale”. *Proceedings on Privacy Enhancing Technologies (PETS)*. 2018.
- [RK13] Roesner, F. & Kohno, T. “Securing Embedded User Interfaces: Android and Beyond”. *Proceedings of the USENIX Security Symposium*. 2013.
- [Ros13] Rosen, S. et al. “AppProfiler: A Flexible Method of Exposing Privacy-related Behavior in Android Applications to End Users”. *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)*. 2013.
- [Sar12] Sarma, B. P. et al. “Android Permissions: A Perspective Combining Risks and Benefits”. *Proceedings of the ACM symposium on Access Control Models and Technologies (SACMAT)*. 2012.
- [Sat17] Sathyendra, K. M. et al. “Identifying the Provision of Choices in Privacy Policy Text”. *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*. 2017.
- [Sch15] Schaub, F. et al. “A Design Space for Effective Privacy Notices”. *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*. 2015.
- [She12] Shekhar, S. et al. “Adsplit: Separating Smartphone Advertising from Applications”. *Proceedings of the USENIX Security Symposium*. 2012.

- [She15] Shengqian Yang and Hailong Zhang and Haowei Wu and Yan Wang and Dacong Yan and Atanas Rountev. “Static Window Transition Graphs for Android”. *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2015.
- [Sla16] Slavin, R. et al. “Toward a Framework for Detecting Privacy Policy Violations in Android Application Code”. *Proceedings of the International Conference on Software Engineering (ICSE)*. 2016.
- [SC13] Smalley, S. & Craig, R. “Security Enhanced (SE) Android: Bringing Flexible MAC to Android”. *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS)*. 2013.
- [Zen] *Smartphone penetration to reach 66% in 2018*. <https://www.zenithmedia.com/smartphone-penetration-reach-66-2018/>. 2018.
- [Appe] *Spotlight on Consumer App Usage*. [http://files.appannie.com.s3.amazonaws.com/reports/1705\\_Report\\_Consumer\\_App\\_Usage\\_EN.pdf](http://files.appannie.com.s3.amazonaws.com/reports/1705_Report_Consumer_App_Usage_EN.pdf). 2017.
- [SR09] Stamey, J. W. & Rossi, R. A. “Automatically Identifying Relations in Privacy Policies”. *Proceedings of the ACM International Conference on Design of Communication (SIGDOC)*. 2009.
- [Sun16] Sun, M. et al. “TaintArt: A Practical Multi-Level Information-Flow Tracking System for Android Runtime”. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2016.
- [Sur12] Suri, N. R. et al. “An Algorithm for Mining Outliers in Categorical Data through Ranking”. *Proceedings of the IEEE International Conference on Hybrid Intelligent Systems (HIS)*. 2012.
- [TR14] Tripp, O. & Rubin, J. “A Bayesian Approach to Privacy Enforcement in Smartphones”. *Proceedings of the USENIX Security Symposium*. 2014.
- [Tsa07] Tsai, J. et al. “The Effect of Online Privacy Information on Purchasing Behavior: An Experimental Study”. *Proceedings of the Workshop on the Economics of Information Security (WEIS)*. 2007.
- [Uir] *UiRef Project Website*. <https://wspr.csc.ncsu.edu/uiref/>. 2017.
- [US65] United States, S. C. of the. *Griswold v. Connecticut (381 U.S. 479)*. 1965.
- [USC99] United States Congress, 106th. *Gramm-Leach-Bliley Act (Public Law 106-102, 113 Statute 1338)*. 1999.

- [USC70] United States Congress, 91st. *Fair Credit Reporting Act (Public Law 91-508)*. 1970.
- [USC16] United States Congress, 91st. *Children’s Online Privacy Protection Act (COPPA)*. *Public Law 105-277*. 2016.
- [Vid11] Vidas, T. et al. “Curbing Android Permission Creep”. *Proceedings of the Workshop on Web 2.0 Security and Privacy (W2SP)*. 2011.
- [Vie14] Viennot, N. et al. “A Measurement Study of Google Play”. *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*. 2014.
- [Wan18] Wang, X. et al. “GUILeak: Tracing Privacy Policy Claims on User Input Data for Android Applications”. *Proceedings of the International Conference of Software Engineering (ICSE)*. 2018.
- [WB90] Warren, S. D. & Brandeis, L. D. “The Right to Privacy”. *Harvard Law Review* (1890), pp. 193–220.
- [Wij15] Wijesekera, P. et al. “Android Permissions Remystified: A Field Study on Contextual Integrity”. *Proceedings of the USENIX Security Symposium*. 2015.
- [Wil16a] Wilson, S. et al. “Crowdsourcing Annotations for Websites’ Privacy Policies: Can It Really Work?” *Proceedings of the International Conference on World Wide Web (WWW)*. 2016.
- [Wil16b] Wilson, S. et al. “The Creation and Analysis of a Website Privacy Policy Corpus”. *Proceedings of the Association for Computational Linguistics (ACL)*. 2016.
- [Wu13] Wu, L. et al. “The Impact of Vendor Customizations on Android Security”. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2013.
- [Xia19] Xiao, X. et al. “IconIntent: Automatic Identification of Sensitive UI Widgets based on Icon Classification for Android Apps”. *Proceedings of the International Conference on Software Engineering (ICSE)*. 2019.
- [XW15] Xu, Y. & Witchel, E. “Maxoid: Transparently Confining Mobile Applications with Custom Views of State”. *Proceedings of the European Conference on Computer Systems*. 2015.

