Doctoral theses at NTNU, 2024:82

Feiyang Tang

Analyzing Privacy in Software

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of Science and Technology

Feiyang Tang

Analyzing Privacy in Software

Thesis for the Degree of Philosophiae Doctor

Gjøvik, April 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

ISBN 978-82-326-7758-0 (printed ver.)
ISBN 978-82-326-7757-3 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2024:82

Printed by NTNU Grafisk senter

To my mother,
for her endless love.

ABSTRACT

In our increasingly digital world, a pressing concern emerges: *How do we secure our privacy as we increasingly depend on software?* As we navigate through apps and platforms, the complexities of data privacy become evident. Understanding the intricate flow of personal data, ensuring compliance with evolving global regulations, and developing adaptable tools for diverse software environments are paramount. This Ph.D. thesis delves deep into these challenges, offering insights and solutions that span from the granular details of code to the broader validation of privacy policies.

The first challenge is the subtlety of personal data. Legal definitions are often abstract and translating them into technical requirements is no easy task. Identifying what constitutes personal data in a sea of code is a daunting challenge. Secondly, understanding how personal data flows within systems is crucial. With regulations like the General Data Protection Regulation (GDPR) in place, it is crucial to know what kind of processing personal data undergoes for compliance checks. Lastly, different projects have different needs. For developers doing self-analysis, a detailed examination of compiled code can reveal intricate data flows. However, for large industry projects, high-level source code analysis may be more practical for third parties to quickly gauge privacy compliance situations across millions of lines.

Investigations into these aspects resulted in the eight papers that are presented in this dissertation. They also led to the following additional contributions: (1) A privacy flow-graph tailored for Java and Android applications; this approach aids in the Data Protection Impact Assessment (DPIA) process. (2) A biometric data identification approach developed to pinpoint biometric API usage within Java and Android applications; this method ensures alignment with the GDPR. (3) An automatic comparison approach that addresses the collection of user interaction data in mobile apps by comparing an app's privacy policy claims with its actual code implementation. (4) An automated code review assistant that offers a method to identify and categorize relevant code segments in source code, thus reducing the manual review effort.

The contributions offer guidance for developers and legal experts, connecting the detailed aspects of software development with the clear rules of privacy regulations. These contributions can pave the way for a clearer, more streamlined, and compliant online environment, ensuring that as we use digital platforms, our privacy is always protected.

CONTENTS

LIST OF PAPERS

Paper 1

Tang, F. and Østvold, B. (2022). Assessing software privacy using the privacy flow-graph. In Proceedings of *the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security (MSR4P&S 2022).* Association for Computing Machinery, New York, NY, USA, 7–15.

Paper 2

Tang, F. (2022)., PABAU: Privacy Analysis of Biometric API Usage, In Proceedings of *the 2022 IEEE Conference on Privacy Computing (PriComp 2022)*, Haikou, China, 2022, pp. 2301-2308.

Paper 3

Tang, F.; Østvold, B. and Bruntink, M. (2023). Identifying Personal Data Processing for Code Review. In Proceedings of *the 9th International Conference on Information Systems Security and Privacy - ICISSP*; ISBN 978-989-758-624-8; ISSN 2184-4356, SciTePress, pages 568-575.

Paper 4

Tang, F.; Østvold, B. and Bruntink, M. (2023). Helping Code Reviewer Prioritize: Pinpointing Personal Data and its Processing. In Proceedings of *the 22nd International Conference on Intelligent Software Methodologies, Tools and Techniques (SOMET 2023). DOI: 10.3233/FAIA230228.*

Paper 5

Tang, F. and Østvold, B. (2023). Transparency in App Analytics: Analyzing the Collection of User Interaction Data. In Proceedings of *the 20th Annual International Conference on Privacy, Security & Trust (PST 2023) DOI: 10.1109/PST58708.2023.10320181.*

Paper 6

Tang, F. and Østvold, B. (2023). User Interaction Data in Apps: Comparing Policy Claims to Implementations. Published at *the 18th IFIP Summer School on Privacy and Identity Management 2023 (IFIPSC 2023).*

Paper 7

Tang, F. and Østvold, B. (2024). Finding Privacy-relevant Source Code. Accepted by *the 2nd International Workshop on Mining Software Repositories Applications for Privacy and Security (MSR4P&S 2024).*

Paper 8

Tang, F. and Østvold, B. (2024). Software Privacy and Program Analysis: Insights, Methods, and Opportunities. A book chapter submitted to the Springer Handbook on *Privacy and Security Matters in Biometric Technologies.*

LIST OF FIGURES

LIST OF TABLES

PREFACE

Two critical moments stand out in my journey, shaping the direction of this Ph.D. research.

After completing my Honours degree at the University of Auckland in 2019, I returned to China and was immediately struck by the rapid digital transformation that had taken place during my absence. Services that were once straightforward now required a digital interface, and to access them, I had to share an extensive array of personal details, from my real-time location to biometric data. This immersion into a digital-first environment sparked my curiosity about the balance between convenience and privacy in software applications.

My interest deepened when I stumbled upon the recruitment post for the PriMa project. The project's focus resonated with my personal experiences, and I immediately shared this with my mother. Her casual acceptance of the extensive data collection by everyday digital tools highlighted a broader societal unawareness. It became clear to me that there was a pressing need to address this gap, to ensure that as we embrace digital tools, we remain informed and protected.

These two moments, one personal and the other academic, have been the guiding lights of my Ph.D. journey. They emphasized the importance of understanding and addressing software privacy, making it accessible and comprehensible to everyone in our increasingly digital world.

ACKNOWLEDGMENTS

Embarking on a Ph.D. journey is never a solitary endeavor, especially when faced with the unique challenges of a global pandemic. I am deeply grateful for the community of scholars, mentors, and friends who have enriched this experience in so many ways. As I reflect on this journey, I find it fitting to acknowledge the invaluable contributions of those who have been crucial in shaping both my research and personal growth.

First and foremost, my advisors deserve my heartfelt gratitude. Dr Bjarte M. Østvold, my primary supervisor, has been a guiding light throughout this process. His openness to meaningful academic dialogue and his meticulous attention to detail have been cornerstones in crafting my research. He has generously devoted his time, even beyond the confines of the office, to elevate the quality of my work. Beside him, Prof Staal Vinterbo, my co-supervisor, has provided timely and insightful advice that has effectively honed my research.

My thesis committee members also deserve special mention and sincere thanks: Dr Nataliia Bielova from Inria, Prof Lothar Fritsch from OsloMet, and Prof Basel Katt from NTNU. Their thorough evaluation and insightful feedback have been critical in refining my dissertation. The time and effort they invested have not just improved this thesis but also broadened my perspective in my field of study.

Moving outward, I acknowledge my NR colleagues Ahmed Fraz Baig, Nicholas Walker, and Dr Izzie Yi Liu, as well as Dr Wolfgang Leister. Our lunchtime discussions have been sources of inspiration and intellectual challenge. Further, my fellow doctoral students from the PriMa project have added depth to my academic experience. Our collective brainstorming and debates have been fertile ground for scholarly development.

I extend my thanks to Dr Magiel Bruntink and Prof Raymond Veldhuis, who guided my secondments at the University of Twente and the Software Improvement Group. Their wisdom has been instrumental in directing my research.

Beyond the academic sphere, my friends have been a sustaining force in my life. Huan Zhang, you are more than just a friend; you have been my anchor in challenging times. To my Leuven mates: Jinjiu, Nanxu, Xinyi, Shuxian and Lieven, you have kept me grounded when the world seemed to spin out of control. And Henrik Hoggen, your ability to make even the darkest Norwegian nights feel *koselig* has been a comfort. To all my friends, thank you genuinely for being there.

Last but certainly not least, this work is dedicated to my mother, Ning Yang. Her sacrifices, boundless love, and unwavering faith have been my guiding light. Her resilience is the strong foundation upon which I have built my achievements. Her enduring love and support have been my pillars, and I can only aspire that my accomplishments make her proud.

As I pen these final words, I am drawn to the wisdom of Norwegian ethnographer Thor Heyerdahl: *“Grenser? Jeg har aldri sett noen. Men jeg har hørt at de eksisterer i tankene til enkelte mennesker.”* In English, this translates to, *“Borders I have never seen one. But I have heard they exist in the minds of some people.”* This journey has indeed been about pushing boundaries — both personal and academic — and each one of you has played a role in that. Thank you for being an integral part of this transformative experience.

「吾生也有涯，而知也無涯。」—《莊子》

Feiyang Tang,
Oslo, February 2024

Part A

Thesis Summary

CHAPTER

ONE

INTRODUCTION

Arguing that you don't care about the right to privacy because you have nothing to hide is no different than saying you don't care about free speech because you have nothing to say.

Edward Snowden

In the realm of digital technology, software plays a key role in our daily interactions. Defined as a set of instructions or programs directing a computer, software has become an integral part of modern life. Alongside this rise of software applications, the concept of privacy has evolved. Privacy, traditionally understood as the right to keep personal matters and information undisclosed, has taken on new dimensions in the digital age, especially concerning how software handles and protects user data.

Privacy has long been valued as an important principle in human societies across history. Westin [1] discusses the history of privacy, emphasizing its relevance from ancient times to the present day. The swift growth of the digital era, marked by the rise of big data and the internet, has further highlighted the significance of privacy. The introduction of the General Data Protection Regulation (GDPR) in 2018 brought a renewed focus on the need to protect personal data online.

As we integrate digital applications into our daily routines, from online banking to social media, these applications often access a large amount of our personal information. The challenge lies in ensuring that this data remains protected and is not used inappropriately.

The primary focus of this dissertation is to examine software development and its implications for privacy. Software systems pervade modern life, from mobile apps to enterprise platforms, handling large amounts of potentially sensitive user data.

How software is designed and built directly impacts what data is collected and how it is processed, stored, and shared. Therefore, understanding privacy issues in the software development process is crucial. The first area of exploration centers on how legal definitions of personal data can be translated into technical requirements. With a diverse range of data types and their associated nuances, it becomes crucial to understand and identify what constitutes personal data within software. The second area delves into the movement and processing of personal data within software systems. With regulations such as GDPR setting clear guidelines, understanding how data is used and processed becomes essential for compliance.

However, the challenges extend beyond these areas. The realities of software development create a need for flexible analysis approaches. For example, developers working on smaller compiled projects may want to do in-depth inspection of bytecode to closely track data flows during coding. This low-level view can uncover subtle leaks missed in source code. In contrast, third-party reviewers assessing privacy compliance in large uncompiled codebases cannot manually comb through millions of lines. They need higher-level static analysis to quickly flag areas of concern for more targeted review. The diversity of software projects calls for privacy tools tailored to different needs, from fine-grained bytecode tracing to fast overviews of massive codebases.

This dissertation pursues advances in two areas to expand privacy protection in software development. One focus is refining analysis approaches themselves to provide impactful insights across diverse codebases and workflows. The second is bridging the gap between privacy regulations and development processes by translating legal principles into technical requirements and integrating privacy analyses into developer workflows.

Before delving deeper into these topics, it is essential to understand the backdrop against which this research is set. Understanding this context will help clarify the motivations and objectives driving this work.

1.1 Motivation

The concept of privacy, traditionally understood as the right to keep personal matters and information undisclosed, has taken on new dimensions in the digital age. The subtlety of privacy, especially in the context of software, poses significant challenges. While legal and psychological definitions of privacy exist, translating them into technical solutions remains a complex endeavor.

The advent of the big data era has led to an exponential increase in the amount of data individuals provide to software applications. From online shopping preferences to health metrics, the range of data we willingly or unknowingly share is broad.

However, the specifics of what software applications collect, how they use this data, and the purposes for which they use it often remain obscured from users. The primary sources of information on these practices, privacy policies, and consent forms, are frequently vague and difficult for the average user to interpret [2].

Moreover, these documents are challenging to validate. The gap between the legal and psychological definitions of privacy and the technical solutions implemented by software developers is widening. This discrepancy raises concerns about the efficacy of current privacy protection measures in software applications. For instance, both studies from Reidenberg et al. [2] and Solove [3] highlighted that while users are concerned about their online privacy, they often struggle to make informed decisions due to the complexity and ambiguity of privacy policies.

Furthermore, the expansive scope of the digital environment entails users interacting with many different software applications every day. Each of these applications has its own set of privacy policies and data collection practices. Keeping track of and understanding the implications of these myriad policies is a daunting task for users [4].

The challenge, therefore, lies in developing technical solutions that align with the nuanced definitions of privacy while ensuring transparency and user trust. As software becomes a more global part of our daily lives, addressing this research problem is of paramount importance. The goal is to ensure that as we navigate the digital realm, our right to privacy remains protected and respected.

1.2 Research questions

The research questions for this Ph.D. thesis were shaped by the motivation. These questions, along with their related sub-questions, are outlined below.

RQ1 What constitutes personal data within the context of software?

- RQ1.1 What is the relationship between the concepts of personal data and personally identifiable information (PII) in software?

RQ2 How to identify personal data in software?

RQ3 How to trace the flow of personal data and categorize these flows in software?

RQ4 How to leverage the identified flows of personal data to support various privacy-related tasks?

- RQ4.1 How to assist legal experts in conducting privacy tasks such as data protection impact assessment (DPIA) or records of processing activities (ROPA)?

RQ4.2 How to make privacy code reviews more effective and efficient?

RQ4.3 How to verify the consistency between privacy policies and actual implementation?

1.3 Contributions

This dissertation follows a collection of articles format. The primary contributions are the papers included within the body of the work. Detailed introductions to each paper can be found in Chapter 5, but we present the titles and their respective venues here:

Paper 1 "Assessing Software Privacy using the Privacy Flow-Graph". In Proceedings of *the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security (MSR4P&S 2022).* Association for Computing Machinery, New York, NY, USA, 7–15.

Paper 2 "PABAU: Privacy Analysis of Biometric API Usage". In Proceeding of *the 2022 IEEE Privacy Computing (PriComp 2022)*, Haikou, China, 2022, pp. 2301-2308.

Paper 3 "Identifying Personal Data Processing for Code Review". In Proceedings of *the 9th International Conference on Information Systems Security and Privacy - ICISSP*; ISBN 978-989-758-624-8; ISSN 2184-4356, SciTePress, pages 568-575.

Paper 4 "Helping Code Reviewer Prioritize: Pinpointing Personal Data and its Processing". Published in *the 22nd International Conference on Intelligent Software Methodologies, Tools and Techniques (SOMET 2023).*
DOI: 10.3233/FAIA230228.

Paper 5 "Transparency in App Analytics: Analyzing the Collection of User Interaction Data". Published in *the 20th Annual International Conference on Privacy, Security & Trust (PST 2023).*
DOI: 10.1109/PST58708.2023.10320181

Paper 6 "User Interaction Data in Apps: Comparing Policy Claims to Implementations". Published at *the 18th IFIP Summer School on Privacy and Identity Management 2023 (IFIPSC 2023).*

Paper 7 "Finding Privacy-relevant Source Code". Accepted by *the 2nd International Workshop on Mining Software Repositories Applications for Privacy and Security (MSR4P&S 2024).*

Paper 8 "Software Privacy and Program Analysis: Insights, Methods, and Opportunities". A book chapter submitted to the Springer Handbook on *Privacy and Security Matters in Biometric Technologies.*

Each included paper is designed to address one or more of the research questions outlined in Section 1.2. A detailed breakdown connecting the papers to the research questions is provided in Table 1.3.1.

Paper	**Knowledge for Research Questions**
Paper 1 [5]	RQ1, RQ2, RQ3, RQ4.1
Paper 2 [6]	RQ2, RQ3
Paper 3 [7]	RQ1, RQ2, RQ4.2
Paper 4 [8]	RQ1, RQ2, RQ3, RQ4.2
Paper 5 [9]	RQ1.1, RQ3, RQ4.3
Paper 6 [10]	RQ1.1, RQ3, RQ4.3
Paper 7 [11]	RQ1, RQ2, RQ3, RQ4.1, RQ4.2
Paper 8 [12]	RQ1, RQ2, RQ3, RQ4

Table 1.3.1: Breakdown of research questions addressed by paper.

1.4 Structure of the dissertation

The dissertation is organized into two main parts. Part A offers an overview and a broader context for the papers that are included in Part B. It is recommended to read the chapters in Part A sequentially, while the chapters in Part B can be read in any preferred order.

Chapter 1 sets the stage by outlining the primary aim, motivation, and a summary of the contributions of the research.

Chapter 2 situates the research within the broader fields of study, providing essential background information.

Chapter 3 discusses current state-of-the-art works on the topic of privacy protection in software. It covers literature from taint analysis and adaptive program analysis to differential privacy. The chapter addresses research gaps identified from studying the state of the art.

Chapter 4 delves into the specific methods and methodologies employed in the dissertation, along with a documentation of selected project activities. While only some activities are included in the dissertation papers, all have contributed to the research.

Chapter 5 offers a summary and rationale for each of the eight papers that constitute this dissertation.

Chapter 6 provides a discussion of the papers, highlighting their contributions and addressing the relevance of each paper to the research questions, different research fields, and social topics.

Chapter 7 concludes Part A with a final summary and explores potential avenues for future research.

Part B of the dissertation contains the reprinted versions of the eight papers, with each paper occupying a separate chapter.

CHAPTER

TWO

BACKGROUND

2.1 Privacy

Privacy is a fundamental human right and a core value that shapes our individuality, autonomy, and freedom. It encompasses the right to control personal information, maintain confidentiality, and make independent decisions without undue interference. In the context of software and technology, privacy takes on new dimensions and complexities, reflecting the evolving nature of human interaction with the digital world. This section explores the multifaceted concept of privacy, its historical evolution, and the pressing challenges and issues in the era of big data and digital transformation.

2.1.1 A historical perspective

Privacy is a principle that has evolved significantly throughout human history. In ancient civilizations, such as those of Greece and Rome, privacy was primarily associated with physical seclusion and personal autonomy [13]. The invention of the printing press shifted the concept towards control over personal data, giving rise to the idea of *the right to be left alone* [14].

In contrast, ancient Eastern philosophies like Confucianism and Taoism did not have a concept directly corresponding to the Western notion of privacy. These philosophies emphasized unity with society or nature rather than individual autonomy [15].

The digital age has further expanded the scope of privacy, introducing challenges such as the Privacy Paradox, where people's actions do not align with their stated privacy concerns [13]. Advances in technology have led to significant changes in Asian privacy regulations, including the introduction of laws like China's Personal Information Protection Law (PIPL), Japan's Personal Information Protection Act

(PIPA), and Singapore's Personal Data Protection Act (PDPA) [16].

Despite these regulatory efforts, privacy remains a complex and evolving issue, with ongoing debates about balancing privacy protection with utility and the challenges of creating privacy-compliant applications [17].

2.1.2 Privacy in the big data era

The era of big data has introduced both opportunities and significant privacy challenges. The extensive collection of personal data has heightened awareness of potential risks and abuses, such as targeted advertising, profiling, and decision-making [18]. High-profile incidents like the Cambridge Analytica scandal have further emphasized the risks of data misuse [19].

Regulatory frameworks like the GDPR and the California Consumer Privacy Act (CCPA) aim to protect individual privacy rights and offer control over personal data [16]. However, these efforts face challenges due to the complexity of privacy policies, lack of user understanding, and difficulties in verifying compliance [20]. The trade-offs between privacy and utility, as well as the tension between individual rights and societal benefits, add another layer of complexity to the privacy landscape [21].

In summary, the big data era has fundamentally reshaped our understanding of privacy, introducing new risks, regulations, and debates. These ongoing efforts to balance privacy protection with innovation and societal needs highlight the enduring complexity and importance of privacy in our increasingly digital world.

2.1.3 Definition of personal data

The concept of personal data is central to the discourse on privacy, but its definition is far from straightforward. Traditionally, personal data has been equated with PII, such as names, social security numbers, and addresses [22]. However, this narrow definition is increasingly being challenged.

Beyond PII: a broader scope Recent studies argue that personal data extends beyond PII to include behavioral and interaction data [23]. For instance, our interaction data, such as location changes in a weather app, can reveal patterns about our lifestyle. A frequent change in the location setting could indicate frequent travel, which could then be used for targeted advertising, such as travel insurance offers. This kind of data may not be directly identifiable but can still be highly personal and sensitive [24]. The implication is that privacy protection measures need to consider this broader scope of data that can be personal in nature.

Is privacy just about personal data? While personal data is a significant aspect of privacy, it is not the only concern. Privacy also involves the right to solitude and the freedom from surveillance [25]. The collection of seemingly non-personal data can still have implications for privacy. For example, smart office environments collect data for energy efficiency but can inadvertently reveal information about individual work patterns [26]. Therefore, privacy protection should also encompass these less obvious but equally important dimensions.

The complexity of defining personal data The definition of what constitutes personal data can vary depending on legal, cultural, and individual perspectives [27]. For instance, some privacy laws focus on *personalized privacy loss*, which allows individuals to set their own boundaries on what they consider private [28]. This suggests that a one-size-fits-all approach to defining personal data is insufficient and that context-specific definitions are necessary.

Implications for privacy protection Understanding the nuanced definition of personal data is crucial for effective privacy protection. Privacy-enhancing technologies and regulations need to adapt to these broader definitions to provide comprehensive privacy safeguards [29]. The challenge lies in developing frameworks that are both flexible and robust enough to accommodate the evolving nature of personal data.

Table 2.1.1: Aspects of personal data in privacy

Aspect	**Examples**
Traditional PII	Names, SSN, Addresses, phone numbers
Behavioral data	App interactions, online behavior
Non-PII but sensitive	Work patterns, masked IP addresses
Beyond data	Right to solitude, freedom from surveillance

The table above (Table 2.1.1 [1]) summarizes the different aspects of personal data in the context of privacy. It highlights that personal data is not just about traditional PII but also includes behavioral data and other sensitive information. Moreover, it emphasizes that privacy concerns extend beyond just data to include other dimensions like the right to solitude and freedom from surveillance. This multifaceted view of personal data underscores the complexity of privacy protection and the need for a nuanced approach.

2.2 Program analysis

Program analysis refers to a broad set of techniques for analyzing software systems, with the goal of extracting information that can support various software engineering tasks such as testing, debugging, maintenance, and verification [30]. Program

[1] *Behavioral data* refers to user activities within an application, including app interactions and online behavior. In our papers, *interaction data* is a specific subset of behavioral data, focusing on user-initiated actions.

analysis can be classified into two main categories: static analysis and dynamic analysis.

Static analysis examines a program without executing it, analyzing properties and behaviors based on the program source code, bytecode, or binary code. Static analysis techniques include data flow analysis, control flow analysis, abstract interpretation, type systems, model checking and symbolic execution [31, 30]. These techniques have been commonly applied for program optimization, bug finding, security, and other applications. For example, static analysis can check for undefined values, null pointer exceptions, buffer overflows, concurrency errors, information leaks, etc. While static analysis has the potential for precision and completeness, these are not guaranteed benefits; they are properties that depend on the specific techniques and tools employed. In fact, static analysis can often be imprecise and, due to inherent limitations, may not be fully complete. However, one definitive advantage of static analysis is its capacity for automation, which allows for scalable and systematic code evaluation.

In contrast, dynamic analysis examines properties and behaviors of a running program, often instrumented to collect runtime data on variables, method calls, memory accesses, etc. Dynamic analysis is traditionally used for profiling, testing, debugging, and malware analysis, especially in-depth malware analysis which executes an instrumented version of the malware sample. While dynamic analysis provides insight into a specific program execution, it cannot guarantee full coverage of all possible executions. Hybrid analysis combines static and dynamic techniques to leverage their complementary strengths.

2.2.1 Static analysis for security and privacy

Many static analysis tools and techniques have been developed for analyzing security and privacy properties of software systems, as these are difficult to thoroughly test and often lead to exploitable vulnerabilities if overlooked. For example, static analyzers can detect access control violations, insecure data flows, use of weak cryptography, and violations of information flow policies [31].

Lightweight syntactic pattern matching can find potentially dangerous APIs and language constructs based on rules. Data flow analysis tracks the flow of data through a program to identify leaks of sensitive data. Taint analysis is a common approach using information flow tracking to detect when untrusted data reaches sensitive program point. Type systems can enforce security policies by checking variable and data types. Well-designed type systems prevent bugs and vulnerabilities by guaranteeing program properties. They also allow encoding security and privacy policies into program semantics, enabling enforcement of allowed information flows and operations on sensitive data. Abstract interpretation offers sound over-approximate analysis through approximating program semantics. Meanwhile, symbolic execution symbolically follows execution paths and generates inputs to uncover vulnerabilities. This approach represents values as symbolic variables rather than concrete data. Theorem proving uses logical deduction to verify program correctness relative to a specification.

Widely-used static analysis frameworks such as FindBugs, Fortify, Coverity, and CodeSonar implement many of these techniques to find bugs and security issues in Java, C/C++, and other languages [32]. Specialized tools like IccTA [33], FlowDroid [34], and Amandroid [35] leverage advanced static analysis to detect inter-component vulnerabilities and sensitive data leaks in Android apps. Facebook's Infer, Google's Tricorder, and GitHub's Semmle apply abstract interpretation for scalable bug finding. The Clang analyzer provides source code analysis for C/C++. Theorem provers like Dafny are used to verify correctness of mission-critical software. Thus, continued innovation in static analysis is enabling more automated reasoning about complex software behaviors.

2.2.2 Program analysis for different languages

The type system of a programming language greatly influences how program analysis is performed. Techniques can be broadly categorized based on whether they are applied to statically typed languages, such as Java and C#, or dynamically typed languages, such as JavaScript and Python.

For statically typed languages, program analyses often work on bytecode, which is the result of compiling high-level code into a standardized instruction set architecture [36]. Examples include Java bytecode, .NET's Common Intermediate Language, and LLVM's intermediate language. Analyzing bytecode has the advantage of preserving detailed type information from the high-level code [37]. This enables analyses to make stronger assumptions about data types and call graphs, such as precisely determining object types being referenced [38]. Whole-program analysis is also easier since bytecode files for an entire codebase can be combined. However, challenges exist, such as handling reflection and native calls [37].

For dynamically typed languages like JavaScript and Python, analyzing raw source code is often necessary since compilation to bytecode loses type details [39]. This analysis relies more on data flow analysis to recover type information [40], but the lack of native type declarations limits precision [41]. Challenges also arise from dynamic code loading and eval statements [39]. Tools like JSNice have been developed to improve analysis by predicting missing type annotations [42].

The key differences between program analysis techniques for statically typed and dynamically typed languages are summarized in Table 2.2.1:

Table 2.2.1: Comparison of analysis techniques for statically typed and dynamically typed languages

Statically typed (Java, C#, etc.)	**Dynamically typed (JavaScript, Python, etc.)**
Bytecode analysis	Source code analysis
Uses declared type information	Infers types from usage
Precise points-to and call graph analysis	Approximate type determination
Facilitates whole-program analysis	Challenged by dynamic code loading
Challenges with reflection and native calls	Limited by eval statements and missing annotations

Both statically typed and dynamically typed program analysis follow core principles like data flow analysis, but key differences in handling type information lead to distinct techniques.

2.3 Issues with privacy policies

Privacy policies are vital documents that inform users about the collection, use, and sharing of their personal information by companies. However, numerous studies have identified significant shortcomings in current privacy policies, undermining their effectiveness in informing and protecting users. This section explores the main issues with privacy policies, including their length, complexity, inconsistencies, and lack of transparency.

2.3.1 Length and complexity

A consistent finding across studies is that privacy policies tend to be extremely long and complex, making them difficult for average users to read and comprehend. A 2008 study analyzed privacy policies from 75 popular websites and found the average length was over 2,500 words, with the longest being over 11,000 words [43]. This length far exceeds the reading level of most internet users.

Another study tested the readability of Facebook's privacy policy over time and found it required a university graduate reading level, while getting increasingly longer and more complex [44]. The complexity is further increased by the use of vague, legalistic language, making it difficult for users to fully grasp what is covered under the policy [2].