- [Yan15] Yang, W. et al. "Appcontext: Differentiating Malicious and Benign Mobile App Behaviors using Context". *Proceedings of the International Conference on Software Engineering (ICSE)*. 2015.
- [Yan13] Yang, Z. et al. "AppIntent: Analyzing Sensitive Data Transmission in Android for Privacy Leakage Detection". *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2013.
- [Yu15] Yu, L. et al. "AutoPPG: Towards Automatic Generation of Privacy Policy for Android Applications". *Proceedings of the ACM Workshop on Security and Privacy in Mobile Devices (SPSM)*. 2015.
- [Yu16] Yu, L. et al. "Can We Trust the Privacy Policies of Android Apps?" *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2016.
- [Zae13] Zaeem, R. N. et al. "PrivacyCheck: Automatic Summarization of Privacy Policies Using Data Mining". *ACM Transactions on Internet Technology (TOIT)* (2013).
- [Zha15] Zhang, M. et al. "Towards Automatic Generation of Security-Centric Descriptions for Android Apps". *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 2015.
- [Zha13] Zhang, Y. et al. "Vetting Undesirable Behaviors in Android Apps with Permission Use Analysis". *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2013.
- [ZJ12] Zhou, Y. & Jiang, X. "Dissecting Android Malware: Characterization and Evolution". *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2012.
- [Zho12] Zhou, Y. et al. "Hey, You, Get off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets". *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*. 2012.
- [ZB14] Zimmeck, S. & Bellovin, S. M. "Privee: An Architecture for Automatically Analyzing Web Privacy Policies". *Proceedings of the USENIX Security Symposium*. 2014.
- [Zim17a] Zimmeck, S. et al. "A Privacy Analysis of Cross-device Tracking". *Proceedings of the USENIX Security Symposium*. 2017.

[Zim17b] Zimmeck, S. et al. "Automated Analysis of Privacy Requirements for Mobile Apps". *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*. 2017.