2.3.2 Inconsistencies and contradictions

In addition to their length and complexity, privacy policies often contain inconsistent or contradictory statements. A 2017 study of mobile app privacy policies found a number of internal inconsistencies, such as apps claiming they did not share personal information while also stating they shared data with third parties [45]. Another study found inconsistencies in how concepts like opt-out choices were described across sections of the same policy [2]. These inconsistencies create confusion over what policies actually mean in practice.

2.3.3 Lack of transparency

Numerous critiques have argued that privacy policies often lack key details and transparency about how user data is really handled. Policies tend to describe data practices in broad, abstract ways that obscure specifics on what is collected and shared [46]. Vague references to *third parties* make it impossible to know exactly who user data is going to [45]. And details on secondary uses of data beyond the immediate service are often missing entirely [47]. This lack of transparency runs counter to the purpose of informing users.

While privacy policies aim to disclose data practices to users, in reality, their length, complexity, inconsistencies, and lack of transparency often achieve the opposite effect. These issues make privacy policies difficult to comprehend and obscure key details on how user data is handled. Significant improvements to privacy policies are needed for them to properly serve user rights and interests.

CHAPTER

THREE

STATE OF THE ART

Whatever worked in the past, build on it; whatever didn't work in the past, break the chain that binds you to it.

Marianne Williamson

This section discusses the challenges of protecting personal data in software, highlighting the essential technique of taint analysis and detailing the various phases of data flow analysis, all while using studies from recent research as references.

3.1 Taint analysis for personal data flows

Understanding the flow of personal data is essential for the secure and private operation of software systems. Taint analysis serves as a key in this context, as it tracks sensitive data as it moves through a program. Slavin et al. [48] have notably contributed to this area by developing a framework that aligns Android application behavior with privacy policies.

Data flow analysis is a structured approach to tracking how personal data move through a software system. This process involves different stages, each with its own set of challenges and methods.

3.1.1 Identifying data flows

Identifying the paths through which personal data traverses within a system is foundational to data flow analysis. Li et al. [33] introduced ICCTA, a tool that employs static taint analysis to detect inter-component privacy leaks in Android apps. This approach is particularly effective for bytecode, where data flows can be traced through specific I/O methods.

Staicu et al. [49] conducted an empirical study on real-world JavaScript applications, revealing the prevalence of different kinds of flows and their significance in ensuring security. Their findings suggest that while explicit flows are common, implicit flows, which arise from not executing certain branches, are more challenging to detect and analyze.

3.1.2 Categorizing data flows

Once data flows are identified, they must be categorized based on their nature and associated risks. This categorization is crucial for understanding the potential implications of each flow and for devising appropriate mitigation strategies. Zimmeck et al. [45] automated the analysis of privacy requirements for mobile apps, streamlining the categorization process. Their approach leverages machine learning to automatically classify data flows based on predefined privacy requirements.

Wilson et al. [45] took a different approach by analyzing a corpus of website privacy policies. Their work underscores the importance of consistency in data flow categorizations, especially when considering the legal and regulatory implications of data processing.

Piskachev et al. [50] introduced fluentTQL, a query language specifically designed for taint-flow. This internal Java DSL allows for the expressive specification of various taint-style vulnerabilities, facilitating the categorization and clustering of data flows in software systems.

3.2 Policy analysis

Numerous studies have focused on analyzing and improving privacy policies in mobile apps. Researchers have explored various NLP approaches to automatically process and understand privacy policy texts, as well as to assist users in comprehending these policies more effectively [51, 52, 53]. Tools like PrivacyFlash Pro [54] and AutoCog [55] have been developed to audit privacy policy compliance by comparing disclosed policies with actual app behavior. A recent study by Bardus et al. [56] systematically mapped existing contact-tracing apps and evaluated the permissions required and their privacy policies.

3.3 Tailoring approaches for front-end and back-end systems

Different systems, such as front-end and back-end, necessitate distinct approaches for data flow analysis. Von Maltitz et al. [57] presented a formalization based on static taint analysis to assess software architectures. Their work highlights the differences in data flow patterns across various system components and emphasizes the need for adaptive solutions tailored to each system's unique characteristics.

Kohli [58] proposed a coarse-grained dynamic taint analysis technique that tracks information flow at the level of application data objects. This approach reduces taint management overhead and can detect a wide range of attacks, including non-control data attacks, without requiring source code access.

3.4 Evaluating transmission and storage mechanisms

The methods through which data is transmitted and stored also warrant meticulous examination. Differential privacy and anonymity are two critical concepts in this domain. Leoni [59] surveyed non-interactive differential privacy applications, emphasizing its applicability on real-life datasets. Li, Qardaji, and Su [60] discussed the relationship between k-anonymity and differential privacy, highlighting the potential of random sampling in enhancing privacy protection.

Domingo-Ferrer and Soria-Comas [61] explored the connection between t-closeness and differential privacy, suggesting that both models can offer robust privacy guarantees when used in tandem. Kroll [62] delved into pointwise adaptive kernel density estimation under local approximate differential privacy, emphasizing the importance of adaptive methods for data analysis. Lastly, Sánchez et al. [63] proposed utility-preserving differentially private data releases using individual ranking microaggregation, focusing on the preservation of data utility while ensuring privacy.

3.5 Research gaps

The existing research offer valuable contributions to the understanding of personal data processing and risk assessment. However, there are limitations in the current state of the art that pose challenges for software privacy. This dissertation attempts to address some of these open issues and advance the field.

Firstly, the issue of scalability in personal data identification methods is a clear gap. While existing research has made strides in identifying personal data within specific programming languages or types of analysis (compiled or uncompiled), there is a lack of methods that are universally scalable. For example, current tools may excel in Java but falter when applied to other languages like Python or JavaScript. Furthermore, the speed of these methods is often not up to the mark, especially in real-time development environments. This gap is concerning given the diverse range of programming languages and software types in use today.

Secondly, the categorization of data flows is another area where existing research falls short. While contributions from Zimmeck et al. [45] and Wilson et al. [64] have aided in the categorization of data flows, their methods are not universally applicable. Moreover, even when data flows are categorized, these categorizations often do not align with relevant legal regulations such as GDPR. This is an important gap, as it hampers the generation of Records of Processing Activities (ROPA) and Data Protection Impact Assessments (DPIA), which are crucial for compliance with privacy laws.

Thirdly, there is a noticeable lack of support for code review tasks, which are an integral part of the software development life cycle. Code review requires methods that are not only fast but also applicable across multiple programming languages [65, 66]. These methods should provide insights that enable reviewers to quickly identify potential privacy hotspots in the code. The absence of such methods creates a bottleneck in the development process, delaying the release of privacy-compliant software.

Fourthly, the debate on what technically constitutes personal data is another area that has been largely overlooked. While there is decent amount of research focusing on the sociological or legal aspects of what constitutes personal data [67], there is a lack of technical evidence and analysis. This gap is particularly problematic because it leaves room for interpretation, which could lead to inconsistencies in how personal data is handled across different software systems.

Lastly, another gap lies in the integration of privacy policy analysis with tangible program analysis. While there are efforts to analyze privacy policies and understand their implications [68, 45, 69], these are often not combined with a detailed analysis of the actual software behavior. This lack of integration results in a disconnect between what the privacy policy promises and what the software actually does, making it difficult to verify alignment between the two. This is a critical gap, as it undermines the trust users place in software systems and poses challenges for ensuring privacy compliance.

By addressing these gaps, this dissertation aims to make significant contributions to the field, particularly in the areas of scalable personal data identification, effective data flow categorization, privacy code review support, and privacy policy consistency analysis, all while ensuring compliance with evolving legal regulations.

CHAPTER

FOUR

METHOD

The proper method for inquiring after the properties of things is to deduce them from experiments.

Isaac Newton

This dissertation employs a combination of methods from computer science, law, and empirical analysis to address the research questions outlined in Chapter 1. The choice of methods is crucial to obtain valid and insightful results. This chapter provides an overview and justification of the key techniques used in the dissertation.

4.1 Literature analysis

A literature review forms the basis of this research. We systematically studied existing works on the technical definitions of privacy, program analysis, policy analysis, and related areas. This provided essential context on the current state of knowledge, open challenges, and promising directions.

We used snowballing to expand the literature search. Starting from seminal papers, we followed citations and references to uncover additional relevant works. We also examined recent conference proceedings and journal publications, particularly the latest preprints on Arxiv, to identify cutting-edge advancements.

In total, over 150 papers were reviewed, analyzed, and synthesized, especially in the first 1.5 years. We extracted key concepts, methods, evaluations, and limitations. Comparing approaches revealed gaps and opportunities for contribution. The literature analysis guided the formulation of the research questions and methods.

4.2 Program analysis

Program analysis, particularly static analysis, is the core of our research method. The choice of static analysis is motivated by its ability to analyze code without execution, making it scalable for large codebases and suitable for privacy assessments.

We employ control flow analysis, a well-established static analysis technique, to trace personal data flows in software. This choice is motivated by the technique's proven effectiveness in identifying potential data leaks and risky data processing activities. Control flow analysis constructs a model that reveals how data flows through a program, from its source to its destination, offering insights for privacy compliance.

Our implementation leverages Soot [70], a Java optimization and analysis framework, along with FlowDroid [34], a precise Android-focused data flow analyzer. We customized Soot and FlowDroid to identify sources of personal data, categorize sinks representing data usage, and extract flows between them. The modifications account for features specific to privacy analysis like sensitive data types and domain knowledge of risky sinks derived from legal principles.

We further applied semantic rules and customized inter-procedural data flow analysis using the source code analyzer Semgrep[1] to rapidly analyze source code written in diverse languages like Java, JavaScript, Python, and PHP (Papers 3, 4, 7). Semgrep's flexibility in handling diverse languages, AST-based analysis and support for custom rules allows building rich static checkers that go beyond simple syntactic pattern matching. Our rules leverage Semgrep's intra- and inter-procedural data flow analysis to identify flows of personal data between sources and sinks across class boundaries. The rules account for the contextual nature of privacy by incorporating restrictions based on data types, packages, and other semantic criteria. This achieves more precise source code analysis while maintaining scalability across large, multi-language codebases. The combination of Semgrep and our custom rules complements the precision of Soot and FlowDroid for Java with expanded language support and practical utility.

In summary, the combined use of Soot, FlowDroid and Semgrep provides both precise data flow tracing for Java and Android, and rapid analysis of extensive multi-language codebases. This balances accuracy with practical utility when analyzing personal data usage in diverse software.

[1]`https://semgrep.dev`

4.3 Machine learning

In Paper 2, we develop a multi-label classifier using machine learning to automatically categorize biometric API usage. The motivation for using machine learning stems from its ability to generalize from training data, thereby automating tasks that would be labor-intensive to perform manually. This classifier uses features derived from static analysis as input and predicts privacy-related behaviors, such as authentication and encryption.

We experiment with different algorithms, including logistic regression, random forests, and neural networks, using the scikit-learn library in Python. Through parameter tuning, we identify the models that perform best, and we use cross-validation to prevent overfitting. This illustrates the potential of machine learning in identifying privacy attributes based on program analysis data.

Additionally, in Papers 5 and 6, we employ natural language processing techniques to extract and categorize claims about user interaction data collection from privacy policies. We fine-tune BERT-based models on a manually annotated subset of the APP-350 mobile app privacy policy corpus [71]. This enables us to automatically identify relevant sentences in privacy policies and classify them into categories such as data types and collection methods. Our models achieve high levels of precision and recall, underscoring the effectiveness of neural language models in structured policy analysis. By extracting these structured claims, we can compare them with evidence from static analysis to evaluate their consistency. Overall, our research highlights the utility of machine learning in enhancing both privacy policy and code analysis.

4.4 Experiments

Experiments are our primary method of validation. This approach is chosen for its capacity to provide quantifiable metrics, such as precision and recall. These metrics are particularly useful for tasks like identifying personal data (as discussed in Papers 3, 4, 7) and categorizing biometric API usage (Paper 2). By comparing these metrics against a ground truth, we can objectively evaluate the accuracy and reliability of our methods.

We supplement these experiments with case studies on widely-used applications, such as Signal Desktop (Paper 7). These case studies offer real-world context and qualitative insights, enhancing the practical relevance of our research. We also address potential threats to validity and outline steps taken to mitigate biases. This balanced approach, combining quantitative metrics and qualitative analysis, ensures a comprehensive evaluation of our techniques.

4.5 Expert validation

In addition to experiments, we utilize expert analysis for further validation. The reason for incorporating expert input is its ability to capture nuances that automated methods may miss. For example, experts can manually cross-validate the claims made in top Android apps' privacy policies against the actual data practices coded into the apps. This is demonstrated in Paper 5, where we manually verified such claims, and in Paper 6, where we annotated policy sentences to train classifiers for identifying user interaction data claims.

This layer of expert validation adds depth to our empirical evaluations, offering an additional layer of scrutiny that complements machine-based assessments. Alongside literature analysis, surveys, and case studies, this multifaceted validation approach contributes to a rigorous and transparent evaluation of our proposed methods.

In summary, our method is designed to address the research questions through a combination of literature analysis, program analysis, machine learning, experiments, and expert validation. This integrated approach allows us to explore and evaluate privacy-related issues in software from both technical and legal perspectives.

CHAPTER FIVE

SUMMARY OF PAPERS

Art is not what you see, but what you make others see.

Edgar Degas

5.1 Paper 1: *Assessing Software Privacy using the Privacy Flow-graph*

Tang, F. and Østvold, B. (2022). Assessing software privacy using the privacy flow-graph. In Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security (MSR4P&S 2022). Association for Computing Machinery, New York, NY, USA, 7–15.

Motivation

With increased digitalization, processing of personal data is common in software services. Regulations like GDPR mandate documenting software's privacy and data protection, requiring collaboration between developers (who understand the code) and legal experts (who understand privacy laws). However, developers may lack legal knowledge and tracking all privacy-related code changes is difficult. This paper proposes using static analysis to build *privacy flow-graphs* showing how personal data flows through the software, providing an abstraction understandable to legal experts. This facilitates mutual understanding and aids legal requirements like Data Protection Impact Assessments (DPIAs).

Summary

This paper proposes privacy flow-graphs to help developers and legal experts document personal data processing in software for legal compliance like GDPR.

The approach transforms bytecode to an intermediate representation using Soot. It finds source (entry) and sink (exit) methods in code using pre-built datasets. A privacy flow-graph is built by tracing data-flow from each source method using control-flow-graphs. Each flow is a series of connected flows between methods of different classes. The graph nodes are all methods in a privacy flow and edges are method invocations. To simplify, an abstraction is created representing key privacy symbols like start source, security process, and sink.

The approach is applied to Signal (messaging) and NextCloud (file hosting) services. For Signal, it identified 11 privacy flows and produced abstractions showing end-to-end encryption. For NextCloud, it revealed personal data upload steps. The privacy flow-graphs help answer key Data Protection Impact Assessment (DPIA) questions by providing flows and abstractions to developers and legal experts.

5.2 Paper 2: *PABAU: Privacy Analysis of Biometric API Usage*

Tang, F. (2022)., PABAU: Privacy Analysis of Biometric API Usage, In Proceeding of the 8th IEEE Conference in Privacy Computing (PriComp 2022), Haikou, China, 2022, pp. 2301-2308.

Motivation

Biometric authentication is becoming common, raising privacy concerns as biometric data is sensitive. Most apps use biometric APIs, so understanding their usage is important. Technical and legal experts face a communication gap - developers provide specifications but legal experts need privacy assessments like DPIAs. Manually reviewing code is difficult. This paper proposes automatically categorizing biometric API usage into privacy behaviors to help both developers and legal experts quickly understand biometric processing and aid privacy assessments.

Summary

This paper proposes PABAU, an approach to analyze privacy behaviors in the usage of biometric APIs in apps. The architecture of our technique is illustrated in Fig. 5.2.1. Biometric authentication is increasingly employed in apps, raising privacy concerns as biometric data is sensitive. Most apps rely on standard biometric APIs,

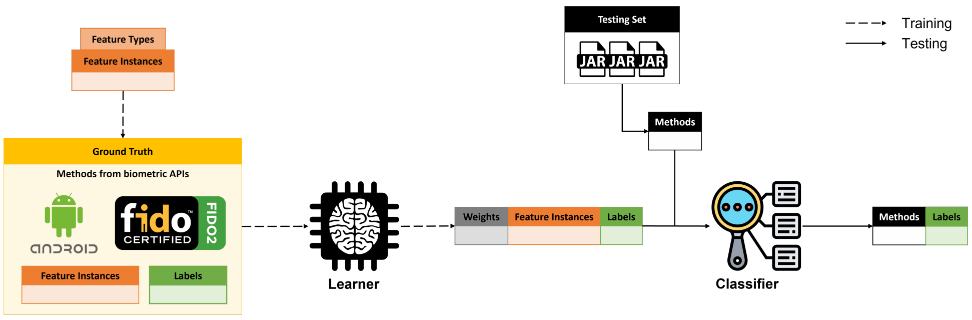


Figure 5.2.1: General architecture of PABAU

necessitating an understanding of their usage for legal compliance requirements like DPIAs. However, reviewing code manually is difficult. PABAU utilizes static analysis and a multi-label classifier to categorize biometric API usage into behaviors like authentication, cryptography, data deletion etc.

It is evaluated on Android biometric APIs and a FIDO2 implementation, achieving high precision. When applied to 8 Android apps, it provided categorizations of their biometric API usage that could help answer DPIA questions.

PABAU enables developers and legal experts to quickly gain a high-level understanding of how biometric APIs are used, aiding privacy assessments. Overall, PABAU shows promise in bridging communication gaps between technical and legal teams regarding privacy behaviors in biometric systems.

5.3 Paper 3: *Identifying Personal Data Processing for Code Review*

Tang, F.; Østvold, B. and Bruntink, M. (2023). Identifying Personal Data Processing for Code Review. In Proceedings of the 9th International Conference on Information Systems Security and Privacy - ICISSP; ISBN 978-989-758-624-8; ISSN 2184-4356, SciTePress, pages 568-575.

*Paper 3 serves as a position paper, motivating the technique proposed in Paper 4, which presents a core technique for pinpointing and grouping personal data processing activities in code.

Motivation

Ensuring privacy compliance like GDPR requires identifying personal data processing in code, often done manually by reviewers. This is time-consuming, requiring specialized knowledge. This paper proposes an approach to automatically identify personal data and processing to assist reviewers in prioritizing their efforts on relevant code. It provides specialized views highlighting data types and flows to focus manual examination, expediting review. This conserves resources, enhancing productivity.

Summary

This paper tackles the challenge of manually identifying personal data processing for GDPR compliance, which is time-consuming and requires specialized knowledge.

It proposes an approach to facilitate rapidly pinpointing relevant code using static analysis. Personal data sources/sinks are identified via Semgrep pattern matching. Code fragments are abstracted into flow patterns capturing processing context. Two specialized views are provided: one showing personal data types, another showing processing flows and details.

The approach was evaluated on 4 open-source GitHub projects, achieving 0.87 precision in identifying personal data flows. It also fact-checked privacy statements of 15 Android apps. The specialized views enable focused manual review on high-priority code areas, expediting analysis.

Overall, by highlighting key code fragments and providing task-specific information, the approach simplifies GDPR compliance tasks like ROPA creation. The multi-faceted views conserve reviewer time/effort, showcasing the potential to streamline privacy analyses.

5.4 Paper 4: *Helping Code Reviewer Prioritize: Pinpointing Personal Data and its Processing*

Tang, F.; Østvold, B. and Bruntink, M. (2023). Helping Code Reviewer Prioritize: Pinpointing Personal Data and its Processing. In Proceedings of the 22nd International Conference on Intelligent Software Methodologies, Tools and Techniques (SOMET 2023). DOI: 10.3233/FAIA230228

Motivation

Manual identification of personal data processing is challenging for code reviewers doing GDPR compliance, requiring specialized knowledge and being time-consuming. Reviewers often resort to techniques like keyword searches which yield overwhelming results. There is a need for a more abstract, categorized view highlighting relevant code fragments. The paper proposes an approach to facilitate rapid pinpointing of personal data processing to assist reviewers. It provides specialized views displaying data types and abstract flows, focusing manual examination on key areas. This saves time and effort, expediting GDPR compliance tasks like ROPA creation.

Summary

This paper extends paper 3, aims to assist code reviewers performing privacy analyses for GDPR compliance by pinpointing personal data processing. Manual identification in extensive codebases is challenging, requiring specialized knowledge and being time-consuming. Reviewers often use inefficient techniques like keyword searches that yield overwhelming results.

The paper proposes an approach to rapidly locate relevant code fragments. Figure 5.4.1 presents an overview of our approach, which consists of three major phases. It employs Semgrep pattern matching to identify personal data sources and sinks. Code fragments are abstracted into flow patterns capturing processing context. Two specialized views are provided: one displaying personal data types, another outlining abstract flows with optional detailed exploration.

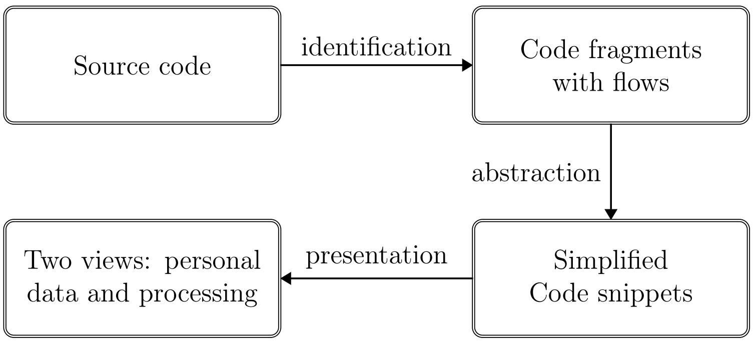


Figure 5.4.1: Overview of our approach in Paper 4

The approach was evaluated on 4 open-source GitHub projects, achieving 0.87 precision in identifying flows. It also fact-checked 15 Android app privacy statements. The views focus manual review on high-priority areas, directing reviewers to key code fragments. This streamlines analysis, saving significant time and effort.

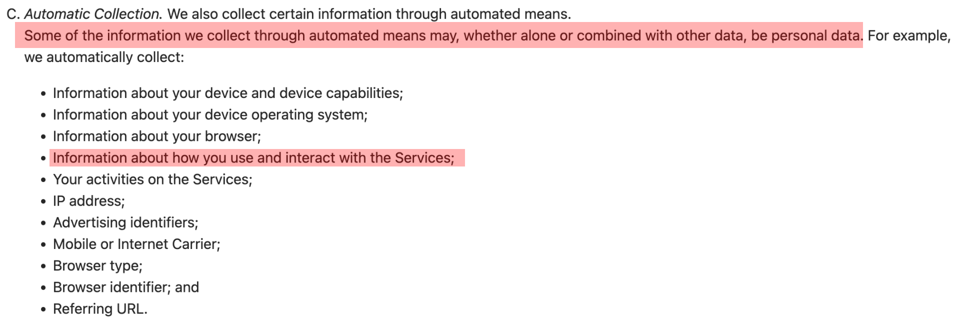
Overall, by simplifying the pinpointing of personal data processing, the approach has potential to expedite compliance tasks like ROPA creation. The multi-faceted assistance demonstrates promise in augmenting the efficiency of privacy analyses.

5.5 Paper 5: *Transparency in App Analytics: Analyzing the Collection of User Interaction Data*

Tang, F. and Østvold, B. (2023). Transparency in App Analytics: Analyzing the Collection of User Interaction Data. In Proceedings of the 20th Annual International Conference on Privacy, Security & Trust (PST2023).

Motivation

Mobile apps extensively collect user interaction data like clicks and scrolls through analytics services. Privacy policies often vaguely describe this collection as *usage data* or *how you interact with the service*, lacking transparency, as exemplified by The Weather Channel app[1] in Figure 5.5.1. This data can reveal sensitive information about users when aggregated, raising ethical concerns. The paper examines common practices of user interaction data collection in apps and proposes a standardized collection claim template for summarizing an app's practices. This is compared to collection evidence from static analysis to fact-check policy claims, addressing the transparency issue.


C. *Automatic Collection.* We also collect certain information through automated means.

Some of the information we collect through automated means may, whether alone or combined with other data, be personal data. For example, we automatically collect:

- Information about your device and device capabilities;
- Information about your device operating system;
- Information about your browser;
- Information about how you use and interact with the Services;
- Your activities on the Services;
- IP address;
- Advertising identifiers;
- Mobile or Internet Carrier;
- Browser type;
- Browser identifier; and
- Referring URL.

Figure 5.5.1: Example of a vague privacy policy fragment describing the collection of interaction data in The Weather Channel app.

Summary

This paper tackles the issue of lack of transparency in mobile apps' collection of user interaction data through analytics services. Privacy policies often vaguely describe such collection as *usage data*, raising concerns. User interaction data can reveal personal information when aggregated, challenging its *non-personal* label. The paper studies common practices by analyzing top analytics libraries.

[1] https://weather.com/en-US/twc/privacy-policy

It proposes a standardized collection claim template summarizing an app's practices. This is compared to collection evidence from static analysis of apps to fact-check claims, addressing the transparency issue. Analysis of the top 100 Android apps revealed widespread collection across UI types like View and Button, showing pervasive data gathering. Case studies manually fact-checking the top 10 apps found all but one failed to declare all data types collected or specify some techniques used. The collection claim template promotes clearer policy disclosures. Comparing claims to evidence assesses alignment and transparency.

Overall, the study enhances understanding of user interaction data collection practices and demonstrates potential for the proposed techniques to improve transparency.

5.6 Paper 6: *User Interaction Data in Apps: Comparing Policy Claims to Implementations*

Tang, F. and Østvold, B. (2023). User Interaction Data in Apps: Comparing Policy Claims to Implementations. Published at the 18th IFIP Summer School on Privacy and Identity Management 2023 (IFIPSC 2023).

*Paper 6 develops the ideas first introduced in Paper 5 for analyzing consistency between app privacy policies and implementation.

Motivation

Mobile apps extensively collect user interaction data like taps and swipes. This data is often labeled *non-personal* in privacy policies, but can reveal personal information when aggregated. There are gaps between policy claims and actual data collection practices, raising transparency concerns. The paper proposes automatically extracting and classifying claims from policies, analyzing apps to categorize collection evidence, and comparing both to identify inconsistencies. This enhances transparency, informing discussions on appropriate data classification.

Summary

This paper extends Paper 5, tackles the lack of transparency in mobile apps' collection of user interaction data, often labeled *non-personal* in privacy policies despite privacy risks when aggregated.

It proposes automatically extracting and classifying claims about such collection from policies using NLP. Apps are statically analyzed to categorize actual collection evidence. Claims are compared to evidence to assess transparency. Our two-fold approach, encompassing privacy policy analysis and application code analysis, is illustrated in Fig. 5.6.1.

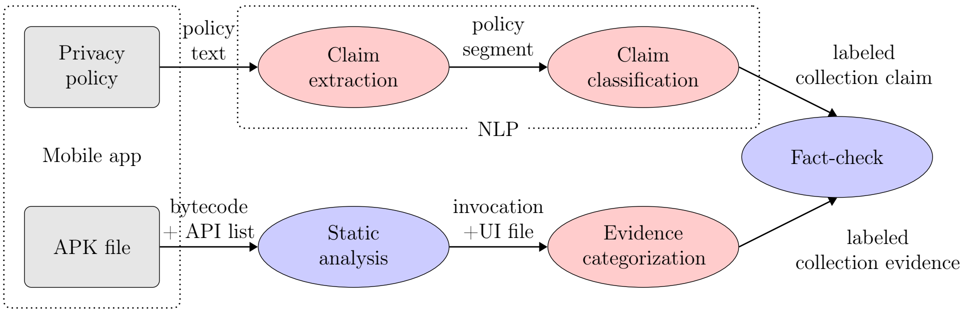


Figure 5.6.1: Overview of our two-fold approach in Paper 6

Analysis of 100 apps revealed widespread interaction data collection, with policy claims often misaligned with practices. Case studies of 4 popular apps showed vague policy contexts and incomplete disclosures. The approach identifies transparency gaps, enabling informed data classification discussions.