## APPENDIX

APPENDIX

A

APPENDIX

**Table A.1** Sensitive User Input Request Outliers

Category	k	Cluster	Concepts
Books & Reference	61	Largest (190)*	credit_card (190)
		Outliers (723)	person_name (561), passwd (552), username_or_email (520), location (416), phone_num (396), DOB (293), credit_card (282), gender (111), device_id (73), company_name (18), salary (14), loan_info (10), insurance_policy_num (8), vehicle_vin (7), vehicle_info (7), lic_plate (7), driver_lic_num (6), twitter_passwd (3), medication_name (3), school_name (2), person_wt (2), job_title (2), social_media_url (1), political_affil (1), person_ht (1), person_age (1), marital_status (1), family_member_name (1), adobe_passwd (1)
Business	221	Largest (523)*	username_or_email (523), phone_num (523), passwd (523), location (523), loan_info (523), person_name (523), credit_card (523)
		Outliers (1408)	username_or_email (1176), passwd (1029), person_name (866), phone_num (717), location (668), credit_card (158), company_name (154), loan_info (148), DOB (124), gender (82), vehicle_info (72), job_title (58), house_fin_info (49), vehicle_vin (36), lic_plate (35), bank_acct (29), salary (26), insurance_policy_num (26), driver_lic_num (16), person_wt (11), bank_info (11), twitter_passwd (9), person_age (9), vehicle_reg (8), school_name (8), education_info (8), id_num (7), social_media_url (6), marital_status (6), mac_addr (6), device_id (6), SSN (5), serial_num (5), person_ht (5), marriage_date (5), demographic_info (5), lic_expiry_date (3), sim_card (2), service_provider (2), native_language (2), medication_name (2), drug_dosage (2), device_manufac_info (2), wifi_ssid (1), wifi_passwd (1), tax_id (1), medicaid_num (1), family_member_name (1), exchange_passwd (1), driver_id (1), citizenship (1), body_mass_index (1), blood_type (1), blood_pressure (1), birthplace (1)
Comics	13	Largest (5)*	username_or_email (5), passwd (5), person_name (5)
		Outliers(18)	username_or_email (13), passwd (13), person_name (8), phone_num (7), location (6), credit_card (3), loan_info (2), house_fin_info (1), gender (1), DOB (1)
Communication	92	Largest (83)*	username_or_email (83), passwd (83)
		Outliers (438)	phone_num (289), username_or_email (275), passwd (252), person_name (195), location (142), credit_card (59), DOB (43), gender (16), loan_info (12), job_title (9), serial_num (6), company_name (5), school_name (4), wifi_passwd (2), twitter_passwd (2), service_provider (2), person_age (2), marital_status (2), mac_addr (2), lic_plate (2), id_num (2), gmail_passwd (2), facebook_passwd (2), education_info (2), device_id (2), bank_acct (2), vehicle_vin (1), vehicle_info (1), smtp_passwd (1), sim_card (1), religion (1), person_wt (1), person_ht (1), family_contact_phone (1), device_manufac_info (1), demographic_info (1)
Education	98	Largest (255)*	username_or_email (255), passwd (255)
		Outliers (778)	username_or_email (621), person_name (567), passwd (565), location (394), phone_num (354), credit_card (300), loan_info (233), DOB (90), gender (62), school_name (28), family_member_name (11), company_name (9), job_title (6), education_info (6), twitter_passwd (5), person_wt (4), person_age (4), medication_name (3), native_language (2), guardian_email (2), vehicle_info (1), tax_id (1), SSN (1), serial_num (1), salary (1), religion (1), drug_dosage (1), driver_lic_num (1), device_id (1), demographic_info (1), blood_pressure (1), bank_acct (1)

Category	k	Cluster	Concepts
Entertainment	81	Largest (128)*	username_or_email (128), passwd (128)
		Outliers (625)	person_name (448), username_or_email (444), passwd (423), location (349), phone_num (332), credit_card (289), DOB (167), loan_info (126), gender (97), twitter_passwd (6), person_age (6), company_name (5), service_provider (3), school_name (2), person_wt (2), id_num (2), demographic_info (2), vehicle_info (1), smtp_passwd (1), sim_card (1), salary (1), person_ht (1), native_language (1), medication_name (1), lic_plate (1), job_title (1), ftp_passwd (1), education_info (1), device_manufac_info (1)
Finance	125	Largest (60)*	username_or_email (60), passwd (60)
		Outliers (389)	username_or_email (235), passwd (234), person_name (222), location (215), phone_num (164), credit_card (86), DOB (66), bank_acct (54), salary (46), loan_info (43), bank_info (40), SSN (25), gender (25), company_name (19), vehicle_info (10), house_fin_info (5), marital_status (4), job_title (4), driver_lic_num (4), vehicle_vin (3), person_age (3), lic_plate (3), family_member_name (3), school_name (2), insurance_policy_num (2), birthplace (2), vehicle_reg (1), tax_id (1), person_wt (1), mint_passwd (1), id_num (1)
Game	56	Largest (425)*	username_or_email (425), passwd (425)
		Outliers (604)	username_or_email (348), person_name (313), passwd (200), location (142), DOB (114), gender (87), credit_card (86), phone_num (83), person_age (52), demographic_info (10), salary (9), marital_status (9), education_info (9), SSN (3), school_name (3), passport_num (3), twitter_passwd (2), person_wt (2), device_id (2), person_ht (1), loan_info (1), job_title (1), gmail_passwd (1), bank_info (1), bank_acct (1)
Health & Fitness	142	Largest (105)*	username_or_email (105), phone_num (105), passwd (105), location (105), loan_info (105), person_name (105), credit_card (105)
		Outliers (554)	username_or_email (367), passwd (342), person_name (298), location (244), phone_num (194), DOB (193), credit_card (176), gender (105), person_wt (75), person_ht (60), medication_name (18), person_age (17), blood_pressure (10), drug_dosage (9), heart_rate (8), blood_type (8), company_name (7), body_mass_index (7), loan_info (6), family_member_name (5), job_title (4), blood_glucose (4), prescription_num (3), education_info (3), demographic_info (3), birthplace (2), bank_acct (2), SSN (1), service_provider (1), serial_num (1), school_name (1), marital_status (1), insurance_policy_num (1), gmail_passwd (1), doctor_email_id (1)
Libraries & Demo	13	Largest(5)*	passwd (5), username_or_email (4), person_name (3)
		Outliers(18)	person_name (12), username_or_email (10), passwd (10), location (10), phone_num (8), credit_card (6), device_id (3), DOB (3), person_age (2), SSN (1), loan_info (1), gender (1), education_info (1)
Lifestyle	89	Largest (336)*	username_or_email_address (336), phone_number (336), password (336), loan_info (336), full_name (336), credit_card_info (336), location_info (331)
		Outliers (1147)	username_or_email_address (943), password (840), full_name (734), location_info (538), phone_number (523), credit_card_info (358), date_of_birth (316), gender (224), company_name (35), loan_info (20), prescription_number (18), medication_name (13), person_weight (8), person_age (7), twitter_password (5), vehicle_info (3), school_name (3), person_height (3), job_title (3), salary (2), marital_status (2), education_info (2), device_id (2), wifi_password (1), vehicle_vin (1), tax_id (1), serial_number (1), religion (1), native_language (1), mac_address (1), insurance_policy_number (1), house_financial_info (1), gmail_password (1), family_member_name (1), family_contact_phone (1), drug_dosage (1), device_manufac_info (1), demographic_info (1), citizenship (1), blood_type (1), birth_place (1), bank_info (1), bank_account_info (1)