Overall, comparing claims to evidence promotes transparency, providing insights into discrepancies between stated practices and actual app behavior. This lays groundwork for improving data collection transparency in mobile apps, underscoring the need for clearer policy communication to foster user trust.

5.7 Paper 7: *Finding Privacy-relevant Source Code*

Tang, F. and Østvold, B. (2024). Finding Privacy-relevant Source Code. Accepted by the 2nd International Workshop on Mining Software Repositories Applications for Privacy and Security (MSR4P&S 2024).

*Paper 7 builds directly on Paper 4, extending the introduced technique.

Motivation

Privacy code reviews are crucial for compliance with data protection regulations like GDPR. However, manually reviewing large codebases to identify methods that process personal data is resource-intensive. Existing tools lack a structured framework for categorizing the diverse ways personal data can be handled in code. This paper proposes an automated approach to assist code reviewers by identifying and categorizing privacy-relevant methods — methods that directly process personal data. By analyzing popular libraries and ranking methods based on their usage frequency, it highlights the methods most relevant for privacy. This provides reviewers with a focused starting point, enabling more efficient reviews.

Summary

This paper builds upon previous work and aims to assist privacy code reviewers by automatically identifying and categorizing privacy-relevant methods. The challenge of reviewing extensive codebases to pinpoint methods involved in personal data processing is both time-intensive and crucial for compliance with regulations like GDPR. The privacy code review process is illustrated in Figure 5.7.1.

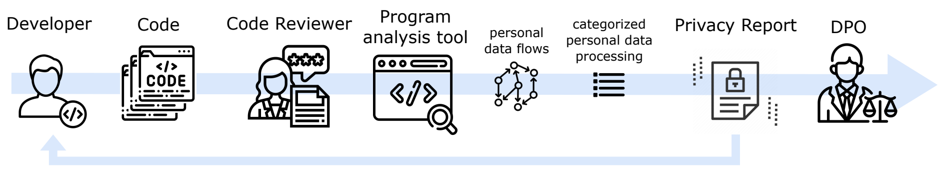


Figure 5.7.1: The process of privacy code review

The paper investigates 50 commonly used libraries to understand their role in personal data processing. It introduces a set of labels of methods, grounded in legal and privacy considerations, and ranks these methods based on their frequency of usage in 30 popular GitHub applications to highlight prevalent practices.

By analyzing 100 open-source applications, the paper demonstrates that its approach narrows the focus to fewer than 5% of privacy-relevant methods, thereby enabling a more targeted review. Case studies on Signal Desktop and Cal.com further illustrate how the approach reduces the scope of manual reviews by focusing on key areas of concern. The labeling serves as a guide for categorizing types of data processing, directing reviewers to high-priority sections of code and saving time.

Overall, the paper shows potential to assist code reviewers in conducting more efficient privacy reviews, thereby aiding in legal compliance. By identifying and categorizing common patterns and practices, the approach streamlines the review process, conserving valuable resources.

5.8 Paper 8: *Software Privacy and Program Analysis: Insights, Methods, and Opportunities*

Tang, F. and Østvold, B. (2024). Software Privacy and Program Analysis: Insights, Methods, and Opportunities. Submitted to the Springer Handbook: Privacy and Security Matters in Biometric Technologies.

Motivation

As software permeates daily life, protecting user privacy is paramount. However, identifying and managing personal data in code is challenging. Privacy policies inadequately disclose data practices, and regulations like GDPR are ambiguous to apply in software. Program analysis can analyze code to detect personal data flows and recommend interventions, crucial for compliance. But limitations exist, including unclear definitions of personal data and complex system architectures. This chapter explores applying program analysis to enhance software privacy, examining techniques and discussing challenges. It aims to provide insights to advance privacy-respecting software.

Summary

This book chapter examines the evolving role of program analysis in software privacy. It discusses the ambiguity in regulations like GDPR for identifying personal data in code. Different program analysis techniques are explored, like taint analysis to trace data flows. Benefits are highlighted, such as automated policy compliance checks and streamlining code reviews. However, challenges remain, including classifying ambiguous forms of personal data. Potential research directions are identified, like employing machine learning to categorize data flows. Overall, the chapter provides a nuanced overview of applying program analysis to software privacy. While acknowledging limitations, it offers insights into existing techniques and future opportunities to advance privacy protections in software systems.

CHAPTER SIX

DISCUSSION

In this section, we first summarize the contributions of this dissertation in relation to the research questions presented in Chapter 1. We then situate these contributions within the broader field of privacy protection, and finally discuss their relevance to relevant social topics.

6.1 The results as a whole

In this section, we revisit the primary contributions and research questions outlined in Section 1.2 and 1.3 to provide an overview of the results. Our research offers a multi-faceted understanding of personal data management in software applications, particularly Java/Android applications. The contributions can be organized into four main categories: (1) defining personal data in software, (2) methods for identifying personal data, (3) mapping and categorizing personal data flow, and (4) practical applications in privacy-related tasks such as DPIA and privacy code review.

The first part of our contributions revolves around defining what constitutes personal data within the realm of software. This is crucial because the boundaries of personal data have expanded beyond traditional identifiers like names and email addresses. Our research provides a nuanced framework that includes categories like biometric data, thereby enhancing the understanding of what needs to be protected under privacy laws like GDPR.

The second part focuses on the methods we developed for identifying personal data within software applications. We introduced an automated approach that employs static analysis techniques to scan Java and Android applications for potential personal data. This method not only identifies traditional forms of personal data but also flags biometric data, thereby filling a gap in existing approaches.

Our third contribution is a privacy-flow-graph that visualizes how personal data moves within an application. This is particularly useful for DPIA processes, as it allows for a more granular understanding of data flow, especially concerning I/O user input. This method advances the field by providing a tool that can be used to identify potential vulnerabilities or compliance issues.

The fourth and final layer of our contributions is the application of our findings to real-world privacy-related tasks. We have developed techniques that can be integrated into existing DPIA processes and code review practices. These techniques not only make these processes more efficient but also ensure that they are more effective in identifying and mitigating privacy risks.

6.2 Linking results to the research questions

To assess how the results address the original research aims, we will revisit the research questions outlined in Section 1.2. We will go through each research question individually, analyzing how the findings provide insights related to the question and associated sub-questions.

RQ1: What constitutes personal data within the context of software?

Our work offers a framework for defining personal data within software, particularly Java/Android applications. We developed a list of personal data sources in Paper 1 that can be identified through I/O methods, thereby providing a structured way to understand data entry points. Additionally, we engaged in a discussion on whether behavioral interaction data, such as mouse clicks and keystrokes, should be classified as personal data. This debate in Paper 5 enriches the understanding of personal data by considering non-traditional forms, thereby expanding the scope of what needs to be protected under privacy laws like GDPR.

RQ2: How to identify personal data in software?

To locate where personal data resides in software, we introduced methods for identifying personal data in software. We developed a list in Paper 3 and 4 that can identify personal data in source code based on regular expression rules. We also proposed an automated approach in Paper 1 that employs static analysis to scan Java and Android applications for potential personal data acquired from users.

RQ3: How to trace the flow of personal data and categorize these flows in software?

RQ3 involved the adoption of program analysis in order to taint personal data, identify their flows, and further classify them. We developed a privacy flow-graph in Paper 1 that visualizes how personal data moves within an application. This tool is particularly useful for Data Protection Impact Assessment (DPIA) processes, as it allows for a more granular understanding of data flow, especially concerning user I/O.

By categorizing these flows based on the different types of processing involved, we offer methods that can identify potential vulnerabilities or compliance issues specific to various contexts and data types. For example, our methods can address issues related to biometric data (Paper 2), general personal data processing in source code (Papers 4 and 7), and user interaction data (Papers 5 and 6). This advances the field by providing a more structured approach to data flow analysis that is sensitive to the nature of the data being processed.

RQ4: How to apply the findings to support various privacy-related tasks?

Results from the papers can provide answers to the main question and the sub-questions.

In response to RQ4, we demonstrated how our findings could be applied to real-world privacy tasks. We developed methods that can be integrated into existing DPIA processes (RQ4.1, Paper 1) and code review practices (RQ4.2, Papers 3, 4 and 7). These methods streamline the tasks for code reviewers, enabling them to complete their assessments in a shorter amount of time without compromising on the quality of the review. They also improve the effectiveness of both DPIA and code review processes by providing more accurate identification and mitigation of privacy risks.

We also proposed methods which can be used to automatically extract relevant parts in privacy policies and actual implementation in code to verify the policy (RQ4.3, Papers 5 and 6). Our contributions in this area offer both theoretical insights and practical methods for better data management, thereby fulfilling the need for actionable outcomes based on our research findings.

6.3 Linking results to fields of research

This dissertation contributes to multiple academic and practical areas, such as software privacy, program analysis, and policy analysis. In software privacy, our research clarifies the types of data considered personal within software applications. The term *software privacy* is often cited in scholarly works but lacks a universally accepted definition [72, 73]. Our research adds to this discussion by proposing methods to identify personal data in software, especially in the Internet of Things (IoT) context.

The rise of IoT makes our work timely, as the scope of what is considered personal data is continuously expanding. The privacy flow-graph we introduce can be a useful tool for DPIA, offering a systematic way to assess privacy risks in IoT data flows.

In program analysis, our research provides methods for identifying and categorizing personal data flows. These methods are adaptable to different software types, enhancing their broad applicability. In policy analysis, our work can guide the formulation of robust privacy regulations, bridging the technical and legal aspects [74, 75].

Additionally, the methods we propose for identifying and categorizing personal data can be integrated into the software development life cycle. This facilitates compliance with privacy regulations like the GDPR and opens avenues for developing privacy-preserving design patterns applicable to various programming languages and platforms.

Regarding program analysis, our techniques offer new perspectives for analyzing different types of data flows. Our source code analysis technique is versatile, applicable to multiple programming languages and various codebase sizes. It also allows for personalized rules for identifying personal data, offering scalability. This adaptability suggests that our methods could be extended to detect other software vulnerabilities and code smells, thereby enhancing software quality and security.

From a policy standpoint, our research offers a structured approach to evaluate the privacy implications of software systems. This is particularly relevant given the growing legislative focus on digital privacy, as seen in laws like the GDPR and CCPA. Our work can inform and shape more effective privacy policies and regulations.

Lastly, our research has interdisciplinary applications, facilitating collaboration among computer scientists, legal experts, and policymakers. The adaptable solutions we provide for analyzing personal data in software can benefit various fields, including software engineering, security, law, and policy. These techniques could be used to inform policy debates, assist companies in identifying compliance gaps, or guide developers in building privacy-preserving features.

6.4 Linking results to social topics

Our research has implications for addressing privacy issues that are often overlooked in society. One of the most pressing concerns is the extensive collection of interaction data by various software applications. While users may be aware that their personal information is being collected, they often overlook the extent to which their interactions — such as clicks, scrolls, and time spent on specific tasks — are also being monitored and stored. Our work in identifying and categorizing personal data flows in apps can raise awareness for users and policy makers on the need to enforce regulations around collecting interaction data.

Moreover, our research can help inform public discourse on the ethical considerations surrounding data privacy. By providing methods to trace and categorize data flows, we offer tools that can be used to assess the ethical implications of data collection and processing. This is particularly relevant in the age of big data, where massive datasets are often used without adequate scrutiny of the potential privacy risks involved.

Our work also has the potential to influence consumer behavior. As users become more aware of the types of data being collected and how it is being used, they may become more selective in the software and services they choose to engage with. This could drive a market shift towards more privacy-conscious products, thereby encouraging software developers to prioritize privacy in their design and development processes.

Lastly, our research can serve as a resource for educational initiatives aimed at improving digital literacy, particularly in the context of data privacy. By understanding the technical aspects of how personal data is processed, individuals can make more informed decisions about their online activities, contributing to a more privacy-aware society.

CHAPTER

SEVEN

CONCLUSION

There is no end of learning until
one is encased in coffin.
「學而不已，闔棺乃止。」

Confucius 《孔子·韓詩外傳》

7.1 Conclusion

This dissertation has focused on addressing gaps in the field of software privacy, particularly in the identification, categorization, and tracing of personal data within software applications. We started by examining what types of data can be considered personal in a software context. Through the development of a privacy flow-graph, we have expanded the understanding of personal data to include interaction data, aligning with the first challenge mentioned in the abstract.

We then introduced methods for identifying personal data within software applications. These methods, such as regular expression matching and biometric API usage detection, aim to help software applications comply with privacy regulations like the GDPR. These methods align with the second challenge in the abstract and offer practical tools for developers while laying the groundwork for future research.

Next, we developed techniques for visualizing and categorizing data flows within software systems. These methods are useful for Data Protection Impact Assessments (DPIA) and other privacy-related tasks. This aligns with the DPIA-focused contribution in the abstract and is relevant in a data-driven world where understanding data movement is essential for compliance and ethical considerations.

We also demonstrated how our methods could be applied to various privacy-related tasks, such as improving code reviews and policy analysis. This aligns with the automated code review assistant mentioned in the abstract and has practical applications in multiple areas.

Beyond academic contributions, our work has societal implications. It can inform public discussions on less-discussed privacy issues, like the capture of interaction data, and potentially influence regulatory changes. Additionally, our methods offer ways to encourage a more privacy-conscious consumer base and software development community.

Lastly, our work facilitates collaboration among computer scientists, legal experts, and policymakers. It addresses various aspects of software privacy, laying a foundation for interdisciplinary work aimed at creating a more secure and privacy-respecting digital environment.

In conclusion, this dissertation has addressed several key issues in software privacy. While we have made strides in this area, the continually evolving nature of technology means that new challenges will always arise. We hope this work will serve as a basis for future research and practical applications, contributing to a more privacy-aware digital world.

7.2 Future directions

The ever-changing world of software development and privacy regulations calls for ongoing research and innovation in software privacy. As we look to the future, several key research areas and challenges stand out for both academia and industry.

7.2.1 Redefining and expanding the scope of personal data

The traditional scope of personal data, often limited to explicit identifiers like names and email addresses, is increasingly challenged by emerging technologies such as the IoT [76]. A pressing issue is the categorization of *non-personal data*, like aggregated user interactions, which can become personally identifiable when combined with other data types. This complexity calls for a re-evaluation of what constitutes personal data in the context of modern software systems.

A concrete research problem for future scholars could be the development of program analysis tools that can identify and assess *composite personal data*. This refers to data that becomes personally identifiable only when aggregated, posing unique challenges for privacy assessments. For instance, how should a system that collects both movement data and shopping history be evaluated for privacy risks when each data type alone might be considered non-personal after de-identification?

To address these evolving challenges, future research should focus on creating program analysis tools adaptable to new, nuanced definitions of personal data. These tools should be capable of analyzing a wide range of software domains, from web applications to IoT, thereby ensuring comprehensive and up-to-date privacy assessments.

7.2.2 Bridging the gap between legal and technical experts

The challenge in software privacy is that technical experts can analyze code but often lack legal knowledge, while legal experts may not understand code. This creates a gap in effective collaboration. Current tools mainly serve technical experts and offer little for legal professionals [77].

Current program analysis tools primarily target technical experts and do not provide an interface that legal experts can easily use. This lack of a common platform blocks effective collaboration between the two groups. Such platform should not only be able to translate technical findings into legal terms but also allow legal experts to input their requirements, which can then be translated into technical criteria for program analysis.

Future research could focus on creating program analysis tools equipped with user interfaces specifically designed for legal experts. These interfaces could facilitate a two-way translation between technical and legal languages. For instance, legal requirements could be converted into configurable rules for program analysis, and the results of the analysis could be mapped back to legal categories. To achieve this, advanced techniques such as natural language processing and machine learning could be employed to enable seamless communication between code analysis and legal assessments.

7.2.3 Real-time analysis

The current methods in our research are not designed for real-time feedback, which is important as software development moves towards continuous integration and deployment [78]. While we have provided techniques to help reviewers and developers quickly gain an overview of personal data locations and processing in code, running such analyses cannot happen in real-time during active development or reviewing tasks.

Future research should aim to develop real-time program analysis techniques that can be integrated into Integrated Development Environments. The goal would be to have privacy analyses occur concurrently as developers write code, allowing them to immediately see when personal data is likely being accessed or manipulated.

For example, IDE plugins could use lightweight static analysis to highlight personal data flows and accesses as they are coded. Developers could then appropriately mark or document these areas to streamline future privacy tasks like ROPA creation.

To be practical, these real-time methods would need to be efficient to provide feedback within seconds, without disrupting coding flow. Accuracy would also be crucial to avoid many false positives. Achieving this combination of speed and precision poses research challenges but promises to greatly assist developers in building privacy directly into applications.

7.2.4 Machine learning for data flow classification

The problem with our current rule-based methods for data flow classification is their limited adaptability to new types of data flows [79]. These rule-based systems are confined to a static taxonomy, which makes them less effective for classifying data flows in software that utilize unconventional APIs or have unique data processing patterns.

The problem becomes even more complex when considering software diversity. For instance, how can a classification system accurately categorize data flows in a healthcare app versus a social media app? Each domain has its own specific types of data and processing activities, requiring a classification system that can adapt accordingly.

By training machine learning models on large datasets of software code, systems could learn to automatically classify personal data flows without relying solely on predefined rules. The models could infer categories of processing directly from data flow features and context. With sufficient training data encompassing diverse software types and languages, the models could develop a nuanced understanding of how different applications handle personal data. They could then automatically provide flow categorizations tailored to the specific software, beyond our static taxonomy.

This data-driven approach, together with potential real-time analysis, would greatly benefit developers and reviewers. By automatically generating insightful flow categories directly from code, machine learning would streamline compliance tasks and augment human analysis with adaptable, large-scale pattern recognition.

7.2.5 Enhanced alignment of program and policy analysis

Our current approach to privacy analysis is limited by its focus on user interaction data, leaving a gap in the analysis of many other diverse types of personal data in privacy policies. This limitation poses a concrete challenge: How can we develop methods that not only identify but also categorize a broader range of personal data types mentioned subtly in privacy policies?

The challenge extends to the dynamic nature of privacy policies and regulations. As these policies evolve, how can program analysis techniques adapt in real-time to ensure ongoing compliance? This is particularly crucial for software that undergoes frequent updates or operates in regulatory environments with changing privacy laws.

Future work could develop large language models tailored for privacy policies and regulations. These could identify key features in legal text related to personal data processing and consent. The extracted details could be translated into configurable rules to guide dynamic program analysis aligned with each policy's specifics.

By creating a tight feedback loop between policy interpretation and tailored software analysis, we can work towards ensuring alignment between privacy promises and system behaviors. This co-design of policy analytic and program analysis techniques shows promise in verifying that privacy commitments are fulfilled in implementation.

REFERENCES

[1] Alan F. Westin. "Privacy and freedom". In: *Washington and Lee Law Review* 25.1 (1968), p. 166.

[2] Joel R Reidenberg et al. "Disagreeable privacy policies: Mismatches between meaning and users' understanding". In: *Berkeley Tech. LJ* 30 (2015), p. 39.

[3] Daniel J. Solove. "Privacy self-management and the consent dilemma". In: *Harvard Law Review* 126 (2012), p. 1880.

[4] David Thompson. "I Agreed to What-A Call for Enforcement of Clarity in the Presentation of Privacy Policies". In: *Hastings Comm. & Ent. LJ* 35 (2012), p. 199.

[5] Feiyang Tang and Bjarte M. Østvold. "Assessing Software Privacy Using the Privacy Flow-Graph". In: *Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security.* MSR4P&S 2022. Singapore: Association for Computing Machinery, 2022, pp. 7–15. ISBN: 9781450394574. DOI: 10.1145/3549035.3561185.

[6] Feiyang Tang. "PABAU: Privacy Analysis of Biometric API Usage". In: *The 8th IEEE International Conference on Privacy Computing (PriComp 2022).* IEEE, Dec. 2022. DOI: 10.1109/smartworld-uic-atc-scalcom-digitaltwin-pricomp-metaverse56740.2022.00327.

[7] Feiyang Tang, Bjarte M. Østvold, and Magiel Bruntink. "Identifying Personal Data Processing for Code Review". In: *arXiv preprint arXiv:2301.01568* (2023). DOI: 10.5220/0011725700003405.

[8] Feiyang Tang, Bjarte M. Østvold, and Magiel Bruntink. "Helping Code Reviewer Prioritize: Pinpointing Personal Data and its Processing". In: *arXiv preprint arXiv:2306.11495* (2023). DOI: 10.3233/FAIA230228.

[9] Feiyang Tang and Bjarte M. Østvold. "Transparency in App Analytics: Analyzing the Collection of User Interaction Data". In: *arXiv preprint arXiv:2306.11447* (2023). DOI: 10.1109/PST58708.2023.10320181.

[10] Feiyang Tang and Bjarte M. Østvold. *User Interaction Data in Apps: Comparing Policy Claims to Implementations.* 2023. arXiv: `2312.02710 [cs.SE]`.

[11] Feiyang Tang and Bjarte M. Østvold. *Finding Privacy-relevant Source Code.* To be appeared in the 2nd International Workshop on Mining Software Repositories Applications for Privacy and Security. 2024. arXiv: `2401.07316 [cs.SE]`.

[12] Feiyang Tang and Bjarte M. Østvold. "Software Privacy and Program Analysis: Insights, Methods, and Opportunities". A book chapter submitted to the Springer Handbook on Privacy and Security Matters in Biometric Technologies. 2024.

[13] Meredydd Williams, Jason R. C. Nurse, and Sadie Creese. "The Perfect Storm: The Privacy Paradox and the Internet-of-Things". In: *2016 11th International Conference on Availability, Reliability and Security (ARES).* 2016, pp. 644–652. DOI: `10.1109/ARES.2016.25`.

[14] Michela Iezzi. *The Evolving Path of "the Right to Be Left Alone" - When Privacy Meets Technology.* 2021. arXiv: `2111.12434 [cs.CR]`.

[15] Christina B Whitman. "Privacy in Early Confucian and Taoist Thought". In: *Individualism and Holism: Studies in Confucian and Daoist Values* (1985).

[16] Isabel Wagner. "Privacy Policies across the Ages: Content of Privacy Policies 1996–2021". In: *ACM Trans. Priv. Secur.* 26.3 (May 2023). ISSN: 2471-2566. DOI: `10.1145/3590152`.

[17] Ghazaleh Beigi et al. "Protecting User Privacy: An Approach for Untraceable Web Browsing History and Unambiguous User Profiles". In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining.* WSDM '19. Melbourne VIC, Australia: Association for Computing Machinery, 2019, pp. 213–221. ISBN: 9781450359405. DOI: `10.1145/3289600.3291026`.

[18] Omer Tene and Jules Polonetsky. "Big data for all: Privacy and user control in the age of analytics". In: *Nw. J. Tech. & Intell. Prop.* 11 (2012), p. 239.

[19] Hannes Grassegger and Mikael Krogerus. "The data that turned the world upside down". In: *Vice Motherboard* 28 (2017).

[20] Rebecca Balebako et al. ""Little brothers watching you" raising awareness of data leaks on smartphones". In: *Proceedings of the Ninth Symposium on Usable Privacy and Security.* 2013, pp. 1–11.

[21] Alessandro Acquisti, Laura Brandimarte, and George Loewenstein. "Privacy and human behavior in the age of information". In: *Science* 347.6221 (2015), pp. 509–514.

[22] Juan Luis Herrera et al. *Personal Data Gentrification.* 2021. arXiv: `2103.17109 [cs.CY]`.

[23] Sarah Spiekermann and Alexander Novotny. "A vision for global privacy bridges: Technical and legal measures for international data markets". In: *Computer Law & Security Review* 31.2 (2015), pp. 181–200.

[24] Shuyuan Zheng, Yang Cao, and Masatoshi Yoshikawa. *Trading Location Data with Bounded Personalized Privacy Loss.* 2019. arXiv: `1906.05457 [cs.CR]`.

[25] Zhiqi Bu et al. "Deep Learning With Gaussian Differential Privacy". In: *Harvard Data Science Review* 2.3 (Sept. 2020).

[26] Beatrice Li, Arash Tavakoli, and Arsalan Heydarian. "Occupant privacy perception, awareness, and preferences in smart office environments". In: *Scientific Reports* 13.1 (2023), p. 4073.

[27] Rachana Nget, Yang Cao, and Masatoshi Yoshikawa. "How to balance privacy and money through pricing mechanism in personal data market". In: *arXiv preprint arXiv:1705.02982* (2017).

[28] Shuyuan Zheng, Yang Cao, and Masatoshi Yoshikawa. "Pricing Private Data with Personalized Differential Privacy and Partial Arbitrage Freeness". In: *arXiv preprint arXiv:2105.01651* (2021).

[29] David J. Phillips. *The Influence of Policy Regimes on the Development and Social Implications of Privacy Enhancing Technologies.* 2001. arXiv: `cs/0109098 [cs.CY]`.

[30] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis.* Springer, 1999.

[31] Brian Chess and Jacob West. *Secure Programming with Static Analysis.* Addison-Wesley Professional, 2007.

[32] Al Bessey et al. "A few billion lines of code later: Using static analysis to find bugs in the real world". In: *Communications of the ACM* 53.2 (2010), pp. 66–75.

[33] Li Li et al. "IccTA: Detecting Inter-Component Privacy Leaks in Android Apps". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering.* Vol. 1. IEEE. 2015, pp. 280–291.

[34] Steven Arzt et al. "FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps". In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation.* ACM. 2014, pp. 259–269.

[35] Fengguo Wei et al. “Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2014, pp. 1329–1341.

[36] Flemming Nielson, Hanne R Nielson, and Chris Hankin. *Principles of program analysis*. Springer, 2015.

[37] Benjamin Livshits, John Whaley, and Monica S Lam. “Reflection analysis for Java”. In: *Programming Languages and Systems: Third Asian Symposium, APLAS 2005, Tsukuba, Japan, November 2-5, 2005. Proceedings 3*. Springer. 2005, pp. 139–160.

[38] Ondrej Lhoták and Kwok-Chiang Andrew Chung. “Points-to analysis with efficient strong updates”. In: *Proceedings of the 38th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 2011, pp. 3–16.

[39] Esben Andreasen et al. “A survey of dynamic analysis and test generation for JavaScript”. In: *ACM Computing Surveys (CSUR)* 50.5 (2017), pp. 1–36.

[40] Simon Holm Jensen, Anders Møller, and Peter Thiemann. “Type analysis for JavaScript”. In: *Static Analysis: 16th International Symposium, SAS 2009, Los Angeles, CA, USA, August 9-11, 2009. Proceedings 16*. Springer. 2009, pp. 238–255.

[41] Matıéas Toro, Ronald Garcia, and Éric Tanter. “Type-driven gradual security with references”. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 40.4 (2018), pp. 1–55.

[42] Veselin Raychev, Martin Vechev, and Andreas Krause. “Predicting program properties from “big code””. In: *ACM SIGPLAN Notices* 50.1 (2015), pp. 111–124.

[43] Aleecia M. McDonald and Lorrie Faith Cranor. “The cost of reading privacy policies”. In: *ISJLP*. Vol. 4. 2008, p. 543.

[44] Sanda Erdelez and Abhijit Bhowmick. “Readability of Privacy Policies of Healthcare Websites”. In: *Online Journal of Public Health Informatics* 12.1 (2020).

[45] Sebastian Zimmeck et al. “Automated analysis of privacy requirements for mobile apps”. In: *Twenty-Sixth Annual Network and Distributed System Security Symposium*. 2017.

[46] Kirsten E. Martin. “Transaction costs, privacy, and trust: The laudable goals and ultimate failure of notice and choice to respect privacy online”. In: *First Monday* (2013).

[47] Fred H. Cate. “The failure of fair information practice principles”. In: *Consumer protection in the age of the information economy* (2006), p. 341.

[48] Rocky Slavin et al. “Toward a framework for detecting privacy policy violations in android application code”. In: *Proceedings of the 38th International Conference on Software Engineering*. 2016, pp. 25–36.

[49] Cristian-Alexandru Staicu et al. “An empirical study of information flows in real-world javascript”. In: *Proceedings of the 14th ACM SIGSAC Workshop on Programming Languages and Analysis for Security*. 2019, pp. 45–59.