Category	k	Cluster	Concepts
Media & Video	48	Largest (49)*	username_or_email (49), passwd (49)
		Outliers (179)	passwd (146), person_name (82), username_or_email (80), location (48), phone_num (45), credit_card (35), DOB (21), gender (9), loan_info (8), wifi_passwd (6), mac_addr (4), ftp_passwd (3), wifi_ssid (2), twitter_passwd (2), smtp_passwd (2), device_id (2), company_name (2), social_media_url (1), medication_name (1)
Medical	88	Largest (87)*	username_or_email (87), phone_num (87), passwd (87), location (87), person_name (87), DOB (87), credit_card (87)
		Outliers (258)	username_or_email (196), person_name (160), passwd (155), phone_num (133), location (109), credit_card (68), loan_info (49), gender (32), medication_name (27), DOB (22), person_age (21), person_wt (15), person_ht (12), blood_pressure (10), drug_dosage (9), prescription_num (6), heart_rate (6), family_member_name (5), family_contact_phone (4), body_mass_index (3), blood_type (3), blood_glucose (2), twitter_passwd (1), SSN (1), serial_num (1), marital_status (1), insurance_policy_num (1), doctor_email_id (1), device_id (1), company_name (1)
Music & Audio	45	Largest (138)*	username_or_email (138), passwd (138)
		Outliers (246)	person_name (156), username_or_email (152), passwd (134), location (97), phone_num (84), credit_card (59), DOB (43), loan_info (24), gender (21), company_name (12), vehicle_info (1), twitter_passwd (1), school_name (1), person_age (1), mac_addr (1), id_num (1), device_manufac_info (1)
News & Magazines	60	Largest (163)*	username_or_email (163), passwd (163)
		Outliers (429)	username_or_email (341), passwd (309), person_name (281), location (219), phone_num (161), credit_card (145), company_name (113), DOB (38), gender (30), loan_info (14), bank_acct (5), service_provider (4), person_age (4), lic_plate (4), twitter_passwd (2), school_name (2), political_affil (2), job_title (2), id_num (2), demographic_info (2)
Personalization	27	Largest (134)*	username_or_email (134), phone_num (134)
		Outliers (93)	username_or_email (59), passwd (58), person_name (52), phone_num (41), location (39), DOB (27), credit_card (27), gender (7), loan_info (2), vehicle_vin (1), vehicle_info (1), serial_num (1), salary (1), lic_plate (1), device_id (1), birthplace (1), bank_info (1)
Photography	30	Largest (34)*	username_or_email (34), phone_num (34), passwd (34), location (34), person_name (34), DOB (34), credit_card (34)
		Outliers (90)	username_or_email (73), passwd (64), person_name (43), location (37), phone_num (24), credit_card (22), gender (16), DOB (13), loan_info (6), twitter_passwd (2), person_age (2), person_ht (1), ftp_passwd (1), driver_lic_num (1), company_name (1)
Productivity	108	Largest (95)*	username_or_email (95), passwd (95)
		Outliers (417)	username_or_email (276), passwd (262), person_name (254), location (199), phone_num (178), credit_card (74), DOB (48), loan_info (46), vehicle_info (43), company_name (43), gender (19), job_title (12), salary (11), insurance_policy_num (7), driver_lic_num (7), lic_plate (6), vehicle_vin (5), SSN (4), person_age (4), marital_status (4), bank_info (4), bank_acct (4), serial_num (3), person_ht (3), device_id (3), twitter_passwd (2), sim_card (2), school_name (2), person_wt (2), gmail_passwd (2), wifi_passwd (1), vehicle_reg (1), student_id (1), religion (1), nq_account_passwd (1), mac_addr (1), gritsafe_passwd (1), family_contact_phone (1), blood_type (1)

Category	k	Cluster	Concepts
Shopping	81	Largest (108)*	password (108), username_or_email_address (72), full_name (72), phone_number (36), location_info (36), loan_info (36), credit_card_info (36)
		Outliers (214)	username_or_email_address (148), location_info (132), password (115), full_name (110), phone_number (98), credit_card_info (73), date_of_birth (26), gender (21), bank_account_info (20), bank_info (17), company_name (15), person_age (5), loan_info (3), vehicle_info (2), twitter_password (2), person_weight (2), person_height (2), vehicle_vin (1), tax_id (1), service_provider (1), school_name (1), native_language (1), marital_status (1), license_plate (1), issue_date (1), flight_number (1), family_member_name (1)
Social	108	Largest (84)*	username_or_email (84), passwd (84)
		Outliers (420)	username_or_email (312), passwd (290), person_name (260), location (208), phone_num (188), DOB (110), credit_card (85), gender (68), person_age (23), loan_info (22), job_title (12), school_name (7), company_name (6), marital_status (4), education_info (4), twitter_passwd (3), blood_type (2), vehicle_info (1), serial_num (1), person_wt (1), flight_num (1), family_member_name (1), family_contact_phone (1), citizenship (1)
Sports	68	Largest (75)*	username_or_email (75), phone_num (75), passwd (75), location (75), loan_info (75), person_name (75), credit_card (75)
		Outliers (353)	username_or_email (269), passwd (251), person_name (186), location (126), phone_num (93), DOB (86), credit_card (76), gender (39), person_wt (9), company_name (9), person_ht (7), loan_info (5), twitter_passwd (4), person_age (4), wifi_passwd (1), SSN (1), serial_num (1), school_name (1), salary (1), medication_name (1), marital_status (1), education_info (1), driver_lic_num (1), demographic_info (1), citizenship (1)
Tools	109	Largest (157)*	username_or_email_address (157), password (157), location_info (157), full_name (157), credit_card_info (157), phone_number (156), date_of_birth (155)
		Outliers (634)	password (317), username_or_email_address (301), location_info (219), phone_number (189), full_name (168), credit_card_info (51), date_of_birth (35), gender (28), loan_info (13), company_name (13), person_weight (11), mac_address (10), bank_account_info (10), person_height (9), person_age (6), wifi_password (5), serial_number (5), school_name (5), device_id (5), wifi_ssid (4), salary (4), vehicle_info (3), job_title (3), id_number (3), gmail_password (3), driver_license_number (3), service_provider (2), mother_birth_place (2), license_plate (2), twitter_password (1), social_security_number (1), smtp_password (1), medication_name (1), marital_status (1), insurance_policy_number (1), house_financial_info (1), ftp_password (1), education_info (1), drug_dosage (1), driver_id (1), class_name (1), citizenship (1)
Transportation	89	Largest (35)*	location (35)
		Outliers (244)	username_or_email (185), person_name (152), passwd (147), location (141), phone_num (140), credit_card (71), bank_acct (23), vehicle_info (20), loan_info (13), flight_num (13), DOB (13), company_name (11), gender (7), vehicle_vin (6), insurance_policy_num (5), driver_lic_num (5), device_id (5), lic_plate (3), job_title (3), school_name (2), person_wt (2), person_ht (2), bank_info (2), vehicle_reg (1), twitter_passwd (1), service_provider (1), guardian_email (1), family_member_name (1), family_contact_phone (1), driver_id (1)
Travel & Local	108	Largest (111)*	location (111)
		Outliers (806)	username_or_email (653), passwd (598), person_name (515), location (495), phone_num (371), credit_card (299), gender (129), DOB (115), loan_info (92), company_name (20), flight_num (13), SSN (9), vehicle_info (7), person_age (5), bank_acct (4), service_provider (3), twitter_passwd (2), id_num (2), driver_lic_num (2), driver_id (2), citizenship (2), school_name (1), person_wt (1), person_ht (1), native_language (1), lic_plate (1), job_title (1), demographic_info (1), bank_info (1)

Category	k	Cluster	Concepts
Weather	20	Largest (40)	location (40)
		Outliers (51)	username_or_email (38), location (34), passwd (30), person_name (17), phone_num (9), mac_addr (2), DOB (2), credit_card (2), company_name (1)