[50] Goran Piskachev et al. “Fluently specifying taint-flow queries with fluent TQL”. In: *Empirical Software Engineering* 27.5 (2022), p. 104.

[51] Welderufael B. Tesfay et al. “PrivacyGuide: Towards an Implementation of the EU GDPR on Internet Privacy Policy Evaluation”. In: *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*. IWSPA ’18. Tempe, AZ, USA: Association for Computing Machinery, 2018, pp. 15–21. ISBN: 9781450356343.

[52] Rohan Ramanath et al. “Unsupervised alignment of privacy policies using hidden markov models”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. 2014, pp. 605–610.

[53] Abhilasha Ravichander et al. “Breaking Down Walls of Text: How Can NLP Benefit Consumer Privacy?” In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. Vol. 1. 2021.

[54] Sebastian Zimmeck, Rafael Goldstein, and David Baraka. “PrivacyFlash Pro: Automating Privacy Policy Generation for Mobile Apps.” In: *NDSS*. 2021.

[55] Zhengyang Qu et al. “Autocog: Measuring the description-to-permission fidelity in android applications”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 2014, pp. 1354–1365.

[56] Marco Bardus et al. “Data management and privacy policy of COVID-19 contact-tracing apps: Systematic review and content analysis”. In: *JMIR mHealth and uHealth* 10.7 (2022), e35195.

[57] Marcel von Maltitz, Cornelius Diekmann, and Georg Carle. “Privacy Assessment of Software Architectures based on Static Taint Analysis”. In: *arXiv preprint arXiv:1608.04671* (2016).

[58] Pankaj Kohli. "Coarse-grained Dynamic Taint Analysis for Defeating Control and Non-control Data Attacks". In: *arXiv preprint arXiv:0906.4481* (2009).

[59] David Leoni. "Non-interactive differential privacy: a survey". In: *Proceedings of the First International Workshop on Open Data*. 2012, pp. 40–52.

[60] Ninghui Li, Wahbeh Qardaji, and Dong Su. "On sampling, anonymization, and differential privacy or, k-anonymization meets differential privacy". In: *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. 2012, pp. 32–33.

[61] Josep Domingo-Ferrer and Jordi Soria-Comas. "From t-closeness to differential privacy and vice versa in data anonymization". In: *Knowledge-Based Systems* 74 (2015), pp. 151–158.

[62] Martin Kroll. "Pointwise adaptive kernel density estimation under local approximate differential privacy". In: *arXiv preprint arXiv:1907.06233* (2019).

[63] David Sánchez et al. "Utility-preserving differentially private data releases via individual ranking microaggregation". In: *Information Fusion* 30 (2016), pp. 1–14.

[64] Shomir Wilson et al. "The creation and analysis of a website privacy policy corpus". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2016, pp. 1330–1340.

[65] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. "Code review quality: How developers see it". In: *Proceedings of the 38th international conference on software engineering*. 2016, pp. 1028–1038.

[66] Alifia Puspaningrum et al. "Vulnerable Source Code Detection using SonarCloud Code Analysis". In: *arXiv preprint arXiv:2307.02446* (2023).

[67] Awanthika Senarath and Nalin Asanka Gamagedara Arachchilage. "Why developers cannot embed privacy into software systems? An empirical investigation". In: *arXiv preprint arXiv:1805.09485* (2018).

[68] Michael J May, Carl A Gunter, and Insup Lee. "Privacy APIs: Access control techniques to analyze and verify legal privacy policies". In: *19th IEEE Computer Security Foundations Workshop (CSFW'06)*. IEEE. 2006, 13–pp.

[69] Yue Xiao et al. "Lalaine: Measuring and characterizing non-compliance of apple privacy labels at scale". In: *arXiv preprint arXiv:2206.06274* (2022).

[70] Patrick Lam et al. "The Soot framework for Java program analysis: a retrospective". In: *Cetus Users and Compiler Infastructure Workshop (CETUS 2011)*. Vol. 15. 35. 2011.

[71] Sebastian Zimmeck et al. "Maps: Scaling privacy compliance analysis to a million apps". In: *Proceedings on Privacy Enhancing Technologies* 2019 (2019), p. 66.

[72] Abdulrahman Hassan Alhazmi, Mumtaz Abdul Hameed, and Nalin Asanka Gamagedara Arachchilage. "Developers' Privacy Education: A game framework to stimulate secure coding behaviour". In: *arXiv preprint arXiv:2211.03498* (2022).

[73] Marcel von Maltitz, Cornelius Diekmann, and Georg Carle. "Privacy Assessment of Software Architectures based on Static Taint Analysis". In: *arXiv preprint arXiv:1608.04671* (2016).

[74] Leonardo Horn Iwaya, Muhammad Ali Babar, and Awais Rashid. "Privacy Engineering in the Wild: Understanding the Practitioners' Mindset, Organisational Aspects, and Current Practices". In: *IEEE Transactions on Software Engineering* (2023), pp. 1–26. DOI: 10.1109/TSE.2023.3290237.

[75] Blagovesta Kostova, Seda Gürses, and Carmela Troncoso. "Privacy Engineering Meets Software Engineering. On the Challenges of Engineering Privacy ByDesign". In: *arXiv preprint arXiv:2007.08613* (2020).

[76] Z Berkay Celik et al. "Program analysis of commodity IoT applications for security and privacy: Challenges and opportunities". In: *ACM Computing Surveys (CSUR)* 52.4 (2019), pp. 1–30.

[77] Kalle Hjerppe, Jukka Ruohonen, and Ville Leppänen. "Annotation-based static analysis for personal data protection". In: *IFIP International Summer School on Privacy and Identity Management*. Springer, 2019, pp. 343–358.

[78] Gilles Barthe et al. "Deciding Differential Privacy for Programs with Finite Inputs and Outputs". In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '20. Saarbrücken, Germany: Association for Computing Machinery, 2020, pp. 141–154. ISBN: 9781450371049. DOI: 10.1145/3373718.3394796.

[79] Elisabet Lobo-Vesga, Alejandro Russo, and Marco Gaboardi. *A Programming Framework for Differential Privacy with Accuracy Concentration Bounds*. 2019. arXiv: 1909.07918 [cs.CR].

Part B

The Papers

PAPER 1

Assessing Software Privacy using the Privacy Flow-Graph

Tang, F. and Østvold, B. (2022). Assessing software privacy using the privacy flow-graph. In Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security (MSR4P&S 2022). Association for Computing Machinery, New York, NY, USA, 7–15.

Assessing Software Privacy using the Privacy Flow-Graph[⋆]

Feiyang Tang and Bjarte M. Østvold

Norwegian Computing Center, Oslo, Norway
`{feiyang,bjarte}@nr.no`

Abstract. We increasingly rely on digital services and the conveniences they provide. Processing of personal data is integral to such services and thus privacy and data protection are a growing concern, and governments have responded with regulations such as the EU's GDPR. Following this, organisations that make software have legal obligations to document the privacy and data protection of their software. This work must involve both software developers that understand the code and the organisation's data protection officer or legal department that understands privacy and the requirements of a Data Protection and Impact Assessment (DPIA).

To help developers and non-technical people such as lawyers document the privacy and data protection behaviour of software, we have developed an automatic software analysis technique. This technique is based on static program analysis to characterise the flow of privacy-related data. The results of the analysis can be presented as a graph of privacy flows and operations—that is understandable also for non-technical people. We argue that our technique facilitates collaboration between technical and non-technical people in documenting the privacy behaviour of the software. We explain how to use the results produced by our technique to answer a series of privacy-relevant questions needed for a DPIA. To illustrate our work, we show both detailed and abstract analysis results from applying our analysis technique to the secure messaging service Signal and to the client of the cloud service NextCloud and show how their privacy flow-graphs inform the writing of a DPIA.

Keywords: Program analysis · Data protection and privacy · GDPR · Software design documentation.

1 Introduction

Privacy has been widely discussed in recent years — with the rise in public awareness and associated legislative developments, guaranteeing privacy while processing large amounts of private user data has become an important topic. Following recent law implementations such as the GDPR, we now have a regulated and clear framework for ensuring privacy compliance, which mandates documenting software properties through, for example, a Data Privacy Impact Assessment (DPIA). Such an examination must include all parts of the software and it requires a grasp of the software as well as sufficient technical knowledge to analyse the implementation. As a result, we would anticipate a development team expert who has a brief grasp of the implementation while also having sophisticated analysis and tools at their disposal to assist ensure that critical questions in evaluation frameworks such as DPIA can be answered.

The reality, however, is considerably different. While having a privacy compliance checking process operating alongside a software development life cycle is important, analysis and tools at the code level with tailored assistance to legal experts are insufficient. In the meantime, DPIA questions require an understanding of both technical and legal aspects. This means that performing a successful DPIA cannot be done exclusively by a non-technical Data Protection Officer (DPO) who specialises in data protection policy or a technical professional from the data controller (e.g.,

[⋆] Published at *the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security* (MSR4P&S 2022), DOI: 10.1145/3549035.3561185 a developer in the service provider organisation) with programming experience. Simultaneously, it is difficult for developers to keep track of every single change in terms of private data processing among hundreds of lines of code.

This raises the following question: how can we help both technical developers (from or work for data controllers) and non-technical (DPOs) individuals examine privacy compliance in software? Since tracking the flow of data originating from users is important for privacy protection, we must check sensitive user inputs to the software and use an explainable abstraction to illustrate the privacy behaviours in the software, address privacy elements, and provide assistance in producing a better privacy analysis.

We propose privacy flow-graphs as a means to help both developers and DPOs, they can adopt our technique to discover privacy-related behaviours in software. Such graphs produced by our technique enable documenting private data processing actions, assist organisations (the data controller) in showing compliance with their duties and assist the DPO in carrying out its missions. Illustrating the processes may also assist developers to construct more privacy-compliant software and achieve privacy-by-design throughout development and deployment.

Our contributions are:

– The definition of the privacy flow-graph (Section 3.2)
– How to write a DPIA informed by the privacy flow-graph (Section 4).
– A static program analysis that builds the privacy flow-graphs for Java programs (Section 5).

We demonstrate the utility of our research by examining privacy-related trends in two well-known Java applications: Signal and NextCloud (Section 6).

2 Motivation

Examining data protection compliance is essential for the vast majority of software released to the market, as well as for every service update when new user data must be analysed or when the way data is handled changes. Legal regulations such as the GDPR necessitate that legal experts obtain detailed privacy-related information processes from software developers. This implementation-specific information is typically obtained through manual labour by developers, and may not include all that a legal expert needs.

However, there are developers that are unfamiliar with the existing software and might have difficulties providing in-depth information to legal experts.

This circumstance motivated us to design a lightweight, semi-automated program analysis technique that automatically analyses how and where personal data is accessed and processed, therefore providing software developers and DPOs with a great deal of ease.

3 Preliminaries

In this section, we describe the preliminary aspects of our analysis: the local and global data-flow, the privacy flow-graph, the source and sink methods, and the handcrafted datasets we created to support the analysis.

Let c, d denote classes, n, m methods, and let notation $c.m$ make explicit that class c that declares m. We assume that method names are unique in a class.

3.1 Local Data-Flow in Methods

We define some notation to refer to results obtainable from the control flow graph (CFG) of a method. These results concern the kind of values that may flow between various points either inside the method body.

Definition 1 (Method data-flow point p). *A data-flow point p associated with a method $c.m$ is one of the following:*

`start` – *the start of the method;*
`invoke` $d.n_{\,i}$ – *an invocation of method $d.n$;*
$\texttt{i\_primitive}_i$ – *an input primitive;*
$\texttt{o\_primitive}_i$ – *an output primitive;*
$\texttt{return}_i$ – *a return statement.*

Definition 2 (Local data-flow F; beginning, end). *Let p, p' be data-flow points, let F be $p \mapsto p'$ and let $c.m$ be a method. We write $F \dashv CFG(c.m)$ to means that the control-flow-graph of $c.m$ specifies a* local data-flow *F, that is, that values may flow from p to p'. We refer to p as the* beginning *of F, denoted $begin(F)$ and p' as the* end *of F, denoted $end(F)$.*

An invocation can be both a beginning and an end of a flow, whereas the start of the method and an input primitive can only be a beginning, and a return statement and an output primitive can only be an end.

We are concerned with all data-flows that originate from the use of an input primitive. We now define some particular types of flows.

Definition 3 (Source flow, F^o). *Given method $c.m$ where*
$(\texttt{i\_primitive}_i \mapsto \texttt{return}_j) \dashv CFG(c.m)$*. This flow is called a* source flow*, denoted F^o.*

Definition 4 (Sink flow, F^i). *Given method $c.m$ where*
$(\texttt{start} \mapsto \texttt{o\_primitive}_i) \dashv CFG(c.m)$*. The flow is called a* sink flow*, denoted F^i.*

3.2 Global Data-Flow & the Privacy Flow-Graph

We now consider global data-flow, specifically data-flows between methods of different classes, those are, all data-flows that start from the use of an input primitive.

We extend the concept of a data-flow from local flows F inside methods to global flows G across methods. A global data-flow is defined by a series of local data-flows, each corresponding to a method invocation, and that satisfies certain conditions.

Definition 5 (Global data-flow G). *A* global data-flow *G is finite series of two or more local data flows, $F_1 \cdots F_n$. The notions of beginning and end extend to G in an obvious way. Furthermore, any F_k, F_{k+1} above must satisfy the following: Let $c_k.m_k$ be such that $F_k \dashv CFG(c_k.m_k)$ and $c_{k+1}.m_{k+1}$ such that $F_{k+1} \dashv CFG(c_{k+1}.m_{k+1})$ and $end(F_k) = \texttt{return}_i$ and $begin(F_{k+1}) =$ `invoke` $c_k.m_{k\,j}$ for some i, j.*

A global data-flow $G = F_1 \ldots F_n$ is a *privacy flow* if F_1 is a source flow. We are especially interested in global data-flows that involve data from input primitives ending up in output primitives.

Let P be a program with privacy flows $G_1, \ldots, G_n$. The *privacy flow-graph* is a graph where there the nodes are all methods involved in a privacy flow and the edges are pairs of methods involved in successive flows F_k, F_{k+1} part of some G_j.

Java specifics Here we consider some issues in adapting our data-flow definitions to Java.

First, we define rich types with the intuition that we are only interested in flows that involve values of these kinds of types.

Definition 6 (Rich type). *A* rich type *is any of following: the primitive data types* `string`*,* `int`*,* `byte`*, the object types, as well as arrays of rich types.*

Values of rich types are those values that may contain privacy-related information. In principle, a *boolean* could also be relevant to privacy, but we limit our scope to the rich types to simplify our task. We are concerned with the processing of privacy-related data and not with the leakage of bits of privacy information stemming from such processing.

All non-trivial programs refer to either standard libraries or third-party libraries and thus source flows and sink flows may take place inside the methods of these libraries. In order to include these flows without analyzing the libraries, we introduce the concept of source methods and sink methods where such flows happen, and we apply a separate library analysis to pre-build a collection of source and sink methods.

A *source method* is a method whose invocation results in a source flow, and we denote it as *om*. A *sink method* is a method whose invocation results in a sink flow, denoted *im*.

Library analysis We have manually constructed a dataset of source and sink methods in the native Java library[1] as well as the most used third-party Java libraries across different categories[2]. The third-party libraries were selected from the Maven Repository list based on their download frequency[3]. There are 158 Java source methods and 257 third-party library methods, which are divided into five groups based on the return data type. Table 1 displays three Java source method samples and three from third-party libraries.

Table 1. Examples of source methods

Method signature	**Category**
int java.io.DataInputStream.read(byte[])	I/O
java.lang.String java.net.URL.getQuery()	Network
java.sql.ResultSet java.sql.Statement.getResultSet()	Database
int org.apache.commons.io.input.ProxyInputStream.read(byte[])	I/O
org.apache.http.ssl.SSLContextBuilder org.apache.http.ssl.SSLContextBuilder.loadKeyMaterial()	Network
java.sql.ResultSet org.apache.derby.iapi.jdbc.BrokeredStatement.executeQuery(java.lang.String)	Database

Similarly, we created a dataset that included 350 sink methods from the same Java and 365 sink methods from the third-party libraries we investigated for the source method. Five examples of sink methods are displayed below in Table 2.

Table 2. Examples of sink methods

Method signature	**Category**
void java.util.logging.Logger.log(java.util.logging.LogRecord)	Log
void java.io.BufferedWriter.write(int)	I/O
void javax.servlet.http.HttpServletResponse.sendRedirect(java.lang.String)	Network
void com.sun.xml.txw2.output.XMLWriter.comment(char[],int,int)	I/O
java.net.HttpURLConnection org.jsoup.helper.HttpConnection(org.jsoup.Connection)	Network

[1] Based on JDK 8u201

[2] Jackson, Log4j2, Apache Commons, Guava, HttpClient, JMS, Joda Time, Apache MINA, Apache Commons and Derby

[3] Maven Repository: `https://mvnrepository.com/`

A global privacy data-flow is made up of many nodes that represent various methods. Different methods imply different types of data processing; to help demonstrate these processes, we characterise *process* under four categories.

Definition 7 (Process). *A process is a local data-flow F in a* privacy flow $G = F_1 \dots F_n$ *that is not a source flow F^o or a sink flow F^i.*

To specify some special kinds of processes, we use the following separate terms:

– Security process, if a process involves cryptography, database, security, or network packages.
– Authentication process, if authentication is involved.
– Initialisation process, if a process initialises a class.
– Non-privacy process, if it does not belong to either of the three categories above.

4 Assessing data privacy

It is challenging for software developers and legal privacy experts to have a mutual understanding and benefit from each other's expertise and insights. To address this, we examine how to leverage information from data flows in software to answer particular concerns related to GDPR rules. According to Article 4 in GDPR, *"the data controller determines the purposes for which and the means by which personal data is processed"*; hence, software providers (organisations) are data controllers if the organisation develops its own software. Otherwise, the software developers provide the implementation to the data controllers who are responsible for privacy protection. In this paragraph, we first look at the core GDPR obligations of the data controller, which serves as the duty of DPOs, and then discuss how we may help DPOs answer key DPIA questions (the document created by the approach in this study is referred to as a DPIA.).

4.1 Obligation of the Data Controller

Article 24 in the GDPR [9] states several obligations of the data controller which should be monitored by the DPO:

– by default and by design, the data controller should have a record of processing activities (Article 30);
– to ensure the security of the processing (Article 32);
– to notify personal data breaches to the supervisory authorities (Article 33);
– to communicate personal breaches to the data subject (article 34)
– to conduct DPIA (Article 35);
– to conduct prior consultation with supervisory authorities (Article 36).

The DPOs' role is to monitor whether the data controller fulfilled all of their commitments, which includes performing a DPIA when required. The writing of a DPIA is a shared duty for data controllers and DPOs.

As one of the major data protection authorities in Europe, the Irish Data Protection Commission [8] provides a short explanation of what DPIA contains:

> "*A DPIA describes a process designed to identify risks arising out of the processing of personal data and to minimise these risks as far and as early as possible.*"

Here we picked one of the most often used sample templates for generating a DPIA from the British Information Commissioner's Office (ICO) [20].

Under *Section 2: Describe the processing* of the template, there are three questions:

– Describe the nature of the processing: how will you collect, use, store and delete data? What is the source of the data? Will you be sharing data with anyone? You might find it useful to refer to a flow-graph or another way of describing data flows. What types of processing identified as likely high risk are involved?
– Describe the scope of the processing: what is the nature of the data, and does it include special category or criminal offence data? How much data will you be collecting and using? How often? How long will you keep it? and more
– Describe the context of the processing: what is the nature of your relationship with the individuals? How much control will they have?

Also under *Step 5: Identify and assess risks*, DPIA requires *"Describe the source of risk and nature of the potential impact on individuals."*

With a list of privacy data-flows listed under different categories, developers and DPOs could identify the parts of the program that collect privacy data from users and the relevant risky sinks. As a result of identifying privacy flows, they can pinpoint exposure risks and offer solutions to minimise those risks.

4.2 Answering Key DPIA Questions

Based on the previous paragraph, we now define six key questions relevant to the DPIA. Software development teams and DPOs should consider how to answer these questions when writing the DPIA. Each question is followed by an explanation of how our proposed analysis technique can help answer the questions.

Q1 *What is the source & nature of the data?*
A1 We need to know where the data is acquired originally and through which way. By having privacy source methods detected from the target program, we are able to look for all the potential locations in which personal data from users might get captured by the system. Different categories of privacy source methods might also indicate the type and nature of the data. For example, a method from `java.io.File` indicates this method reads from a file in the local file system.
Q2 *How is private data processed?*
A2 We want to identify the parts of the program that involve the processing of private data. This is a discovery study based on the flows that stem from privacy source methods. There are many patterns that might provide details on the processing of privacy data, for example, data travel through multiple sources or reach into multiple different sinks.
Q3 *Will the data be transformed? If so, how to ensure privacy data quality?*
A3 Data transformation and quality control can be subtle. There are clues such as the change of data types, certain types of data manipulation methods or certain APIs that might get involved in data transformation such as encryption or database packages.
Q4 *Will the data be shared/transferred and if yes, how?*
A4 Most of the data transportation happens when the privacy data flow into a sink method. By pinpointing the location and type of sink methods, we are able to identify whether there are private data being shared or transferred out of the target program.
Q5 *Does the data collected include special/highly sensitive personal data?*
A5 The property of privacy data need to be manually identified or with the help of developers. By adopting pure logic we can pick up properties that are directly linked with specific input devices of software.
Q6 *How is the data secured?*
A6 The security of private data is ensured when there are data protection mechanisms adopted, for example, the usage of cryptographic libraries or some encrypted databases. By locating the occurrence of these methods, we are able to analyse the data security protection of the target program.

5 Implementation

In the following paragraphs, we explain how our program analysis technique is implemented. Our implementation is built on Soot [16], a Java optimisation framework that provides four intermediate representations for analysing and transforming Java bytecode. Our technique consists of three parts:

– Transforming program bytecode to intermediate representation;
– Finding the source and sink methods;
– Building a privacy flow-graph by constructing one privacy flow for each source method at a time;
– Producing the abstraction extracted from the privacy flow-graph.

5.1 Finding Source and Sink Methods

Soot helps us transform our target program into a 3-address intermediate representation [23]. By traversing the $CFG(c.m)$ of each method $c.m$ in the program (provided in Jimple), the local data-flow analysis helps us detect the occurrences of source and sink methods in the pre-set annotation datasets (*om* and *im*) defined in Section 3.2. By having a complete list of source and sink methods in the application as $\mathcal{O}$ and $\mathcal{I}$, we now use them to start building the privacy flow-graph.

5.2 Building the Privacy Flow-Graph

For every class that includes a detected source method, we mark it as a class-of-interest (COI). For each COI, we first build a complete call-graph for it.

Definition 8 (Class-of-interest). *A Class-of-interest (COI) is a class that contains an invocation to one of the source methods ($\mathcal{O}$).*

$$c \in \mathcal{COI} \Leftrightarrow \exists o \in c, o \in \mathcal{O} \tag{1}$$

Now for each source method $o \in \mathcal{O}$, we build a global data-flow $G_o = F^o \dots F^n$ for it from the call-graphs of each class that G_o passes through. The final output is a union of all the global data-flows originating from source methods. This graph uses $A \rightarrow B$ to represent that method B invokes method A. Each G_o will be output as a separate `dot` file consisting of all the nodes (full signature of methods) and edges (invocations among the methods) which enables users to easily visualise it with simple tools.

5.3 Abstracting the Privacy Flow-Graph

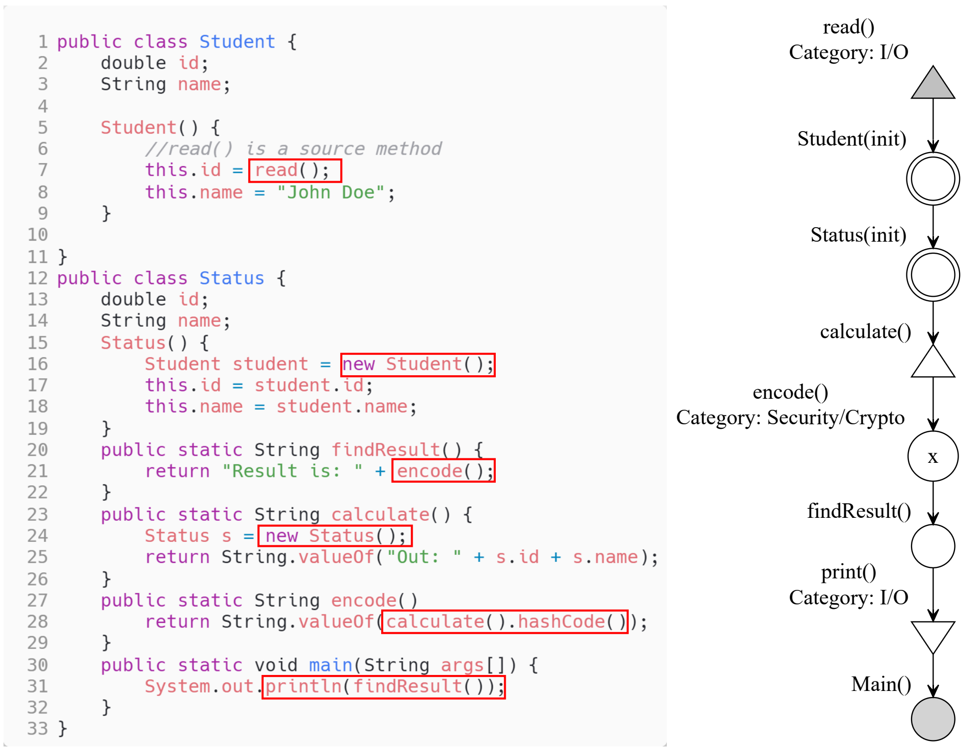
Privacy flows can be lengthy and comprise a variety of non-sensitive processes, many of which are from the same class and are unrelated to privacy protection yet may confound both developers and DPOs. We want to enable DPOs to get a big picture of the important processes without getting bogged down in minutiae by creating an abstraction from the privacy-flow-graphs generated by each source method. The abstraction is powered by a simple Python script running automatically on the initial complete privacy flow-graph. We select several key parts from the complete privacy flow-graph which are listed below as symbols:

– ▲: the starting source method;
– △: a non-starting source method;
– ○: a non-special process;
  Multiple processes that belong to the same package will be grouped into one process symbol in the abstraction.

- ⊗: a security process (cryptography, database, or network);
  A security process is detected by the substring detector, we look for substrings such as ‘encrypt’, ‘db’, ‘send’, ‘connect’ in the method and its package name.
- ▼: the end sink method;
- ▽: a non-ending sink method;
- ●: the end process;
- ◊: an authentication process;
  Similar to a security process, we report an authentication process when we detect the substring ‘auth’ in the method or its package name.
- ⊙: initialisation process(es).
  The initialisation process has ‘init’ in their names which can be picked up by our substring detector.

The above key information can be interpreted to help developers pin down specific issues in code and assist DPOs to have a sketch of high-level privacy patterns in the program, to also better answer the relevant questions in DPIA.

An example abstraction output reflecting the code snippet in Figure 5.3 is shown below: The



```
 1 public class Student {
 2     double id;
 3     String name;
 4
 5     Student() {
 6         //read() is a source method
 7         this.id = read();
 8         this.name = "John Doe";
 9     }
10
11 }
12 public class Status {
13     double id;
14     String name;
15     Status() {
16         Student student = new Student();
17         this.id = student.id;
18         this.name = student.name;
19     }
20     public static String findResult() {
21         return "Result is: " + encode();
22     }
23     public static String calculate() {
24         Status s = new Status();
25         return String.valueOf("Out: " + s.id + s.name);
26     }
27     public static String encode()
28         return String.valueOf(calculate().hashCode());
29     }
30     public static void main(String args[]) {
31         System.out.println(findResult());
32     }
33 }
```

Fig. 1. Example of a privacy data-flow generated for a source code fragment and its abstraction

example has one obvious source method `read()` (line 7) which acts as the starting point of our analysis. The technique then finds the next invocation to the source method when class `Student` gets initialised (line 16). This initialisation is triggered later by another initialisation of class `Status` (line 24). Following the newly created object `Status s`, we can trace the invocations to `calculate()` (line 28), `encode()` (line 21), `findResult()` (line 31) and finally to a sink `print()` (line 31) which is invoked by the `Main()` method. Source method `read()` and sink method `print()` have their categories labelled as well as the special process `encode()`.

Along with the abstraction figure, we provide short labels with the symbols which consist of information such as 1) categories of starting source method and sink methods; 2) categories of the special processes (security, authentication, or initialisation); 3) the class name is displayed when it is an initialisation process (optional).

6 Experiment

We are looking for apps that accept raw sensitive user data and entail data transmission, often in messaging and cloud storage applications. We thus selected the following two applications: Signal[4] and NextCloud[5]. The non-profit Signal Foundation and Signal Messenger LLC created Signal, a cross-platform end-to-end instant messaging service. We intend to study how Signal processes privacy-related user data by analysing both Signal's front-end Android application and the Signal Client Service API because of its expertise in end-to-end encryption. The purpose is to figure out how data is taken from the user and sent to the server. NextCloud is a client-server software package for developing and managing file hosting services. It is free and open-source software that anybody may install and run on their own private servers. We chose an implementation of its Client API that assists developers in developing Java apps with NextCloud integration since it is highly configurable. Similar to Signal, we intend to determine how the application handles privacy-sensitive user data.

6.1 Signal

The Signal Service API contains 17,710 lines of code, which might require developers and DPOs significant time and effort to comprehend. With our samples of DPIA answers, DPOs could effortlessly use our implementation results to create a DPIA.

A total number of 11 privacy flows were detected in Signal Service API (9 out of a total 11 are displayed here), the abstraction of its privacy flow-graph is shown below as Figure 2. We categorise the 9 source methods found into four 4 different functionalities. In Signal, we have discovered a similar pattern for all types of data communication: each raw entry is instantly sent into Signal's own cryptography libraries, allowing all user entries to be completely encrypted before they reach any possible sinks or processes. *Signal: Send Message* and *Signal: Receive Message* in Figure 2 demonstrate this end-to-end encryption mechanism. As indicated by the dashed green lines, there are some source methods that accept some values from local fields which originated from source methods in other flows. `PS01`, for example, gets value from source methods `O6` and `O9`, which are network-related properties associated with the message object.

Now, we answer the DPIA questions we listed in Section 4.2 using the flow that originates from `O1` (blue flow) in *Signal: Send Message* of Figure 2. To analyse privacy compliance, we combine the abstraction figure (which only comprises shapes and categories of critical processes) with detailed privacy flow-graphs (which contain every node in the flow-graph as well as their complete signature), shown as in Table 3.

Q1 *What is the source & nature of the data?*

[4] `https://signal.org/en/`
[5] `https://nextcloud.com/`

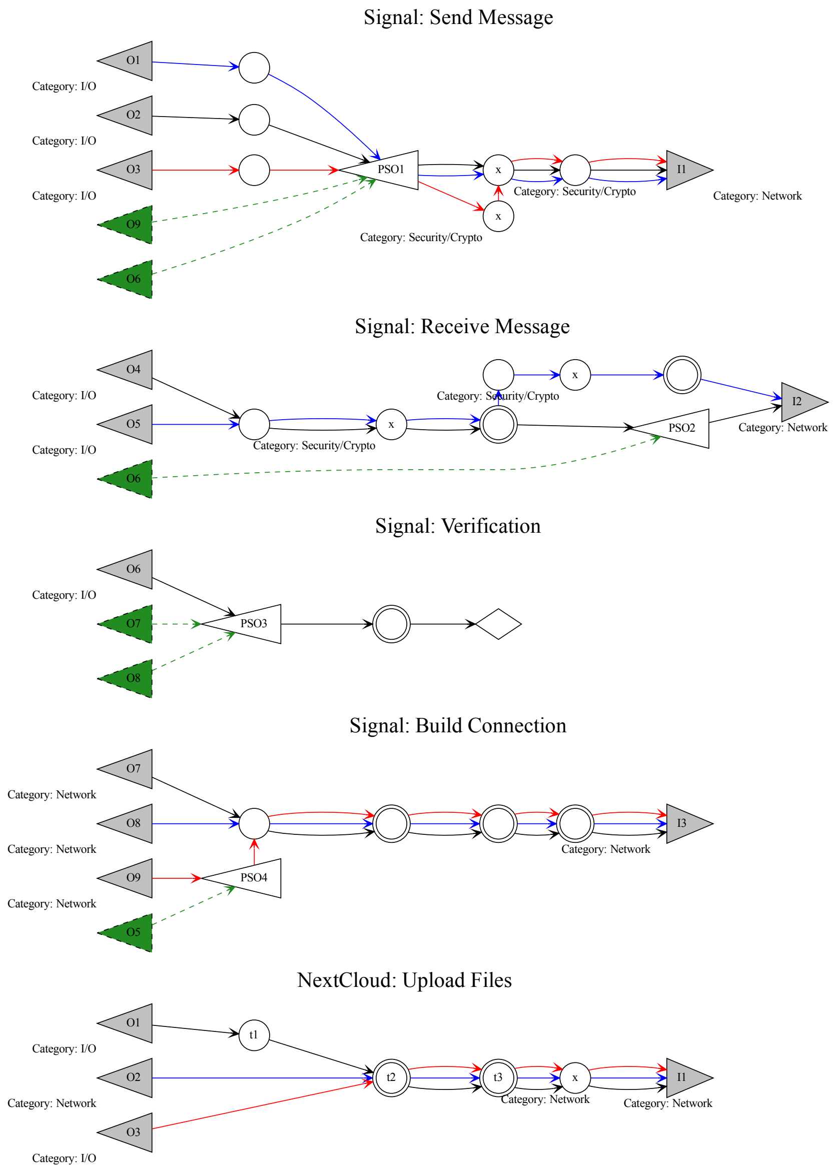


Fig. 2. Sample abstract privacy flows for Signal and NextCloud

Table 3. Complete privacy data-flow with abstraction symbols for sending a text message in Signal

Abstraction	Complete privacy data-flow
▲	android.widget.EditText getText()
○	org.thoughtcrime.securesms.jobs.PushTextSendJob deliver(message)
△	org.thoughtcrime.securesms.messages.MessageContentProcessor handleMessage(content, timestamp, ...)
⊗	org.whispersystems.signalservice.api.crypto.SignalServiceCipher encrypt(destination, message, ...)
○	org.whispersystems.signalservice.api.SignalServiceMessageSender getEncryptedMessage(content, recipient, timestamp, ...)
	org.whispersystems.signalservice.api.SignalServiceMessageSender getEncryptedMessages(content, recipient, timestamp, ...)
	org.whispersystems.signalservice.api.SignalServiceMessageSender createMessageContent(message)
▼	org.whispersystems.signalservice.api.SignalServiceMessageSender sendMessage(message, recipient, ...)

A1 Android applications take text input from a TE object which is a UI fragment providing a text field for users. The message field contains the raw message users want to send out.

Q2 *How is private data processed?*

A2 The abstraction tells us that there exist multiple processes when the text message is being sent out. There are two non-privacy processes from packages `org.signal.securesms.jobs` and `org.signalservice.api.signalservicemessagesender`. The package names indicate the types of processing behind the processes. There are also highly sensitive privacy processes such as the `MessageContentProcessor()` which is a non-starting source method that takes privacy data from a local field, in this case, it combines multiple privacy data including the text message. `org.signalservice.api.crypto` shows a typical encryption process, this also demonstrates the end-to-end encryption in Signal.

Q3 *Will the data be transformed? If so, how to ensure privacy data quality?*

A3 We notice that the data type gets immediately changed after being read into the device as raw strings. Both non-privacy and privacy processes transform data in order to achieve their functionality. However, encrypted messages stay encrypted before they get sent out, which ensures the content will not get manipulated by external parties.

Q4 *Will the data be shared/transferred and if yes, how?*

A4 The final ending sink method `sendMessage()` sends encrypted message objects out to the server from the client.

Q5 *Does the data collected include special/highly sensitive personal data?*

A5 The properties of the message object are sensitive. Not only the text message body itself, its attributes such as the details of senders but receivers and timestamps also remain sensitive during the entire process.

Q6 *How is the data secured?*

A6 Data security is guaranteed here by end-to-end encryption. All the privacy data related to the message get encrypted together as an `EncryptedMessage` object. This encrypted object cannot be decrypted by the server, which remains unreadable until it reaches the destination client.

Our discovery also supports what Signal claims in its privacy policy. By supplying the aforesaid information to both developers and DPOs, they are able to receive adequate information for creating DPIA and examining the privacy protection status in Signal without having to read the original code.

6.2 NextCloud

Since NextCloud recently implemented end-to-end encryption in their products, this feature only offers on the level of 'end-to-end encrypted folders'. Hence in our analysis, we only apply the technique to the client API which is applied to the traditional version that relies on TLS communication for safely transferring files.

From a total of 8,923 lines of code, we are able to extract key information from the NextCloud Client API using a simplified privacy flow-graph along with the complete flow-graphs with full signatures, as we did with Signal. We evaluate the DPIA questions to help DPOs in getting information from a legal standpoint, using the abstraction graph derived from our technique in Figure 2.

Q1 *What is the source & nature of the data?*
A1 NextCloud Client API allows a client to upload a new file via `uploadNewFile()`. The files can be of various types but shall be categorised as the user's personal data. There is also one network source, which links with data that can be used to identify users on the Internet.
Q2 *How is private data processed?*
A2 The file is transmitted from the device to the network; this is how a file is sent from the client to the server.
Q3 *Will the data be transformed? If so, how to ensure privacy data quality?*
A3 Not only the file acquired from the user is transferred to the server, but also network data and configuration settings. These various user data are processed and loaded into multiple fields of various class objects (reflect on the two initialisation processes). During these procedures, data types must be transformed in order to be organised for transmission as a type that the server accepts.
Q4 *Will the data be shared/transferred and if yes, how?*
A4 The final node is a network sink, which indicates that the user's data has been transmitted into the network and shared with the server.
Q5 *Does the data collected include special/highly sensitive personal data?*
A5 In this example, the data comprises user files, settings, and network details. User files are highly sensitive in terms of privacy.
Q6 *How is the data secured?*
A6 The network process here depicts a TLS connection, which is a cryptographic technology meant to ensure network communications security.

With the information provided above, we provide both developers and DPOs a better understanding of how the file upload process works in the NextCloud Client API, as well as what and where are the important aspects of privacy protection for NextCloud.

Privacy flow-graphs illustrate trends in terms of privacy-related data processing, including both benign and bad practices. It can assist not just DPOs and developers in responding to DPIA questions and addressing important processing, but also in identifying potentially questionable practices and ensuring good practices on privacy-related data.

7 Related Work

Using static analysis for security bug detection in software [6,4,10] is a source of inspiration for our work. In our work, we used hand-crafted datasets of source and sink methods for Java and popular third-party libraries as the start point for our analysis. The idea of using a pre-built set as a basis of static analysis is similar to SUSI [2], IccTA [17], MudFlow [3] and AndroidLeak [11] in terms of privacy protection for Android applications. Most current work, including the above, is specific to Android sinks and sources and often uses name features as the basis of their analysis, whereas we focus on Java in general without adopting heuristics. Regarding the GDPR, we demonstrate the utility of employing privacy flow-graphs to ease the DPIA process, which saves manual labour and assists in identifying possible sensitive processes that may be missed by human eyes.

Overall, there is an increasing interest in assuring privacy protection compliance prior to or throughout the software development lifecycle [22]. Privacy-by-design (PbD) has sparked research into methodologies and models for preserving software privacy before implementation begins, as well as forecasting or managing developer privacy compliance throughout implementation [12,1,14]. Many of these approaches may also be employed on a regular basis during the development cycle and while updating software. In the era of GDPR in Europe, there is also prior research [15,5,13] that aims to provide personalised solutions for DPIA in a variety of applications. According to a survey conducted by Dias Canedo *et al.* [7], technical staff frequently lack legal knowledge regarding privacy protection. Many existing works [18,19,21] propose models that limit on a conceptual level, that are not tangible for both technical and non-technical people to apply to implementation, motivating us to propose an automatic technique to analyse privacy compliance in software.

8 Conclusion

In terms of privacy protection, there always exists a barrier between developers and DPOs. DPOs need to generate a successful DPIA to document the privacy protection behaviour of software, this requires the developer's comprehensive knowledge of code details. Our work provides a technique for detecting privacy source and sink methods in software bytecode, generating privacy flow-graphs from the discovered sources, and supporting DPOs in writing a DPIA utilising privacy flow-graphs and associated abstractions.

9 Limitation and future work

Our present method requires predetermined source and sink lists. Given that modern applications typically contain hundreds of direct and indirect dependencies, we may miss a significant number of privacy-related sources and sinks. Therefore, we rely on the knowledge of technical specialists to create a more precise list of sources and sinks. Moreover, despite the fact that our complete privacy flow-graphs and their abstractions can express key privacy-sensitive behaviours such as data acquisition, encryption, and transportation, they are unable to provide complete information regarding which type of data manipulation was involved in terms of privacy protection; therefore, developers may be required to provide additional explanation for DPOs.

Future work includes a more detailed local flow analysis for each local data-flow in a privacy global data-flow, such as tracking how values from privacy-related data are modified in the local method and flagging sensitive manipulations such as value accumulation and separation. In the meantime, it is feasible to extract information from the manifest file on which third-party libraries are imported by the software in order to assist in the construction of a more adaptable list of sources and sinks. This procedure might be automated by including these third-party libraries (which are usually downloadable as JAR files) as a part of the input of the analysis. Additionally, since dynamically-typed languages such as JavaScript are used in many different types of modern systems, it would be advantageous to build a source code-based analyser based on tools such as Semgrep [6], which as a starting point for extending our results to web applications.

Acknowledgement

We appreciate the legal insight that Jan Czarnocki and Lydia Belkadi have given. This work is part of the Privacy Matters (PriMa) project. The PriMa project has received funding from European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 860315.

References

1. Antignac, T., Métayer, D.L.: Privacy by design: From technologies to architectures. In: Annual privacy forum. pp. 1–17. Springer, Berlin, Heidelberg (2014)
2. Arzt, S., Rasthofer, S., Bodden, E.: Susi: A tool for the fully automated classification and categorization of android sources and sinks (2013)

[6] `https://semgrep.dev/`

3. Avdiienko, V., Kuznetsov, K., Gorla, A., Zeller, A., Arzt, S., Rasthofer, S., Bodden, E.: Mining apps for abnormal usage of sensitive data. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. vol. 1, pp. 426–436. IEEE, Italy (2015). https://doi.org/10.1109/ICSE.2015.61
4. Ayewah, N., Pugh, W., Hovemeyer, D., Morgenthaler, J.D., Penix, J.: Using static analysis to find bugs. IEEE software **25**(5), 22–29 (2008)
5. Bu-Pasha, S.: The controller's role in determining 'high risk' and data protection impact assessment (dpia) in developing digital smart city. Information & Communications Technology Law **29**(3), 391–402 (2020)
6. Chess, B., McGraw, G.: Static analysis for security. IEEE security & privacy **2**(6), 76–79 (2004)
7. Dias Canedo, E., Toffano Seidel Calazans, A., Toffano Seidel Masson, E., Teixeira Costa, P.H., Lima, F.: Perceptions of ict practitioners regarding software privacy. Entropy **22**(4), 429 (2020)
8. (DPC), D.P.C.: Data protection impact assessments (jul 2022), `https://www.dataprotection.ie/en/organisations/know-your-obligations/data-protection-impact-assessments`
9. European Commission: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance) (2016), `https://eur-lex.europa.eu/eli/reg/2016/679/oj`
10. Evans, D., Larochelle, D.: Improving security using extensible lightweight static analysis. IEEE software **19**(1), 42–51 (2002)
11. Gibler, C., Crussell, J., Erickson, J., Chen, H.: Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale. In: International Conference on Trust and Trustworthy Computing. pp. 291–307. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
12. Hadar, I., Hasson, T., Ayalon, O., Toch, E., Birnhack, M., Sherman, S., Balissa, A.: Privacy by designers: software developers' privacy mindset. Empirical Software Engineering **23**(1), 259–289 (2018)
13. Henriksen-Bulmer, J., Faily, S., Jeary, S.: Dpia in context: Applying dpia to assess privacy risks of cyber physical systems. Future Internet **12**(5), 93 (2020)
14. Hoepman, J.H.: Privacy design strategies. In: Cuppens-Boulahia, N., Cuppens, F., Jajodia, S., Abou El Kalam, A., Sans, T. (eds.) ICT Systems Security and Privacy Protection. pp. 446–459. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
15. Horák, M., Stupka, V., Husák, M.: Gdpr compliance in cybersecurity software: A case study of dpia in information sharing platform. In: Proceedings of the 14th International Conference on Availability, Reliability and Security. ARES '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3339252.3340516, `https://doi.org/10.1145/3339252.3340516`
16. Lam, P., Bodden, E., Lhoták, O., Hendren, L.: The soot framework for java program analysis: a retrospective. In: Cetus Users and Compiler Infastructure Workshop (CETUS 2011). vol. 15. IEEE, Purdue University (2011)
17. Li, L., Bartel, A., Bissyandé, T.F., Klein, J., Le Traon, Y., Arzt, S., Rasthofer, S., Bodden, E., Octeau, D., McDaniel, P.: Iccta: Detecting inter-component privacy leaks in android apps. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. vol. 1, pp. 280–291. IEEE, Italy (2015). https://doi.org/10.1109/ICSE.2015.48
18. Martin, Y.S., Kung, A.: Methods and tools for gdpr compliance through privacy and data protection engineering. In: 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 108–111. IEEE, London (2018). https://doi.org/10.1109/EuroSPW.2018.00021
19. Massey, A.K., Otto, P.N., Hayward, L.J., Antón, A.I.: Evaluating existing security and privacy requirements for legal compliance. Requirements engineering **15**(1), 119–137 (2010)
20. Office, I.C.: Data protection impact assessments (dpias). `https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/data-protection-impact-assessments-dpias/` (02 2018), (Accessed on 03/02/2022)
21. Piras, L., Al-Obeidallah, M.G., Praitano, A., Tsohou, A., Mouratidis, H., Gallego-Nicasio Crespo, B., Bernard, J.B., Fiorani, M., Magkos, E., Sanz, A.C., et al.: Defend architecture: a privacy by design platform for gdpr compliance. In: International Conference on Trust and Privacy in Digital Business. pp. 78–93. Springer, Springer, Bratislava, Slovakia (2019)
22. Rubinstein, I.S.: Regulating privacy by design. Berkeley Tech. LJ **26**, 1409 (2011)
23. Vallée-Rai, R., Hendren, L.J.: Jimple: Simplifying java bytecode for analyses and transformations (1998)

PAPER 2

PABAU: Privacy Analysis of Biometric API Usage

Tang, F. (2022)., PABAU: Privacy Analysis of Biometric API Usage, In Proceeding of the 2022 IEEE Privacy Computing (PriComp 2022), Haikou, China, 2022, pp. 2301-2308.

PABAU: Privacy Analysis of Biometric API Usage[⋆]

Feiyang Tang

Norwegian Computing Center
N-0314 Oslo, Norway
feiyang@nr.no

Abstract. Biometric data privacy is becoming a major concern for many organizations in the age of big data, particularly in the ICT sector, because it may be easily exploited in apps. Most apps utilize biometrics by accessing common application programming interfaces (APIs); hence, we aim to categorize their usage. The categorization based on behavior may be closely correlated with the sensitive processing of a user's biometric data, hence highlighting crucial biometric data privacy assessment concerns. We propose PABAU, Privacy Analysis of Biometric API Usage. PABAU learns semantic features of methods in biometric APIs and uses them to detect and categorize the usage of biometric API implementation in the software according to their privacy-related behaviors. This technique bridges the communication and background knowledge gap between technical and non-technical individuals in organizations by providing an automated method for both parties to acquire a rapid understanding of the essential behaviors of biometric API in apps, as well as future support to data protection officers (DPO) with legal documentation, such as conducting a Data Protection Impact Assessment (DPIA).

Keywords: Data protection · privacy · GDPR · biometric privacy · program analysis.

1 Introduction

Authentication solutions are shifting from password-based to passwordless solutions due to the documented risks of passwords because of the user-generated credentials, brute-force attacks, recycled passwords, and large-scale breaches. Taking advantage of biometrics, biometric authentication solutions are deployed for both professional usage, for example, company intranet authentication, and personal usage, for example, bank app authentication. The usage diversification of passwordless authentication solutions, based on biometrics, raises privacy concerns since biometric data is acquired and processed within a connected multi-program environment, such as a mobile device or a laptop instead of an offline single-program environment, such as a USB stick [1] that uses biometric as the authentication resource.

With the more common usage of biometric authentication, manipulating biometric data becomes controversial and makes those solutions run the risk of exposing this sensitive data. It is critical to guarantee excellent practice and effective solution implementation throughout the software development process. With more and more privacy-focused legal provisions being implemented, such as the General Data Protection Regulation (GDPR) in Europe and the California Consumer Privacy Act (CCPA), many companies are becoming aware of the importance of privacy in software development. They are beginning to review their code for potential privacy issues in order to avoid the potential legal risks associated with these newly implemented provisions. GDPR classifies biometric data as "sensitive" [6], and privacy protection on biometric data has become one of the most discussed topics in privacy-related research [9, 16]. Biometric processing is now a significant and vital aspect of privacy protection. Given that more software employs biometric

[⋆] Published at *2022 IEEE Conference of Privacy Computing* (PriComp 2022), DOI: 10.1109/SmartWorld-UIC-ATC-ScalCom-DigitalTwin-PriComp-Metaverse56740.2022.00327

[1] https://fidoalliance.org/specifications/

APIs to authenticate users for convenience, it is critical to understand how biometric APIs function and how biometric data is processed. As a result of their nature and sensitive use, biometric APIs must be used with a clear understanding by developers and data protection officers (DPOs) need relevant information from developers. A rigorous privacy assessment of the biometric authenticators is crucial to mitigate privacy risks. However, such an assessment is difficult to perform due to the subtle nature of privacy and the complex structure of programs implementing biometric authenticators.

In this paper, we would like to understand how biometric-relevant APIs are implemented in real-world applications. We propose PABAU, a technique to classify the usage of biometric APIs in the applications. Various usages (those are, methods from the biometric APIs) are classified into different labels according to their behavior, for example, biometric acquisition, user interaction, data transfer, data erasure, etc. We can use the results of the analysis to assist developers in better understanding current solutions that adopt biometric APIs in an action-based analysis and guide them to the section that requires more attention and provide a broad overview of how biometric data is handled and manipulated in the application to either project managers or data protection officers without examining the actual implementation code.

Our contributions are:

- An method for automatically categorizing biometric API methods (Section 4.1 and Section 5).
- A handcrafted ground-truth dataset built from the popular Android and FIDO2 biometric authentication APIs (Section 4.4).
- An automatic scheme to describe the privacy behavior of API usage in client code, for instance, authentication, cryptography, termination, and permission, that could aid developers and DPOs in analyzing biometric-related privacy compliance (Section 4.3).

We demonstrate the utility of our research by testing our classifier on eight popular Android applications (Section 6).

2 Related work

Our study focuses on utilizing static analysis to discover software vulnerabilities and supporting non-technical experts with software privacy compliance checks. This section begins with a discussion of traditional software security vulnerability discovery using program analysis, followed by a discussion of software privacy assessment research.

Taint analysis is a practical approach to information flow analysis that is frequently used by researchers to analyze the transmission of private data. It comprises both dynamic and static taint analysis techniques. Using taint analysis to find privacy vulnerabilities in Android applications has been discussed in multiple research.

By studying data relationships between program variables without running the program, static taint analysis may determine if data can propagate from a taint source to a taint aggregation point [22]. Many Java and Android-based research are based on the tool Soot. Soot [21]^2 is a Java bytecode analysis tool developed by the Sable research group of McGill University in 1996. It provides a variety of bytecode analysis and transformation functions, through which it can perform intra- and inter-process analysis and optimization, and program flow analysis. FlowDroid [2] mimics a component's lifecycle by building virtual main functions to find sensitive data transfer channels for privacy leaks with an 86% accuracy. MudFlow [3] used the static taint analysis tool FlowDroid to test 2,866 benign software and produced a total of 338,610 taint analysis results. The tool can only give out whether a piece of software as a whole has malicious behavior (and the tool will also have false positives) and cannot tell whether a taint analysis path (source to sink) is the result of direct privacy leakage. Moving from Android to general Java applications, in 2018, Sas et al.

^{2} `https://github.com/soot-oss/soot`

proposed SSCM [19] to identify data sources and sinks from arbitrary Java libraries. SSCM uses Soot Framework to extract information from the Java Bytecode and uses its novel rules and WEKA Data Mining Framework [11] to classify different types of security sources and sinks. SWAN [18][3] is similar to SSCM. Both SSCM and SWAN use similar features for the classification model's learning process. SSCM proposed 9 when SWAN contains 25 basic features. These features focus mostly on the name of *Resource Class* and *Resource Method* and *call to the corresponding methods*. Meanwhile, parameters and some specific patterns are also important. Almost all the features used in SSCM have been included in SWAN, and SWAN contains a few more features on return types and some specific patterns in methods. Meanwhile, parameters and some specific patterns are also important.

Privacy-by-design (PbD) has inspired a significant amount of research that provides approaches and models to preserve software privacy before implementation begins, as well as to forecast or manage developer privacy compliance throughout implementation [10,14]. According to a survey [7] written by Dias Canedo et al., technical employees frequently lack legal expertise in terms of privacy protection, which encouraged us to design a new easy-to-use technique for developers to check the privacy compliance of their implementation on a regular basis without diving into hundreds of lines of code.

With requirements like the data protection impact assessment (DPIA), researchers have been attempting to make it easier for non-technical professionals, such as attorneys, to confirm GDPR compliance. To aid the examination of privacy consequences in ubiquitous computing systems, Fernández et al. [8] suggested a software-assisted process methodology. Similarly, the BPR4GDPR [17] initiative has the same objective, while Zibuschka [24] analyzed the automation potential of the DPIA process. Using privacy flow-graphs, Tang et al. [20] presented an approach for analyzing privacy in the context of GDPR. This motivates us to bridge the gap between the technical and non-technical experts with an approach that might assist both parties in terms of assessing privacy in biometric API usage.

3 Motivation

Conducting a privacy analysis necessitates a vast number of software information, many of which might be highly specific. Privacy lawyers may request that developers offer precise answers with particular processing in software, which could be challenging when the software is not built and maintained by a single team from the start. Furthermore, manually reviewing hundreds of lines of code demands a significant amount of effort on the part of the development team. It is difficult for developers and lawyers to have a mutual understanding and benefit from each other's expertise and analyzes [4, 10, 12]. Our goal is to see if we can transpose GDPR principles and requirements with actual code locations or patterns in software. In this section, we look at fundamental GDPR obligations from the standpoint of a developer and then consider how we may change our solution to serve both parties in terms of privacy protection in biometric API usage.

3.1 Legal perspectives

GDPR states several obligations from the data controller which should be monitored by the DPO [5]:

– by default and by design, to have a record of processing activities (Article 30);
– to ensure the security of the processing (Article 32);
– to notify personal data breaches to the supervisory authorities (Article 33);
– to communicate personal breaches to the data subject (Article 34);
– to conduct DPIA (Article 35);

[3] `https://github.com/secure-software-engineering/swan/tree/master/swan_core`

– to conduct prior consultation with supervisory authorities (Article 36).

The DPO's mission is to monitor whether the data controller fulfilled all of his or her commitments, which includes performing a high-quality DPIA when required. This makes the assignment of DPIA creation a duty for both data controllers and DPOs. DPIA is a technique that assists developers and organizations in systematically analyzing, identifying, and mitigating data protection risks in a project or plan. The deliverable document created by the approach in this study is referred to as a DPIA.

We intend to offer tailored solutions for DPIA under diverse applications, similar to earlier research [13, 15]. The aim of our technique is to provide information that can directly benefit developers and DPOs by providing comprehensive guidance on producing a quality DPIA. Developers and DPOs may identify the components of the software that process biometric data from users within applicable legal viewpoints with a list of biometric API use described under different kinds of behavioral labels. Because various API methods can capture different forms of biometrics processing, we can adopt this into our study.

4 Approach

To tackle the barrier between developers and DPOs in terms of biometric privacy protection compliance checks, we utilize SWAN as a foundation to create a technique PABAU that can automatically categorize methods in biometric APIs based on their actions.

4.1 General architecture

The architecture of our technique is illustrated in Fig. 1. The compiled class files from Java or Android applications are our source of analysis. We can learn the features of these APIs and use them to categorize their usage in real-world applications by training a model based on the most common biometric API methods, such as the Android official biometric API and FIDO2 implementations. The biometric types included in this study cover the two most common biometrics: face and fingerprint.

PABAU operates on different classification sets, one for each kind of biometric API method. Most action-based sets are not disjoint (for example, `Authenticate()` may be labeled as both **CRYPTO** (cryptography involved) and **AUTHENTICATE**), but certain sets like the biometric strength levels (for example, **BSC1** and **BSC2**) are disjoint (that is, each method can only get labeled as one of them). The classification for each label operates independently.

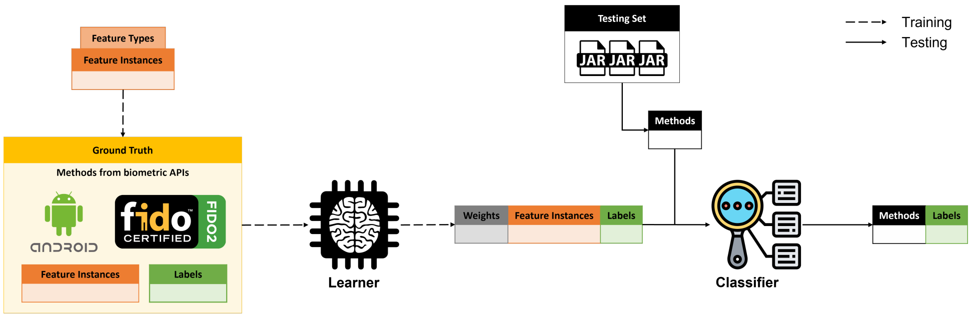


Fig. 1. General architecture of PABAU

4.2 Feature types and feature instances

We adopted the same system as SWAN with feature types and corresponding feature instances. The features are extracted from the semantics of methods in the program. Feature types are general features such as `methodNameContains` and feature instances represent concrete instances of feature types such as `methodNameContainsUser`. We considered the following criteria when constructing feature types.

- Names of the methods and the declaring classes:
  - Start with a keyword;
  - Contain a keyword;
  - End with a keyword.
- Return types of the methods:
  - Primitive types (for example, `int`, `string`);
  - Other objects (for example, `PromptInfo`).
- Parameters:
  - Number of parameters;
  - Types of parameters: primitive or other object types.
- Invocations: whether the current method calls any other method:
  - The name of callee;
  - The return type of callee;
  - The parameter of the callee.
- Particular pattern derived from above:
  - Parameter flows into the return statement;
  - Parameter flows into a local field;
  - A local field flows into the return statement.

4.3 Labels

Some particular actions of biometric APIs might lead to potential privacy risks, which motivates us to create labels representing different actions. The following labels in Tab. 1 are what we used to classify different behaviors in biometric API usage.

4.4 Ground-truth

In order to train PABAU, we need to build a ground-truth dataset specific to biometric authentication and annotate the methods. Because Android apps are the most popular domain for biometric API usage, we chose official Android Biometric APIs as our domain of study. To also cover web applications that use FIDO2, we also find a representative implementation to enrich the ground truth.

Initially, we parse the existing methods from Android biometrics APIs[4] and the FIDO2 server implementation from LINE[5]. To enrich the training set, we also manually annotated over 150 methods from several popular sample implementations of biometric APIs[6][7][8]. We manually decompress the library files, which were originally in JAR format, and label the methods in the class files in accordance with their corresponding behaviors. A general training set should cover all of the common biometric API methods, which is why we include the native Android biometric API methods and some third-party ones. Each of these methods is annotated with privacy behavior labels. Then, in a structured manner, for each method we collect the following properties:

[4] `https://android.googlesource.com/platform/frameworks`
[5] `https://github.com/line/line-fido2-server`
[6] Soter: `https://github.com/Tencent/soter/`
[7] Android-Goldfinger : `https://github.com/infinum/Android-Goldfinger`
[8] `https://github.com/sergeykomlach/AdvancedBiometricPromptCompat`

Table 1. Labels and the description

Label	**Description**
BSC1	Biometric strength level 'convenience'. This level does not use cryptography nor the `BiometricPrompt` API.
BSC2	Biometric strength level 'weak'. This level uses only the `BiometricPrompt` API but no cryptography.
BSC3	Biometric strength level 'strong'. This level uses both the`BiometricPrompt` API and cryptography.
SOURCE	Where the potentially sensitive data come from.
SINK	The places where the sensitive data might end up in.
CHECKER	Checking occurs, for example, check for hardware prerequisites.
PERMISSION	Getting or verifying the user's permission for the biometric process.
AUTHENTICATE	Authentication-related process.
CRYPTO	Where encryption is involved in the process.
TERMINATION	Where termination of specific service(s) happens.
INTERACTION	The system is making interactions with users, for example, giving users a few options.
TRANSFER	Biometric-related data or decisions derived from it are being transferred.
ACQUISITION	The acquisition of biometric-related data.
DELETION	The deletion of biometric-related data.
STORAGE	Where biometric-related data is being stored and kept.
DATABASE	Where a database is involved.

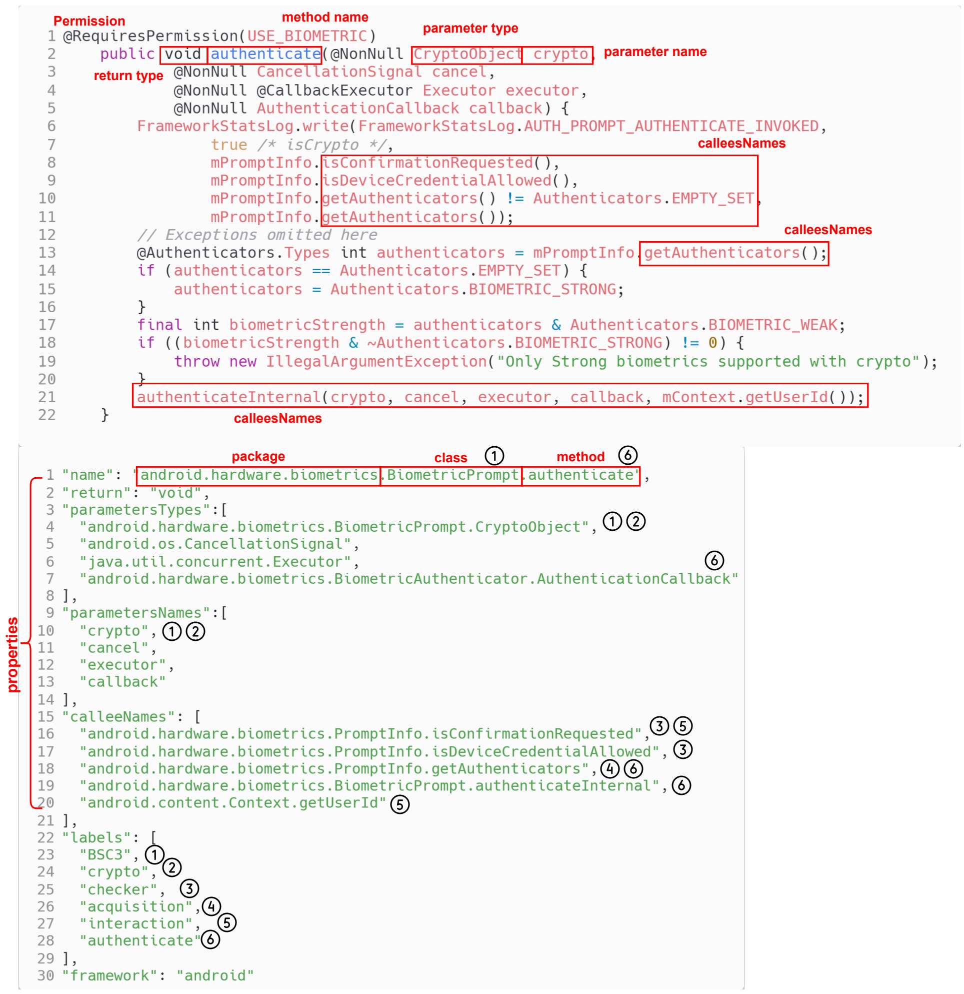
- **name.** The method's fully qualified name (for example, `package.class.method`). This indicates which class the method belongs to.
- **return.** The method's return type (`void`, primitive types (for example, `int`), or objects of other classes). This helps determine whether there is biometric data flow out from a local method.
- **parametersTypes.** The method's parameters types (empty, primitive types, for example, `int`), or reference types).
- **calleeNames.** The full name of the methods invoked inside that method. This helps determine relationships between methods, to locate data flows of biometric data better.

Each method in this dataset is manually associated with the label(s), the detailed explanation of labels is discussed in Section 4.3. The example of how the training process is illustrated in Fig. 2 along with an annotation example in Fig. 3. Details on which type of feature and feature instance are summarized in Section 4.2.

5 Training and classification

Similarly to SWAN [18]'s technique, we develop a collection of binary features that assess certain qualities of the methods to assist the machine learning algorithm in labeling the methods based on their actions.

We evaluated six popular classifiers from WEKA [11], a commonly used machine learning API written in Java: Bayes Net, Naive Bayes, Logistic Regression, C4.5, and Decision Stump and SVM. For each classifier in the training set, ten 10-fold cross-validations were conducted to find the best classifier for PABAU. The median precision and recall statistics for each classifier are presented in



```
@RequiresPermission(USE_BIOMETRIC)
    public void authenticate(@NonNull CryptoObject crypto,
            @NonNull CancellationSignal cancel,
            @NonNull @CallbackExecutor Executor executor,
            @NonNull AuthenticationCallback callback) {
        FrameworkStatsLog.write(FrameworkStatsLog.AUTH_PROMPT_AUTHENTICATE_INVOKED,
                true /* isCrypto */,
                mPromptInfo.isConfirmationRequested(),
                mPromptInfo.isDeviceCredentialAllowed(),
                mPromptInfo.getAuthenticators() != Authenticators.EMPTY_SET,
                mPromptInfo.getAuthenticators());
        // Exceptions omitted here
        @Authenticators.Types int authenticators = mPromptInfo.getAuthenticators();
        if (authenticators == Authenticators.EMPTY_SET) {
            authenticators = Authenticators.BIOMETRIC_STRONG;
        }
        final int biometricStrength = authenticators & Authenticators.BIOMETRIC_WEAK;
        if ((biometricStrength & ~Authenticators.BIOMETRIC_STRONG) != 0) {
            throw new IllegalArgumentException("Only Strong biometrics supported with crypto");
        }
        authenticateInternal(crypto, cancel, executor, callback, mContext.getUserId());
    }
```

```
1  "name": "android.hardware.biometrics.BiometricPrompt.authenticate",
2  "return": "void",
3  "parametersTypes":[
4    "android.hardware.biometrics.BiometricPrompt.CryptoObject", ① ②
5    "android.os.CancellationSignal",
6    "java.util.concurrent.Executor",                                      ⑥
7    "android.hardware.biometrics.BiometricAuthenticator.AuthenticationCallback"
8  ],
9  "parametersNames":[
10   "crypto", ① ②
11   "cancel",
12   "executor",
13   "callback"
14 ],
15 "calleeNames": [
16   "android.hardware.biometrics.PromptInfo.isConfirmationRequested", ③ ⑤
17   "android.hardware.biometrics.PromptInfo.isDeviceCredentialAllowed", ③
18   "android.hardware.biometrics.PromptInfo.getAuthenticators", ④ ⑥
19   "android.hardware.biometrics.BiometricPrompt.authenticateInternal", ⑥
20   "android.content.Context.getUserId" ⑤
21 ],
22 "labels": [
23   "BSC3", ①
24   "crypto", ②
25   "checker", ③
26   "acquisition", ④
27   "interaction", ⑤
28   "authenticate" ⑥
29 ],
30 "framework": "android"
```

Fig. 2. Example of a method from the Android Biometric API: `authenticate` is a method from the class `android.hardware.biometrics.BiometricPrompt` and the corresponding annotated method (in the green text) data point. The numbered circles (from lines 23 to 28) correspond to the labels (Section 4.3). They were assigned to this data point since `authenticate` has the properties in lines 1, 4, 7, 10, 16-20 in the text.

Table. 2. In terms of average accuracy and recall, we can observe that SVM performed the best with an average precision of 0.9725 and recall of 0.98. Except for Stump, which performed far less well than the other five, the majority of classifiers functioned well.

For example, the feature instance `hasCalleeNameStartsCheck` is likely to indicate that the current method calls a checker function to see whether specific conditions have been satisfied. When the learning process starts, PABAU computes a yes or no response for each feature instance for each method in the training set. Based on the feedback, we are able to learn which combination

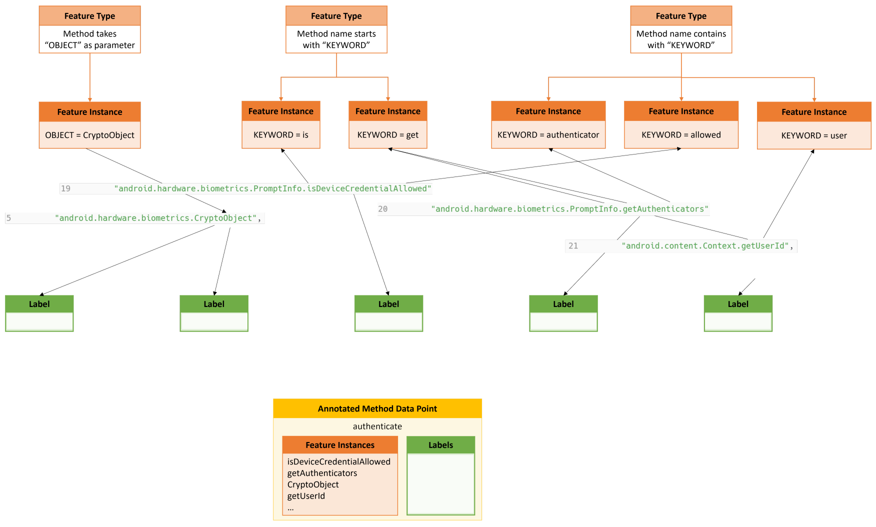

Feature Type
Method takes "OBJECT" as parameter
Feature Type
Method name starts with "KEYWORD"
Feature Type
Method name contains with "KEYWORD"
Feature Instance
OBJECT = CryptoObject
Feature Instance
KEYWORD = is
Feature Instance
KEYWORD = get
Feature Instance
KEYWORD = authenticator
Feature Instance
KEYWORD = allowed
Feature Instance
KEYWORD = user
"android.hardware.biometrics.PromptInfo.isDeviceCredentialAllowed"
"android.hardware.biometrics.CryptoObject",
"android.hardware.biometrics.PromptInfo.getAuthenticators"
"android.content.Context.getUserId",
Label
Label
Label
Label
Label
Annotated Method Data Point
authenticate
Feature Instances
isDeviceCredentialAllowed
getAuthenticators
CryptoObject
getUserId
...
Labels

Fig. 3. Annotation diagram of the sample method

of features is the most associated with the labels. These combinations will be then used to label methods in testing sets.

Table 2. Precision (P) and recall (R) of the 10-cross fold validation for all classifiers averaged over 10 iterations

	Source		Sink		Auth		Crypto		Average	
	P	R	P	R	P	R	P	R	P	R
BayesNet	0.92	0.97	0.95	0.95	1.00	1.00	1.00	1.00	0.9675	0.9800
NaiveBayes	0.93	0.95	0.94	0.96	0.88	1.00	1.00	1.00	0.9375	0.9775
Logistic Reg	0.94	0.95	0.95	0.92	0.88	1.00	1.00	1.00	0.9425	0.9675
C4.5	0.93	0.96	0.93	0.95	0.88	1.00	1.00	0.75	0.9350	0.9775
Stump	0.79	0.84	0.88	0.92	0.75	1.00	1.00	0.5	0.8550	0.8150
SVM	0.94	0.96	0.95	0.96	1.00	1.00	1.00	1.00	**0.9725**	**0.9800**

6 Experiment

The main research question here is how our technique performs in terms of assigning biometric API methods with the corresponding behavioral labels. We aim for high precision because calculating recall is unrealistically tricky. All our experiments were performed on a Windows machine with an Intel i7 1.90GHz CPU and 16 GB memory.

6.1 Description of the datasets

Besides the ground-truth dataset we built using Android biometric APIs and FIDO2 server implementation, we also manually annotated biometric API usages from real-world Android mobile applications. Given that biometric authentication is only used in a subset of Android applications, we chose eight popular apps from the Google Play Store[9] in four categories: banking & finance (Klarna, Sparebank1, Revolut, Paypal), private storage (Private Photo Vault, NordLocker), privacy notebook (Notability), and password management (1Password).

The description of the main functionality and biometric usage of the apps are listed in Table 3.

We split the entire handcrafted dataset into two parts, 70% of the data points were used to train the classifier and the rest 30% were used for testing.

Table 3. Description of the datasets

App name	Description
Klarna	A financial app that provides online payment and direct payments along with post-purchase payments. Users can use biometrics to unlock the app and authorise certain payments.
Sparebank1	A online banking app for Sparebank1. Users can use biometrics to unlock the app, authorize certain payments and approve e-invoice.
Revolut	An online banking app for Revolut Bank. Users can use biometrics to unlock the app, add a new card, authorize payments, and update profiles.
Paypal	A financial app that provides the online payment between users or merchants. Users can use biometrics to unlock the app, authorize payments, and update card/account details.
Private Photo Vault	A photo storage app that provides access control. Users can use biometrics to unlock the app, add photos to the vault and add extra control to certain folders.
NordLocker	A storage app that provides access control. Users can use biometrics to unlock the app, add files into the locker and apply extra encryption to certain files.
Notability	An online notebook. Users can use biometrics to lock and unlock certain notes.
1Password	A password manager. Users can use biometrics to unlock the app, add/update credentials, and sync information with the cloud.

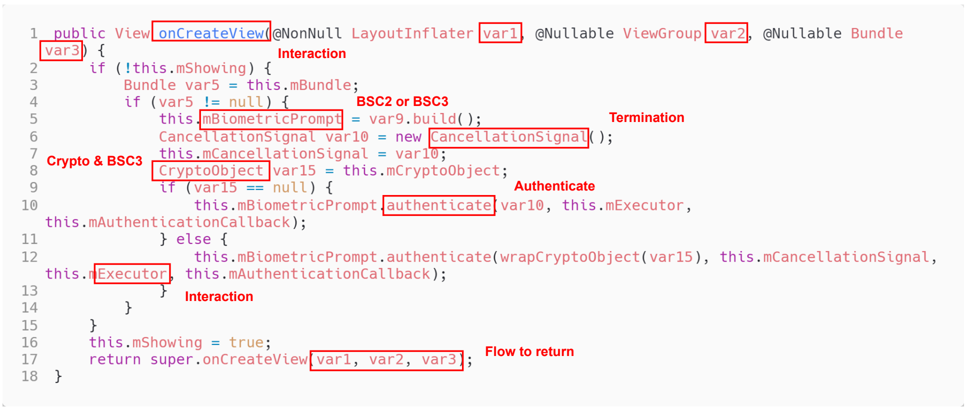
6.2 Example

Fig. 4 gives an example of what kind of result PABAU generates by analyzing the JAR files. The method `onCreateView` was extracted from the class `androidx.biometrics.BiometricFragment`.

We could see that the method itself gets classified by multiple labels, for example, it gets classified as **BSC3** and **Crypto** from the invocation to the data type `CryptoObject`. The app's implementation has a fairly small number of biometric API methods, however, this does not diminish their importance. Actually, their significance might be underestimated, and the underlying correlations between one method and other subtle behaviors reveal a sensitivity to privacy. This

[9] `https://play.google.com/store/apps/`

is also why we need a technique such as PABAU to classify each method rigorously with as many privacy behavior labels as feasible.



```
public View onCreateView(@NonNull LayoutInflater var1, @Nullable ViewGroup var2, @Nullable Bundle var3) {
    if (!this.mShowing) {
        Bundle var5 = this.mBundle;
        if (var5 != null) {
            this.mBiometricPrompt = var9.build();
            CancellationSignal var10 = new CancellationSignal();
            this.mCancellationSignal = var10;
            CryptoObject var15 = this.mCryptoObject;
            if (var15 == null) {
                this.mBiometricPrompt.authenticate(var10, this.mExecutor, this.mAuthenticationCallback);
            } else {
                this.mBiometricPrompt.authenticate(wrapCryptoObject(var15), this.mCancellationSignal, this.mExecutor, this.mAuthenticationCallback);
            }
        }
    }
    this.mShowing = true;
    return super.onCreateView(var1, var2, var3);
}
```

Fig. 4. Sample classification result snippet from Sparebank1

6.3 Runtime and memory performance

Table 4 demonstrates the stability of PABAU, and it does not require huge computational costs to analyze common mobile apps. The cost of time and memory increases as the size of a program grows, which also reflects a greater number of methods inside the application. We also observed that the quantity of biometric-related approaches influences the time and memory requirements. Even though Notability is almost three times the size of Sparebank1, it does not cost three times as much in terms of time and memory since Notability uses biometric APIs for fewer features than Sparebank1.

Table 4. Average runtime and memory usage for classifying different apps

	#Total methods	Average cost Time (s)	Average cost Memory (MB)
Sparebank1	89,142	32.79 ± 0.72	584.37 ± 1.93
Revolut	86,457	30.25 ± 0.78	518.56 ± 1.84
Paypal	94,618	35.96 ± 1.23	604.62 ± 1.89
Private Photo Vault	175,841	43.19 ± 1.48	659.07 ± 2.24
NordLocker	115,294	30.57 ± 1.55	556.29 ± 2.05
Notability	284,585	65.48 ± 2.14	844.73 ± 3.21
1Password	134,806	31.33 ± 1.67	581.46 ± 2.23

6.4 Evaluating the precision

To examine the classification precision, we manually go through the methods that got classified by each behavioral label. The methods were extracted from the decompiled `apk` files and read by our technique. Table 5 displays the total number of methods analyzed in each application as well as the classification results for various behaviors (categories). It is worth noting that all finance apps use the cryptography-related API, and none of the apps have access to actual biometric data storage, database, or acquisition (the only acquisition that happens is the cryptographic keys involved in the authentication process, no biometric data involved). This is aligned with GDPR requirements since transporting and storing biometric data is very sensitive and requires further consent from data owners.

According to Table 6, PABAU has an average precision of 0.84 across all the labels. It is more precise for detecting **Source** and **Sink** (0.96), as well as some categories such as **BSC1/2/3** (0.97), than for other categories such as **Termination** (0.87). Some of the behavior labels appear infrequently in the dataset as well.

PABAU can be improved by using a larger and more diverse training set that includes real-world API usage annotations and domain-specific information.

Table 5. Total number of methods analyzed (#M) and numbers of methods detected by PABAU per behavioral label.

	Klarna	Sparebank1	Revolut	Paypal	PPV	Nordlocker	Notability	1Password
#M	105,185	89,142	86,457	94,618	175,841	115,294	284,585	134,806
Source	127	94	174	140	65	98	39	55
Sink	96	70	143	109	41	64	17	32
BSC1	15	12	21	16	15	20	11	13
BSC2	17	24	29	14	7	9	0	5
BSC3	2	2	3	2	0	0	0	0
Checker	9	7	13	8	3	4	1	3
Permission	7	5	9	8	1	1	1	3
Auth	8	6	9	11	2	3	2	4
Crypto	5	5	6	9	0	0	0	0
Termin	9	7	10	9	3	1	1	5
Interact	4	2	5	7	1	2	1	4
Transfer	0	0	0	0	0	0	0	0
Acquist	1	1	1	1	0	0	0	0
Delete	2	1	2	3	0	0	0	0
Storage	0	0	0	0	0	0	0	0

7 Threats to validity

To check for GDPR compliance in the biometric authentication system and find relevant behavior, here we picked one of the most often used sample templates for generating a DPIA from the Commission nationale de l'informatique et des libertés (CNIL) [1].

Under "*Section 2 Assessment of controls protecting data subjects' rights*", there are key questions that can be answered by the classification results from PABAU. We pick the following examples which can be answered by our classification result:

– *Determination and description of the controls for obtaining consent.*
  Typically, the operating system, such as Apple or Google, obtains biometric permission first from users when they enroll their biometrics into the system. Permission is required the first time an application asks for access to the biometric API. This refers to the methods we classified with the **Permission** label, these methods indicate the code location and description of biometric permissions sought by the application. From the preceding experiment result, it is evident that all apps that implemented biometric authentication requested permission. By analyzing the discovered methods, further information can be uncovered. For instance, the use

Table 6. Number of biometric-related methods detected (#BM), and precision of PABAU for each label on eight Android applications. '/' marks labels for which PABAU detected no methods.

	Klarna	Sparebank1	Revolut	Paypal	PPV	Nordlocker	Notability	1Password
#BM	326	201	427	372	170	233	89	143
Source	0.95	0.94	0.92	0.9	0.99	0.99	0.97	0.99
Sink	0.97	0.95	0.97	0.94	0.98	0.99	0.99	0.96
BSC1	0.93	1	1	0.89	1	0.95	1	0.92
BSC2	1	0.96	0.93	1	1	0.89	/	1
BSC3	1	1	1	1	/	/	/	/
Checker	0.67	0.57	0.77	0.5	0.33	0.5	1	0.67
Permission	1	0.8	0.67	0.89	1	1	1	1
Auth	1	1	1	1	1	1	1	1
Crypto	1	1	1	1	/	/	/	/
Termin	0.56	0.86	0.8	1	1	1	1	0.8
Interact	1	1	1	0.86	1	1	1	1
Transfer	/	/	/	/	/	/	/	/
Acquist	1	1	1	0	/	/	/	/
Delete	0.5	1	1	0.3	/	/	/	/
Storage	/	/	/	/	/	/	/	/

of **Permission** methods may be used to examine privacy policies to determine whether there are locations that utilize biometric data without user consent.

– *Information on the secure data storage method, particularly in the event of sourcing.* Various biometric security class levels signify varying degrees of security. DPOs need to understand how user data is being securely stored. Labels **Crypto** and **BS3** offer the greatest degree of data protection and inform DPOs that the biometric authentication method utilized sophisticated encryption.
– *Detailed presentation of the data processing purposes (specified objectives, data matching where applicable, etc.).* Biometric APIs are utilized for more than just app unlocking. The label **Iteract** classifies the interaction between the user and the application in terms of biometric API usage, as user interaction must be precisely identified. By analyzing these approaches, DPOs can determine which activity motivates the use of biometric APIs. Similarly, labels such as **Auth** explain the authentication procedure in a clear manner by pinpointing the locations of authentication processes and associating the context in the code.
– *Possibility of retrieving, in an easily reusable format, personal data provided by the user, so as to transfer them to another service* Transferring biometric data or the authentication session's decisions is a highly sensitive process. In spite of the fact that our experiment did not reveal any methods with the **Transfer** label, it is still worthwhile to discover them in the implementations. Considering the Android biometric data is stored in the Trusty TEE (Trusted Execution Environment) [23], which is isolated from the rest of the system by both hardware and software. By analyzing the Android implementation, we should not find any data transportation process since Trusty and Android run parallel to each other.
– *Indication of the personal data that will nevertheless be stored (technical requirements, legal obligations, etc.).* The **Storage** and **Delete** labels enable DPOs to identify and comprehend methods that may breach GDPR data retention requirements.

8 Conclusion and future work

It has always been a barrier between technical and non-technical people in terms of privacy protection for biometric data. DPOs rely on technical specifications from developers to make legal decisions in order to continue monitoring privacy protection compliance in software implementations.

In this paper, we propose a technique for labeling biometric API usage with behavioral labels. This approach overcomes this barrier by providing an automated method for both parties to gain a quick overview of the important fundamental behaviors of biometric API in applications, as well as future assistance to DPOs with legal paperwork, such as performing a DPIA.

Our work provides an early promising result in categorizing the behaviors of biometric API methods in eight popular apps. However, we did not have a large number of samples for the training set, the diversity can also be improved. Meanwhile, experienced developers' involvement might benefit the learning process by providing valuable relevant elements for training, such as varied weights for features. We hope that this method will help both technical and non-technical workers gain a better grasp of privacy protection scenarios during the software development process.

Acknowledgment

We appreciate the insightful remarks made by Bjarte M. Østvold and Amina Bassit. This work is part of the Privacy Matters (PriMa) project. The PriMa project has received funding from European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 860315.

References

1. Privacy Impact Assessment (PIA) by CNIL. `https://www.cnil.fr/en/privacy-impact-assessment-pia` (February 2018), (Accessed on 08/19/2022)
2. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., McDaniel, P.: Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. Acm Sigplan Notices **49**(6), 259–269 (2014)
3. Avdiienko, V., Kuznetsov, K., Gorla, A., Zeller, A., Arzt, S., Rasthofer, S., Bodden, E.: Mining apps for abnormal usage of sensitive data. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. vol. 1, pp. 426–436. IEEE (2015)
4. Bettini, C., Riboni, D.: Privacy protection in pervasive systems: State of the art and technical challenges. Pervasive and Mobile Computing **17**, 159–174 (2015)
5. Council of European Union: General data protection regulation (GDPR) (2018), `https://gdpr-info.eu/`
6. Council of European Union: General data protection regulation (GDPR) Art. 9: processing of special categories of personal data (2018),
`https://gdpr-info.eu/art-9-gdpr/`
7. Dias Canedo, E., Toffano Seidel Calazans, A., Toffano Seidel Masson, E., Teixeira Costa, P.H., Lima, F.: Perceptions of ICT practitioners regarding software privacy. Entropy **22**(4), 429 (2020)
8. Fernández, A.P., Sindre, G.: Software Assisted Privacy Impact Assessment in interactive ubiquitous computing systems. In: Conference on e-Business, e-Services and e-Society. pp. 60–71. Springer (2019)
9. Gruschka, N., Mavroeidis, V., Vishi, K., Jensen, M.: Privacy issues and data protection in big data: a case study analysis under GDPR. In: 2018 IEEE International Conference on Big Data (Big Data). pp. 5027–5033. IEEE (2018)
10. Hadar, I., Hasson, T., Ayalon, O., Toch, E., Birnhack, M., Sherman, S., Balissa, A.: Privacy by designers: software developers' privacy mindset. Empirical Software Engineering **23**(1), 259–289 (2018)
11. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. ACM SIGKDD explorations newsletter **11**(1), 10–18 (2009)
12. Hansen, M.: Top 10 mistakes in system design from a privacy perspective and privacy protection goals. In: IFIP primelife international summer school on privacy and identity management for life. pp. 14–31. Springer (2011)
13. Henriksen-Bulmer, J., Faily, S., Jeary, S.: Dpia in context: Applying DPIA to assess privacy risks of cyber physical systems. Future Internet **12**(5), 93 (2020)

14. Hoepman, J.H.: Privacy design strategies. In: Cuppens-Boulahia, N., Cuppens, F., Jajodia, S., Abou El Kalam, A., Sans, T. (eds.) ICT Systems Security and Privacy Protection. pp. 446–459. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
15. Horák, M., Stupka, V., Husák, M.: Gdpr compliance in cybersecurity software: A case study of dpia in information sharing platform. In: Proceedings of the 14th international conference on availability, reliability and security. pp. 1–8 (2019)
16. Jasserand, C.: Legal nature of biometric data: From generic personal data to sensitive data. Eur. Data Prot. L. Rev. **2**, 297 (2016)
17. Lioudakis, G.V., Koukovini, M.N., Papagiannakopoulou, E.I., Dellas, N., Kalaboukas, K., Carvalho, R.M.d., Hassani, M., Bracciale, L., Bianchi, G., Juan-Verdejo, A., et al.: Facilitating GDPR compliance: the H2020 BPR4GDPR approach. In: Conference on e-Business, e-Services and e-Society. pp. 72–78. Springer (2019)
18. Piskachev, G., Do, L.N.Q., Bodden, E.: Codebase-adaptive detection of security-relevant methods. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2019, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3293882.3330556, `https://doi.org/10.1145/3293882.3330556`
19. Sas, D., Bessi, M., Fontana, F.A.: Automatic detection of sources and sinks in arbitrary java libraries. In: 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM). pp. 103–112. IEEE (2018)
20. Tang, F., Østvold, B.M.: Assessing software privacy using the privacy flow-graph. In: Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security. MSR4P&S'22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3549035.3561185, `https://doi.org/10.1145/3549035.3561185`
21. Vallée-Rai, R., Co, P., Gagnon, E., Hendren, L., Lam, P., Sundaresan, V.: Soot: A Java bytecode optimization framework. In: CASCON First Decade High Impact Papers, pp. 214–224 (2010)
22. Wang, L., LI, F., LI, L., et al.: Principle and practice of taint analysis. Journal of Software **28**(4), 860–882 (2017)
23. Wikipedia contributors: Trusted execution environment — Wikipedia, the free encyclopedia (2022), `https://en.wikipedia.org/w/index.php?title=Trusted_execution_environment&oldid=1105563332`, [Online; accessed 15-October-2022]
24. Zibuschka, J.: Analysis of automation potentials in privacy impact assessment processes. In: Computer Security, pp. 279–286. Springer (2019)

PAPER 3

Identifying Personal Data Processing for Code Review

Tang, F.; Østvold, B. and Bruntink, M. (2023). Identifying Personal Data Processing for Code Review. In Proceedings of the 9th International Conference on Information Systems Security and Privacy - ICISSP; ISBN 978-989-758-624-8; ISSN 2184-4356, SciTePress, pages 568-575.

Identifying Personal Data Processing for Code Review*

Feiyang Tang[1], Bjarte M. Østvold[1], and Magiel Bruntink[2]

[1] Norwegian Computing Center, Oslo, Norway
[2] Software Improvement Group, Amsterdam, The Netherlands

Abstract. Code review is a critical step in the software development life cycle, which assesses and boosts the code's effectiveness and correctness, pinpoints security issues, and raises its quality by adhering to best practices. Due to the increased need for personal data protection motivated by legislation, code reviewers need to understand where personal data is located in software systems and how it is handled. Although most recent work on code review focuses on security vulnerabilities, privacy-related techniques are not easy for code reviewers to implement, making their inclusion in the code review process challenging. In this paper, we present ongoing work on a new approach to identifying personal data processing, enabling developers and code reviewers in drafting privacy analyses and complying with regulations such as the General Data Protection Regulation (GDPR).

Keywords: Data privacy protection · Code review · Static analysis.

1 Introduction

The General Data Protection Regulation (GDPR) lays the legal foundation for data protection in the EU and increases individual data protection rights throughout Europe. It also carries significant fines of up to 4% of yearly worldwide revenue for businesses that do not comply with the legislation. Many IT system providers, especially software-producing firms, may need to alter their systems in order to comply with the GDPR. This is predicted to require significant effort [4]. As a result, providing software engineers in the industry with effective and systematic ways to build data protection into software is an essential and beneficial study topic [13]. Organizations are pushing security to the software development life cycle, such as code review, to prevent software security vulnerabilities [5]. Similarly, to comply with privacy-by-design and perform privacy analysis tasks, code reviewers would benefit from similar tools to those used for security to identify privacy-related patterns in software.

Developers address privacy concerns using data security terminology, and this vocabulary confines their notions of privacy to threats outside of the organization [10]. However, even though data security is the main prerequisite of data privacy, privacy protection in software is still very much different from traditional security-related vulnerabilities. And according to Bambauer: "security and privacy can and should be treated as distinct concerns" [1]. Developers struggle to convert legal, ethical, and social privacy concerns into concrete technology and solutions [16].

Assessing privacy involves not only finding personal data in the software but also evaluating compliance with the related processing. GDPR defines as processing: "any operation or set of operations which is performed on personal data or on sets of personal data, whether or not by automated means." The definition encompasses a vast range of actions performed on personal data, such as collecting, recording, organization, structuring, storage, adaption or modification, retrieval, transit, etc. Privacy assessment tasks beg the question: How can we assist code reviewers and software developers in assessing personal data processing? By identifying personal data and the relevant processing in the system, code reviewers can uncover interesting patterns and utilize them to redesign the system to be more privacy-friendly or perform privacy analysis.

* Published at *the 9th International Conference on Information Systems Security and Privacy* (ICISSP 2023), DOI: 10.5220/0011725700003405

In this paper, we present ongoing work on a novel approach designed to assist developers and code reviewers in identifying personal data processing, which can subsequently be used for privacy analysis. This enables developers and code reviewers to assist organizations with a variety of important privacy-related tasks, such as completing a data protection impact assessment (DPIA) and creating a privacy policy.

2 Related work

An essential step in the software development process, code reviewing incorporates both manual and/or automated reviews. The main goal of code reviews is to assess and boost the code's effectiveness and correctness, pinpoint security issues, and raise its quality by adhering to best practices [15]. To automatically evaluate code, a variety of vulnerability detection tools have been built. They are also known as source code analyzers or static analysis tools, as they can analyze a program's code without having to execute it [14].

CodeQL[3], and Semgrep [20][4] are two popular code review tools that utilize static analysis. CodeQL treats code as if it were data, and issues are modeled as queries. Following the extraction of these queries from the code, they are executed against a database. The database is a directory containing data, a source reference for displaying query results, query results, and log files. Semgrep matches grammatical patterns on parsed programs (represented as an Abstract Syntax Tree (AST)) instead of matching string or regular expression (regex) patterns on the program as a string. Semgrep makes it considerably simpler to construct customized rules than CodeQL, which needs rules to be defined in QL, a declarative object-oriented query language.

There is relatively little published work that focuses on code reviews to identify privacy-related vulnerabilities, and it is problematic to translate current security knowledge to privacy, which we will explain in Section 3. There are studies on the identification of personal data that are valuable to our research. Fugkeaw et al. [9] proposed AP2I to enable organizations to identify and manage personal data in the local file system automatically. By monitoring network traffic, ReCon [21] utilized machine learning to identify probable personal data breaches. van der Plas et al. [18] used CodeBERT, a RoBERT-like transformer model, to identify personal data in Git commits.

3 Background and challenges

Data privacy analysis is becoming as crucial as security vulnerability discovery and has brought a new dimension to the data security dilemma [3]. It is advantageous for code reviewers to be able to conduct a similar privacy analysis that they did for security.

The current state of the art is mostly focused on security analysis. Although data security is a primary requirement for data privacy, the analysis domain and identification process are rather different [11]. Simply adopting security mechanisms and mindsets to analyze privacy can be misguided, and even harmful [1].

Integration of recent studies on assessing software privacy during code review is challenging. On the subject of program analysis, three well-known privacy analysis methods are available. First, static analysis based on bytecode requires project compilation, whereas dynamic taint analysis requires project execution. This is not practical nor efficient for code reviewers to implement. A machine learning-based technique is similarly difficult to implement, as it requires a large and diverse training data set. Obtaining and generating such data sets requires additional effort and could be outside the scope of code reviewers' capabilities. Lastly, text analysis based on UI widgets is constrained for privacy by domain-specific UI attributes. A financial web application that employs

[3] `https://codeql.github.com/`
[4] `https://semgrep.dev/`

a model trained on an Android health mobile application is unlikely to benefit. Code reviewers require an approach that is simple to deploy, efficient, and adaptable [6].

Due to the complex nature of privacy and the fluidity of the definition of personal data, identifying the processing of personal data in the codebase presents challenges.

In the following paragraphs, we highlight the two most significant challenges related to the task in the context of code review.

3.1 The Ambiguous Definition of Personal Data

Article 4(1) in GDPR defines personal data as:

> *any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person.*

The definition of personal data in the GDPR is so broad that almost any information may qualify as personal data if it refers to a specific individual, such as the fact that a person is wearing a red shirt [2]. The definition is also semantically ambiguous.

In contrast to the fact that certain data may be anonymous from the start (such as weather sensor data without any connection to real people), other data may initially be personal data but later be successfully altered to no longer have any connection to an identified or identifiable natural person. This emphasizes how flexible the categorization of personal data is [8].

The same data point may be personal or non-personal depending on the context and may thus be covered by the regulation or not. This implies that the categories of personal data in the software vary depending on the software and the processing underlying it. For instance, health data such as blood pressure and medical records, for example, are sensitive for a health application, but location data is sensitive for navigation software.

Even if we accept that content-wise every item of information can be considered personal data if it can be related to an individual, the GDPR's definition is still rather vague structurally since it is not always clear what kind of structure every 'record' of an individual must have to be considered personal data [23].

Due to the ambiguous nature of the definition of personal data in the relevant legislation, it is practically difficult for us to have a clear and fixed identifier to precisely locate personal data in code.

3.2 What Counts as Sensitive Processing?

Data subjects may agree to data processing for particular reasons. This is the usual legal basis but only counts as one factor. Processing may also be "necessary for the performance of a contract to which the data subject is a party or in order to take steps at the request of the data subject prior to entering into a contract." [23]

Unfortunately, concerns that arise in principle about the relationship between contract and consent tend to be avoided in reality by disregarding consent requirements [19].

We cannot rely on existing privacy policies and written consent to uncover personal data processing in the codebase. This requires us to consider all potential personal data processing in the codebase. Later we will explain how we define and identify the relevant processing in software in Section 4.1.

4 Approach

We present an approach to identify instances of personal data processing in the codebase and present them in a way that facilitates the code review.

The approach has three primary phases: pattern matching, labeling, and grouping of results. As input, we take the codebase, which consists of source code files. Then, a static analyzer will evaluate these source code files using our rules and patterns. The code snippets discovered by the static analyzer are then labeled according to the various features they include. Finally, we allow users to group the results by single or several labels, allowing a personalized exploration of the findings.

An illustration of our approach is shown in Figure 1.

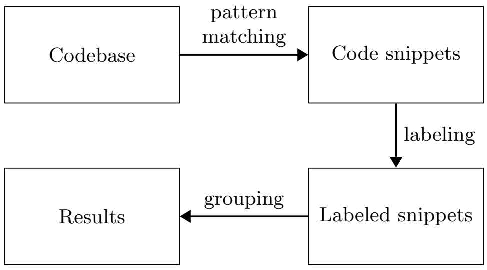


Fig. 1. Approach

4.1 Design Choices

In the following paragraphs, we discuss our design choices for implementing the approach.

Types of Findings We want to have a basic default list of personal data that we want to locate, this is mostly personal identification and characteristics data, such as full name, email address, gender, sexual orientation, and age. We call them fixed personal data.

According to different types of software, we customize default lists for them. For example, for a banking/finance application, the list may contain bank account numbers, credit scores, and salary information. This type of personal data is subject to context - the types of processing in specific software, which we named contextual personal data.

Depending on how we locate the mentioned personal data in the software, we can divide their occurrences in code into simply two types.

The first is in clear text. This includes all kinds of locations where personal data appear in clear text. It is verbatim or direct personal data. For example, a credit card number appears in an SQL query, or an email address falls into a log function.

The other type is more common and subtle, where personal data is stored in a variable or an object. Depending on the different types of programming languages, the object types might vary from a local variable, a class instance, or a prototype. This means we aim to find the code that processes this type of data.

Types of Processing Simply locating every instance of personal data produces a large number of results. Many of these do not directly help the code reviewer's work, which is to find meaningful processing. We want to use a hybrid approach to cover as many as processing as possible.

Processing personal data represents a specific behavior. This motivates our first approach: to use an action name tag to find relevant processing. We adopted most of the verbs from Section 3 of DPV [17] [5]. These vocabularies help us to find relevant processes in the software.

The second approach is the identification of external libraries. We know that modern applications rely on various APIs to achieve different goals. Therefore, obtaining a list of relevant APIs and detecting the existence of personal data that flows into them helps us find meaningful patterns.

4.2 Pattern Matching

The first step is to feed our codebase (consisting of source code files) to the static analyzer for pattern matching. We chose Semgrep as our analyzer because of its user-friendly rules and rapid processing performance. Depending on the different syntactic characteristics of personal data, as we discussed in Section 4.1, we adopt a hybrid approach that combines two different types of analysis.

– Match personal data in clear text using regular expression matching.
– Taint analysis to find flows in each file between a source (where personal data enters the analysis scope) and a sink (where personal data gets processed) that match our criteria.

Our personal data processing rules currently support Java, JavaScript, and TypeScript as our primary analysis domains. However, our rules for identifying clear-text personal data apply to the vast majority of Semgrep-supported languages.

Source and Sink Our prototype classifies the sources into nine separate categories. As stated in Section 4.1, we divide fixed personal data into four different categories: *account*, *contact*, *national ID*, and *personal ID*. Included are five more contextual personal data categories, such as *location*, *health*, and *financial* data. In addition, we provide a template for identifying the processing of personal data and enable code reviewers and developers to submit additional personal data simply by entering the relevant keywords. Then, corresponding rules will be automatically produced for future use.

Sinks are categorized into five main types. Three types of action: *data manipulation* (**M**), *data transportation* (**T**), and *data creation/deletion* (**C/D**). Another two represent two special types: *database* (**DB**) and *encryption* (**E**).

A sink's name may contain a specific type of source. For example, `setLatitude(100,100)` does not take any source into the method, but includes a source identifier `Latitude` and a sink identifier `set`, showing that it processes values directly as a source into a sink. We call this special type of sink a source-specific sink. When a source-specific sink invokes anything, we mark this source-specific sink as the new source but the caller of the source-specific sink as the new sink. For example, in `gpsTracker.setLatitude(100,100)`, `setLatitude` becomes the new source and `gpsTracker` is the new sink.

Inspired by how Privado [6] uses regular expressions to identify GDPR-related data in Java applications, a sample Semgrep rule that matches the pattern of *account* data source goes into a *transportation* (**T**) sink is shown below in Figure 2, followed by a sample code snippet detected in Figure 5.

4.3 Labeling

The identified findings from Semgrep are in the form of various lengths of code snippets (consisting of statements and expressions). Each finding contains at least one detected sink and one source (or an object that received value from a source). We abstract the structure of possible sources and sinks in each code snippet using the symbols below.

[5] `https://w3c.github.io/dpv/dpv/`
[6] `https://www.privado.ai`

```yaml
rules:
  - id: account-data-transportation
    languages:
      - javascript
      - java
      - typescript
    mode: taint
    message: Match found
    pattern-sinks:
      - patterns:
          - pattern: $SINK(..., $Z, ...)
          - metavariable-regex:
              metavariable: $SINK
              regex: (?i)(.*(send|move|connect|escap|stream|redirect|
              erase|query|share|stor|transfer|transmit|move).*)
    pattern-sources:
      - pattern-regex: (?i).*(?:account|user|customer|doctor|patient|
      policyholder|insurer|claimant)[^\\s/(;)|,=!>]{0,3}(id|number|no|
      num)
      - pattern-regex: (?i)(?:facebook|twitter|instagram|linkedin|
      pinterest|behance|dribble)[^\\s/(;)|,=!>]{0,2}(?:id|account|
      username|handle)
      - pattern-regex: (?i).*(?:db|database|jira|sql|postgres|mongo|aws)
      [^\\s/(;)|,=!>]{0,3}(psw|pswd|password|passwrd)
      - pattern-regex: (?i)(.*(?:db|database|jira|sql|postgres|mongo|
      aws)[^\\s/(;)|,=!>]{0,3}user[^\\s/(;)|,=!>]{0,3}name)|(.*(account|
      customer|doctor|patient|teacher|student|person|organi[zs]ation|
      company)[^\\s/(;)|,=!>]{0,3}name)
    severity: WARNING
```

Fig. 2. Semgrep rule: find personal data flows from account data source to transportation sink

```javascript
this.usersService.updateUser(newUser.id,
                             {defaultOrganizationId:
                              newUser.organizationId})
                               .catch((error) => {
      console.error('Error while updating default
                    organization id', error);
    });
```

Fig. 3. Sample code snippet (from ToolJet) detected by Semgrep showing a flow from account personal data to a transportation sink.

- O ranges over sources
- I ranges over sinks
- I^O ranges over source-specific sinks

We write $\bar{O}$ as shorthand for a possibly empty sequence $O_1, \cdots, O_n$. Here the underscore _ represents a placeholder for an expression that is insignificant in terms of privacy - it is neither a source nor sink nor contains a value from a source.

Below is a list of the common flow abstracts between sources and sinks that we observed in each code snippet. Each abstract represents a typical flow, for example, ① to ③ show that there are values passing through a sink to a source, from a non-privacy sensitive value (①) or from another source (②) or from innovating a sink inside another source object (③).

① $O = \_.I(\_)$
② $O_2 = \_.I(O_1, \_)$
③ $O_2 = \_.O_1.I(\_)$
④ $\_ = \_.O.I(\_)$
⑤ $\_ = \_.I(\bar{O}, \_)$
⑥ $\_.O.I(\_)$
⑦ $\_.O.I(\_, \bar{O})$
⑧ $\_.I^O(\_)$
⑨ $\_.I^O(\_, \bar{O})$
⑩ $\_.I(\bar{O}, \_)$

For each identified code snippet, we label them with 22 labels (9 types of source, 5 types of sink, 5 types of source-specific sink, and 3 types of change in the sensitivity level), which are listed in Table 1. Besides the definition of source and sinks, we also introduce an important label: sensitivity. The sensitivity level can increase, decrease, and stay the same in one identified code snippet.

$\mathcal{O}$	Nine types of source: $\{O^1, O^2, \ldots, O^9\}$
$\mathcal{I}$	Five types of sink: $\{I^1, I^2, \ldots, I^5\}$
$\mathcal{I}^O$	Five types of source-specific sink : $\{I^{O^1}, I^{O^2}, \ldots, I^{O^5}\}$
$\mathcal{S}$	Sensitivity level change: {`up, down, equal`}

Table 1. Labels to be assigned to each code snippet

Sensitivity Level Not every result shares the same level of sensitivity regarding personal data processing. After processing, the data from the source might remain at a similar sensitivity level, become more sensitive, or become less sensitive.

– $\mathcal{S}$ = `up`: ①, ④, ⑤
– $\mathcal{S}$ = `equal`: ②, ③, ⑥, ⑦, ⑧, ⑨
– $\mathcal{S}$ = `down`: ⑩

4.4 Result Presentation

Johnson et al. [12] pointed out that "because the results are dumped onto a code reviewer's screen with no distinct structure causing him to spend a lot of time trying to figure out what needs to be done". This indicates that developers and code reviewers may not benefit from ungrouped code snippets from static analysis tools if they are not presented in a sensible manner.

To tackle this issue, we present a two-phase technique to process the findings from Semgrep and present them to code reviewers in a smart way.

After each code snippet is labeled, we start to group them for presentation using their labels and other criteria. Criteria for grouping include not only the labels but also other properties:

– neighboring results will be combined (same file and within a line number threshold);
– same or similar source/sink name;
– same API usage (e.g. every code snippet that is related to the same API MongoDB).

Figure 4 following provides a straightforward illustration of how we present our results. The results are presented in two separate sections: plain text results and flow results. Users have the flexibility to select any label or label combination to filter the results.

5 DEMONSTRATION

We created rules in Semgrep trying to capture as many useful findings for our analysis. The software we analyzed here is ToolJet[7], an open-source low-code framework for building React-based web applications. ToolJet's implementation is mostly in JavaScript and TypeScript. Users can build internal tools using ToolJet's prebuilt UI widgets to connect to data sources like databases, API endpoints, and external services. This means ToolJet has many parts that process personal data, which makes it a good starting example.

[7] `https://github.com/ToolJet/ToolJet`

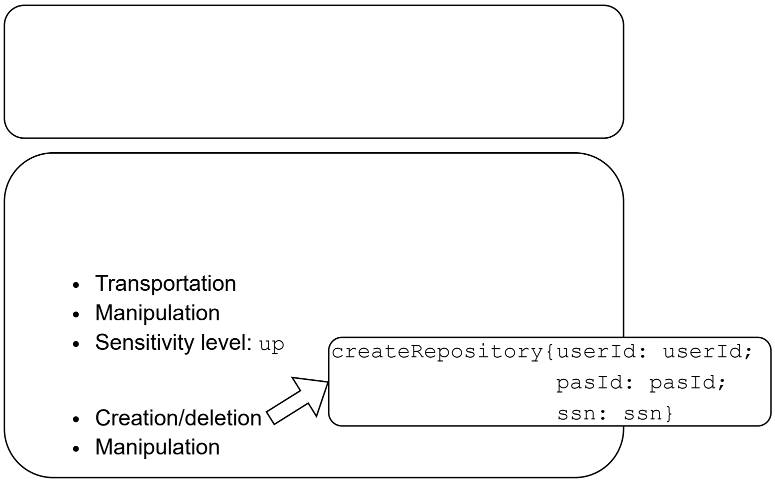


Fig. 4. Example presentation of the result. *Personal data occurrences* is at the top and *personal data processing code* is at the bottom.

Our Semgrep rules produce a total of 1,589 results from ToolJet's source code. We manually reviewed each of the results and calculated the precision for each category. If a single result can clearly demonstrate the processing of personal data, we consider it relevant and it could be beneficial for privacy code review. Surprisingly, most false positives come from the personal data occurrence detector (with a precision of only 46.6%), while most personal data processing results are relevant (with an average of 90.9% precision for categories that have more than 50 code snippets identified).

Detailed statistics are listed in Tables 2 and Table 3.

	M	T	C/D	DB	E	L
Account	66	171	84	24	-	21
Contact	89	175	36	3	-	3
Personal ID	56	133	41	7	1	4
Online ID	6	26	1	-	-	1
Location	1	2	-	-	-	-

Table 2. The code snippet count for each identified source and sink identified, '-' marks labels for which our approach detected no code snippet. Sink types are: *data manipulation* (**M**), *data transportation* (**T**), *data creation/deletion* (**C/D**), *database* (**DB**), *encryption* (**E**) and *log* (**L**).

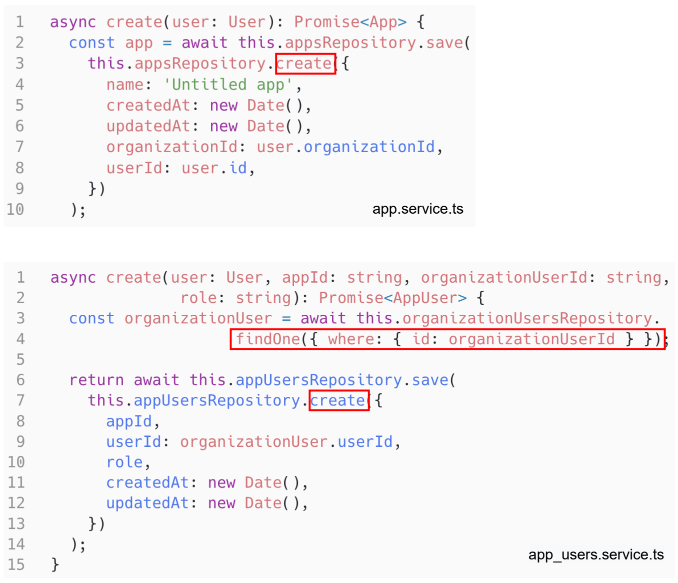
Figure 5 shows a simple interesting example of a grouped result showing how personal data `userId` is retrieved from a local repository in *app_users.service.ts* and then utilized to generate many data structures, such as the `app` object in *app_service.ts*.

5.1 Future work

Since our objective is to identify all relevant processing of personal data in source code, reducing false negatives is our next primary priority. However, in our case, false positives are not a major

	M	T	C/D	DB	E	L
Account	90.9	90.6	95.2	91.67	-	95.2
Contact	89.9	94.9	80.6	*	-	*
Personal ID	92.9	81.9	85.4	*	*	*
Online ID	*	84.6	*	-	-	*
Location	*	*	-	-	-	-

Table 3. The precision of code snippet relevance (in %) for each identified type of source and sink, '-' marks the labels for which our approach did not detect any code snippet, '*' marks the labels for which our approach detected less than 10 results. Sink types are: *data manipulation* (**M**), *data transportation* (**T**), *data creation/deletion* (**C/D**), *database* (**DB**), *encryption* (**E**) and *log* (**L**).



```
async create(user: User): Promise<App> {
  const app = await this.appsRepository.save(
    this.appsRepository.create({
      name: 'Untitled app',
      createdAt: new Date(),
      updatedAt: new Date(),
      organizationId: user.organizationId,
      userId: user.id,
    })
  );                                          app.service.ts
```

```
async create(user: User, appId: string, organizationUserId: string,
             role: string): Promise<AppUser> {
  const organizationUser = await this.organizationUsersRepository.
                         findOne({ where: { id: organizationUserId } });

  return await this.appUsersRepository.save(
    this.appUsersRepository.create({
      appId,
      userId: organizationUser.userId,
      role,
      createdAt: new Date(),
      updatedAt: new Date(),
    })
  );
}                                                app_users.service.ts
```

Fig. 5. Grouped example results showing how `organizationUserId` flows between functions.

concern. Due to the subtlety of personal data processing, determining relevance without human assistance is particularly challenging. Specifying the analysis to certain specific patterns would ease manual analysis. This necessitates the implementation of a privacy taxonomy. Using Ethyca's taxonomy [7] as an example, we may modify our labels to match the technique with the taxonomy.

As an extension of this article, we propose an automated mapping of personal data in an unpublished (under review) manuscript [22] to assist developers and code reviewers in identifying privacy-related code. The mapping based on static analysis automatically detects personal data and the code that processes it, and we offer semantics of personal data flows.

6 CONCLUSIONS

This short paper presented ongoing work on a novel, customizable approach to identify personal data processing for code review. This three-phase technique first uses Semgrep to match patterns in the code based on rules for sources and sinks, then associates code snippets generated from pattern matching with a set of behavioral labels, and finally groups results to reduce code reviewer workload. Our demonstration shows the utility and feasibility of this method for gathering and presenting code snippets related to personal data processing from a codebase.

Along with the continued development of the approach architecture (refined rules for source and sink, more meaningful labels, and additional criteria for grouping), future work will focus on expanding the case study to include a larger set of open-source software from various domains and conducting a thorough user evaluation of the resulting platform.

ACKNOWLEDGEMENTS

This work is part of the Privacy Matters (PriMa) project. The PriMa project has received funding from European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 860315.

References

1. Bambauer, D.E.: Privacy versus security. J. Crim. L. & Criminology **103**, 667 (2013)
2. Berčič, B., George, C.: Identifying personal data using relational database design principles. International Journal of Law and Information Technology **17**(3), 233–251 (2009)
3. Bertino, E.: Data security and privacy: Concepts, approaches, and research directions. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC). vol. 1, pp. 400–407. IEEE (2016)
4. Blume, P.: Impact of the EU General Data Protection Regulation on the public sector. Journal of Data Protection & Privacy **1**(1), 53–63 (2016)
5. Braz, L., Bacchelli, A.: Software security during modern code review: The developer's perspective. arXiv preprint arXiv:2208.04261 (2022)
6. Buse, R.P., Zimmermann, T.: Information needs for software development analytics. In: 2012 34th International Conference on Software Engineering (ICSE). pp. 987–996. IEEE (2012)
7. Ethyca: Fides language. `https://ethyca.github.io/fideslang/` (2022), (Accessed on 11/15/2022)
8. Finck, M., Pallas, F.: They who must not be identified—distinguishing personal from non-personal data under the GDPR. International Data Privacy Law **10**(1), 11–36 (2020)
9. Fugkeaw, S., Chaturasrivilai, A., Tasungnoen, P., Techaudomthaworn, W.: AP2I: Adaptive PII scanning and consent discovery system. In: 2021 13th International Conference on Knowledge and Smart Technology (KST). pp. 231–236. IEEE (2021)
10. Hadar, I., Hasson, T., Ayalon, O., Toch, E., Birnhack, M., Sherman, S., Balissa, A.: Privacy by designers: software developers' privacy mindset. Empirical Software Engineering **23**(1), 259–289 (2018)
11. Jain, P., Gyanchandani, M., Khare, N.: Big data privacy: a technological perspective and review. Journal of Big Data **3**(1), 1–25 (2016)
12. Johnson, B., Song, Y., Murphy-Hill, E., Bowdidge, R.: Why don't software developers use static analysis tools to find bugs? In: 2013 35th International Conference on Software Engineering (ICSE). pp. 672–681. IEEE (2013)
13. Lenhard, J., Fritsch, L., Herold, S.: A literature study on privacy patterns research. In: 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 194–201. IEEE (2017)
14. McGraw, G.: Automated code review tools for security. Computer **41**(12), 108–111 (2008)
15. McIntosh, S., Kamei, Y., Adams, B., Hassan, A.E.: The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In: Proceedings of the 11th working conference on mining software repositories. pp. 192–201 (2014)
16. Notario, N., Crespo, A., Martín, Y.S., Del Alamo, J.M., Le Métayer, D., Antignac, T., Kung, A., Kroener, I., Wright, D.: PRIPARE: integrating privacy best practices into a privacy engineering methodology. In: 2015 IEEE Security and Privacy Workshops. pp. 151–158. IEEE (2015)
17. Pandit, H.J., Polleres, A., Bos, B., Brennan, R., Bruegger, B., Ekaputra, F.J., Fernández, J.D., Hamed, R.G., Kiesling, E., Lizar, M., et al.: Creating a vocabulary for data privacy. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 714–730. Springer (2019)

18. van der Plas, N.: Detecting PII in Git commits. TU Delft Master's thesis (2022)
19. Pormeister, K.: Informed consent to sensitive personal data processing for the performance of digital consumer contracts on the example of "23andMe". Journal of European Consumer and Market Law **6**(1) (2017)
20. r2c: Semgrep. `https://semgrep.dev/` (2022), (Accessed on 11/15/2022)
21. Ren, J., Rao, A., Lindorfer, M., Legout, A., Choffnes, D.: Recon: Revealing and controlling pii leaks in mobile network traffic. In: Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services. pp. 361–374 (2016)
22. Tang, F., Østvold, B.M., Bruntink, M.: Mapping personal data in source code for GDPR compliance (2023)
23. Voss, W.G., Houser, K.A.: Personal data and the GDPR: providing a competitive advantage for US companies. American Business Law Journal **56**(2), 287–344 (2019)

PAPER 4

Helping Code Reviewer Prioritize: Pinpointing Personal Data and its Processing

Tang, F.; Østvold, B. and Bruntink, M. (2023). Helping Code Reviewer Prioritize: Pinpointing Personal Data and its Processing. In Proceedings of the 22nd International Conference on Intelligent Software Methodologies, Tools and Techniques (SOMET 2023). DOI: 10.3233/FAIA230228

Helping Code Reviewer Prioritize: Pinpointing Personal Data and its Processing[⋆]

Feiyang Tang[1], Bjarte M. Østvold[1], and Magiel Bruntink[2]

[1] Norwegian Computing Center, Oslo, Norway
[2] Software Improvement Group, Amsterdam, The Netherlands

Abstract. Ensuring compliance with the General Data Protection Regulation (GDPR) is a crucial aspect of software development. This task, due to its time-consuming nature and requirement for specialized knowledge, is often deferred or delegated to specialized code reviewers. These reviewers, particularly when external to the development organization, may lack detailed knowledge of the software under review, necessitating the prioritization of their resources.
To address this, we have designed two specialized views of a codebase to help code reviewers in prioritizing their work related to personal data: one view displays the types of personal data representation, while the other provides an abstract depiction of personal data processing, complemented by an optional detailed exploration of specific code snippets. Leveraging static analysis, our method identifies personal data-related code segments, thereby expediting the review process. Our approach, evaluated on four open-source GitHub applications, demonstrated a precision rate of 0.87 in identifying personal data flows. Additionally, we fact-checked the privacy statements of 15 Android applications. This solution, designed to augment the efficiency of GDPR-related privacy analysis tasks such as the Record of Processing Activities (ROPA), aims to conserve resources, thereby saving time and enhancing productivity for code reviewers.

Keywords: Personal data processing · GDPR analysis · Static analysis · Code review.

1 Introduction

In the 21st century, companies have been collecting and distributing massive amounts of personal data [1]. The sheer volume and capacity to access and integrate data in unprecedented ways is novel [14]. To protect individual rights, the European Union's General Data Protection Regulation (GDPR) requires substantial data protection, posing both challenges and opportunities for global software companies and imposing severe fines for non-compliance [7]. Article 30 of the GDPR necessitates the creation of a detailed document, the Records of Processing Activities (ROPA), to ensure GDPR compliance. This document, which must be readily accessible to the supervisory authority, can be challenging to construct. Code reviewers often play a crucial role in this process, tasked with analyzing extensive codebases for GDPR-relevant aspects. Their findings not only help ensure compliance but also assist in the formation of the intricate ROPA, a task that can be quite demanding.

While the manual identification of personal data and its processing in the codebase is an inevitable part of a code reviewer's work, our goal is to facilitate the rapid pinpointing and understanding of relevant code fragments. To this end, we propose an approach that identifies and abstracts both personal data and its processing in the codebase. Our solution provides two specialized views: one that emphasizes the types of personal data, and another that outlines an abstract perspective of data processing flows.

[⋆] Published at the 22nd International Conference on Intelligent Software Methodologies, Tools and Techniques (SOMET 2023).

These views aim to empower code reviewers by streamlining the task of locating and comprehending personal data processing within the software. By providing a focused direction, highlighting key code fragments, and supplying task-specific information through different views, our approach conserves time and effort. This multi-faceted assistance greatly simplifies GDPR-related tasks, such as the formation of a ROPA.

The three main components of our approach are:

– A set of adaptable static analysis rules for pinpointing personal data and its processing in source code. (Refer to Section 3.1, *identification* in Figure 1)
– A collection of flow patterns derived from large-scale analysis, which simplifies identified code fragments containing a data flow into abstracted code snippets. These snippets capture the context and manner in which personal data is processed. (Refer to Section 3.2, *abstraction* in Figure 1)
– Two specialized views provide code reviewers with options for displaying information about personal data or flow-specific information, depending on their specific task, thereby reducing manual work. (Refer to Section 3.3, *presentation* in Figure 1)

We demonstrate the effectiveness of our approach by 1) achieving high precision and generating corresponding ROPAs compared to published privacy statements for four trending GitHub projects and 2) evaluating the accuracy of privacy statements provided by 15 popular Android applications from the Google Play store.

2 Challenges in Personal Data Identification

Identifying personal data within extensive codebases poses a significant challenge for code reviewers, especially given the diversity of personal data types and the lack of standardized patterns for their identification. This issue is further complicated by the reality that personal data may be either directly or indirectly associated with an individual, a relationship that can vary across cultural, linguistic, and domain-specific contexts. Privacy statements often describe personal data collection and processing in broad terms, making it difficult to pinpoint what data is collected and how it is processed within the source code.

When reviewing code for GDPR-relevant aspects, code reviewers often resort to manual techniques such as keyword searching, filtering, and grepping to identify potential areas of concern. However, these methods can be time-consuming and often yield an overwhelming number of results, making it challenging to discern the key areas of focus. There is a clear need for a more abstract, categorized view of the results that could help reviewers identify and focus on the most relevant code fragments.

User Requirement Study In order to grasp the challenges faced by code reviewers in GDPR-related tasks, we undertook a user requirement study with six experienced code reviewers from a medium-sized European software company. They were selected based on their experience, familiarity with GDPR tasks, and diverse European representation. The research, primarily focused on European data formats, did not consider formats outside the EU. The interviews elicited key issues in privacy analysis tasks, preferred presentation formats, and the potential of an abstract view of results.

During the structured interviews, participants discussed the most challenging facets of their privacy analysis tasks, the desired format for the presentation of findings, and their views on the possible advantages of having an abstract, categorized overview of the results. The study highlighted that identifying personal data within source code remains a significant hurdle for reviewers. This challenge arises primarily due to the fluctuating context and semantics. Participants voiced a preference for a results presentation that offers ample context and underscores potential areas of concern, such as personal data processing.

There was unanimous agreement among participants about the potential benefits of incorporating an automated, abstract presentation of the results into their workflows. They preferred comprehensive coverage, even if it occasionally introduced false positives, on the condition that their manual examination could be concentrated on a smaller, more relevant segment of the software. While this study has certain limitations, such as a smaller sample size and potential bias due to the limited participant selection, it does provide valuable initial insights into the needs and preferences of code reviewers tasked with GDPR-related duties in Europe.

Presentation Design Current static analysis tools offer some flexibility in results grouping and presentation. For instance, tools like FindBugs and ESLint allow results to be grouped by various criteria, such as bug category/name, code location, and bug ranking. However, these tools often lack the ability to provide a task-specific view that aligns with the code reviewers' current analysis focus. Such a view could allow reviewers to focus on personal data and its processing, thereby simplifying the analysis process and making it more efficient.

2.1 The Role of Code Reviewers in ROPA Creation

In the context of GDPR compliance, code reviewers often play a crucial role in creating a Record of Processing Activities (ROPA). A ROPA should contain detailed information about personal data processing, such as the categories of individuals and personal data, recipients of personal data, details of transfers to third countries, retention schedules, and technical and organizational security measures. Code reviewers, through their thorough analysis of the codebase, contribute significantly to the collection of this information, thereby helping to ensure GDPR compliance.

3 Constructing Task-Specific Views for Code Reviewers

We propose an approach that creates two distinct views, providing code reviewers with varying levels of information about personal data types and their processing within the codebase. Our approach employs static analysis to identify personal data and the source code where the processing of such data occurs. The result of the static analysis comprises identified code fragments that match the specified patterns. We introduce an abstracted flow representation that simplifies each identified code fragment, capturing the essence of personal data processing. This flow representation not only translates effectively into a more comprehensible format but also facilitates labeling and grouping of code fragments, presenting the results to code reviewers in a unified and efficient manner for analysis.

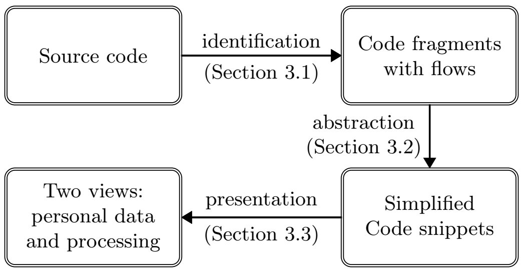


Fig. 1: Overview of our approach

Figure 1 presents an overview of our approach, which consists of three major phases: pattern matching with static analysis, abstraction of code fragments to generate flow representations, and creation of the two task-specific views for code reviewers.

3.1 Pattern Matching on Source Code

The first phase of our approach involves static analysis to pinpoint code fragments that contain or process personal data. For this task, we employ Semgrep, a powerful static analysis tool well-regarded for its flexibility and efficiency in analyzing voluminous source code files [9]. Semgrep's multi-language support and local data flow analysis capabilities are instrumental to our endeavor. This section elaborates on our rules for identifying personal data sources and sinks, and the subsequent extraction of flows.

Identification of Personal Data and Processing The concept of sources and sinks is vital to our approach. In the context of our analysis, sources refer to personal data, while sinks represent different forms of personal data processing. We define personal data as 1) literal personal data in source code text (constants identified via real values), and 2) variables (identified via name identifier). As for personal data processing, it refers to any distinct action or operation performed on personal data. Our rules for identifying personal data and its processing currently support Java, JavaScript, and TypeScript. Nevertheless, our rules applicable to plain-text personal data can be extended to the majority of Semgrep-supported languages. Semgrep parses the source code to build an abstract syntax tree (AST) for taint analysis, similar to how ESLint processes JavaScript code. This method enables us to efficiently identify sources, sinks, and data types. We further augment Semgrep's identification process with pattern matching.

Crafting Identification Rules Literal personal data identification relies on matching specified regular expressions, such as the syntax of national ID numbers. For variable sources, we have established a default list of personal data identifiers, covering data from 10 categories: *Account*, *Contact*, *Personal ID*, *Online identifier*, *Location*, *Feedback*, *Health*, *National ID*, *Technical*, and *Financial*.

These distinct identifiers are used to construct Semgrep rules with regular expressions (regex). To reduce false positives and enhance recall, we apply restrictions to the regex rules. For instance, to identify all human names in source code, we improve precision and cover first, last, and full names by using regex such as `(?i).(?:first|given|full|last|sur(?!geon)
)[s/(;)|,=!>]name)`.

Simultaneously, we identify potential sinks, which represent distinct actions of personal data processing. We utilize the majority of verbs from Section 3 of the Data Privacy Vocabulary (DPV) [10] to identify relevant processes. These verbs are utilized to generate corresponding regex for our taint analysis rules, and specific conditions are incorporated to pinpoint relevant sinks in the code effectively.

APIs are extensively used to implement functionalities in current software development. We performed a simple static analysis on the 20 most popular libraries each from Maven and npm (40 in total), followed by a manual inspection to filter out false positives. The resulting list consists of potential sink methods, predominantly related to databases from prominent providers such as AWS and Google Firebase. These API methods, akin to the verb identifiers, serve as sinks for our analysis. We categorize sinks into six classes: *Manipulation*, *Transportation*, *Creation/Deletion*, *Database*, *Encryption*, and *Log*.

Flow Extraction and Semgrep Output Once sources and sinks have been identified, we need to understand how personal data flows from the former to the latter. A flow represents a sequence

Table 1: Source and sink category abbreviations

Source type	Abbreviation	Sink type	Abbreviation
Account	ACC	Manipulation	M
Contact	CON	Transportation	T
Personal ID	PID	Creation/Deletion	C/D
Online identifier	OID	Database	D
Location	LOC	Encryption	E
Feedback	FEE	Log	L
Health	HEA		
National ID	NID		
Technical	TEC		
Financial	FIN		

of operations, starting from a source, moving through intermediate nodes (if any), and ending at a sink. We extend Semgrep's capabilities to not only identify but also comprehend these flows.

Semgrep's output typically includes the code fragment where the flow ends, i.e., the statement that processes personal data. However, we have customized Semgrep to provide more detailed information, such as the precise location and name of the source and sink, along with the code fragment. This information is vital for code reviewers, helping them quickly locate and comprehend personal data processing within the software.

Example: Application on ToolJet To illustrate the effectiveness of our approach, we applied our rules to ToolJet, a low-code, open-source framework for creating and deploying internal tools. ToolJet was selected because of its popularity and extensive use of personal data, making it a representative case study. The ease of access to its source code also facilitated our analysis.

Figure 2a depicts an example of a rule we created to identify flows of *account*-specific data into a *manipulation* sink in ToolJet. The corresponding identified code snippet, an output of Semgrep that includes both the code fragment and key information about the source and sink, is shown in Figure 2b.

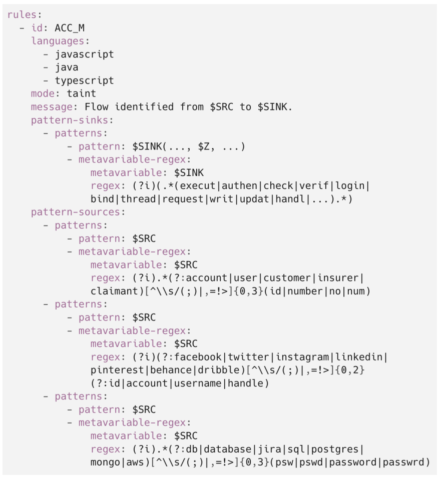
3.2 Flow Patterns: Abstracting Personal Data Flows

A critical aspect of our approach is the abstraction of identified personal data flows into more manageable and comprehensible representations. This step facilitates the review and understanding of these flows, offering a more streamlined approach to software privacy analysis. Here, we introduce the concept of flow patterns, which are simplified, standardized representations of personal data flows, distilled from the identified code fragments.

Semgrep outputs a statement of code, which corresponds to a chunk of a code fragment that could span multiple lines in the source code, where the personal data flow culminates. The value at the sink within this statement may not necessarily be the original source, yet it definitely encapsulates the value originating from the source. However, such code fragments can be intricate and challenging to comprehend. In our work, we address this by abstracting each result we capture, which represents a flow from a personal data source to a processing sink. Our abstraction combines this code fragment with source and sink information to depict the flow in a simplified form.

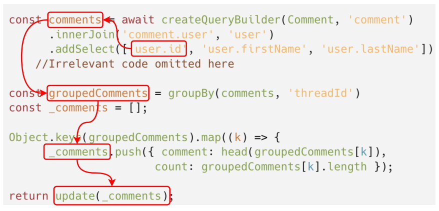
We applied our pattern-matching rules to over 20 open-source software projects on GitHub (the top 20 trending ones across three languages: JavaScript, TypeScript, and Java) and identified a vast number of personal data flow instances. From these instances, we generalized the structure of more than 150,000 detected code snippets into a set of eight distinct flow patterns.

Defining Flow Patterns The flow patterns, collectively denoted as $\mathcal{F}$, abstractly represent various forms of personal data flows from a source to or via a sink. Each pattern in $\mathcal{F}$ reflects a



```
rules:
  - id: ACC_M
    languages:
      - javascript
      - java
      - typescript
    mode: taint
    message: Flow identified from $SRC to $SINK.
    pattern-sinks:
      - patterns:
          - pattern: $SINK(..., $Z, ...)
          - metavariable-regex:
              metavariable: $SINK
              regex: (?i)(.*(execut|authen|check|verif|login|
              bind|thread|request|writ|updat|handl|...).*)
    pattern-sources:
      - patterns:
          - pattern: $SRC
          - metavariable-regex:
              metavariable: $SRC
              regex: (?i).*(?:account|user|customer|insurer|
              claimant)[^\\s/(;)|,=!>]{0,3}(id|number|no|num)
      - patterns:
          - pattern: $SRC
          - metavariable-regex:
              metavariable: $SRC
              regex: (?i)(?:facebook|twitter|instagram|linkedin|
              pinterest|behance|dribble)[^\\s/(;)|,=!>]{0,2}
              (?:id|account|username|handle)
      - patterns:
          - pattern: $SRC
          - metavariable-regex:
              metavariable: $SRC
              regex: (?i).*(?:db|database|jira|sql|postgres|
              mongo|aws)[^\\s/(;)|,=!>]{0,3}(psw|pswd|password|passwrd)
```

(a) Sample identification rule to find flows between *account* data and *manipulation* sink



```
const comments = await createQueryBuilder(Comment, 'comment')
      .innerJoin('comment.user', 'user')
      .addSelect(['user.id', 'user.firstName', 'user.lastName'])
    //Irrelevant code omitted here

const groupedComments = groupBy(comments, 'threadId')
const _comments = [];

Object.keys(groupedComments).map((k) => {
      _comments.push({ comment: head(groupedComments[k]),
                       count: groupedComments[k].length });

return update(_comments);
```

Semgrep output:

`return update(_comments);`
Message: *Flow identified from* `user.id` *to* `update`

(b) Code fragment identified using the rule illustrated in Figure 2a. In this case, the flow originates from the source **`user.id`** and ends at the sink **`update()`**. Along with the code fragment, Semgrep provides an accompanying message that details the names of both the source and the sink involved in the flow.

Fig. 2: An example of pattern matching on ToolJet

typical flow type, and Table 2 presents these flow patterns alongside their English interpretations. The table also introduces syntactic conventions for meta-variables used in the following discussion. E ranges over expressions, m ranges over methods, and v ranges over source variables. An expression with an underscore as an argument, $E[\_]$, signifies that the expression is not significant in terms of personal data processing — it is neither a source nor a sink, nor does it contain a value from a source.

In a flow pattern $E_1 \xrightarrow{m} E_2$, a solid arrow $\rightarrow$ indicates that a value on the left-hand side (LHS) contributes to the value on the right-hand side (RHS). If we are uncertain whether a value flows from the LHS to the RHS, such as when the LHS value is processed and merely outputs a Boolean to the RHS, we use a dashed arrow $\dashrightarrow$ to indicate that the LHS value may not fully reach the RHS. Each pattern's interpretation in English is also provided in Table 2.

The flow patterns in $\mathcal{F}$ abstract the flows between sources and sinks occurring in the identified code snippets. Table 3 offers eight examples illustrating the code snippets corresponding to flow patterns, ordered according to the flow patterns in Table 2.

These examples represent the different types of flows that may occur when processing personal data in code snippets. By identifying, analyzing, and categorizing these flows into simplified flow patterns, we can provide a uniform view of the data flows in code, capturing the properties of sources and sinks in a single flow, the pathway from source to sink, and any other involved values. This abstraction step is crucial in creating an intuitive and efficient approach to understanding personal data flows, thereby enhancing privacy protection in software systems.

3.3 Specialized Views for GDPR Analysis Tasks

As GDPR compliance tasks can vary in their requirements, we provide specialized views to assist code reviewers in understanding and analyzing personal data handling within the system. An overview of personal data types and their distribution is useful when developing a broad understanding of personal data usage, while a detailed analysis of each data flow is required for tasks like constructing a Record of Processing Activities (ROPA) or assessing the risk of personal data handling. By providing these specialized views, we aim to support reviewers in efficiently and effectively performing GDPR compliance tasks.

Table 2: The collection of flow patterns $\mathcal{F}$

Flow Patterns	**Translation to English**
$E[\_] \xrightarrow{m} v$	A non-personal data value flows via sink m into a source v.
$v_2 + E[\_] \overset{m}{\dashrightarrow} v_1$	Value from source v_2 and non-personal data are both processed by sink m and the result flows into source v_1.
$v_2(E[\_]) \xrightarrow{m} v_1$	Value from source v_2 flows via sink m into source v_1.
$v \overset{m}{\dashrightarrow} E[\_]$	Value from source v gets processed by sink m and the result flows into a new expression.
$v \xrightarrow{m} E[\_]$	Value from source v flows via sink m into a new expression, making it a new source.
$v_1 + v_2 \overset{m}{\to} v_1$	Value from source v_2 flows via sink m declared by v_1 into source v_1.
$v + E[\_] \overset{m}{\to} v$	Value from source v and non-personal data are both processed by sink m and the result flows into itself.
$v \overset{m}{\to} m(v)$	Value from source v flows into sink m.

Table 3: Examples demonstrating the relationship between code snippets and their corresponding flow pattern instances in $\mathcal{F}$. The examples are related to the flow patterns listed in Table 2.

Code Snippet	**Flow Pattern Instance**
`full_name = retrieve(record_data,2)`	$\text{E[\_]} \overset{\text{retrieve}}{\to} \text{full\_name}$
`isFemale = check(user_detail,'F')`	$\text{user\_detail} + \text{E[\_]} \overset{\text{check}}{\dashrightarrow} \text{isFemale}$
`first_name = UserInfo.get(2)`	$\text{UserInfo(E[\_])} \overset{\text{get}}{\to} \text{first\_name}$
`choice = match(name,list)`	$\text{name} \overset{\text{match}}{\dashrightarrow} \text{choice}$
`choice = UserInfo.retrieve(2)`	$\text{UserInfo} \overset{\text{retrieve}}{\to} \text{choice}$
`AccountInfo.update(userId,index)`	$\text{AccountInfo} + \text{userId} \overset{\text{update}}{\to} \text{AccountInfo}$
`AccountInfo.update(index)`	$\text{AccountInfo} + \text{E[\_]} \overset{\text{update}}{\to} \text{AccountInfo}$
`print(SSN)`	$\text{SSN} \overset{\text{print}}{\to} \text{print(SSN)}$

Personal Data Type View The Personal Data Type View presents an overview of personal data identified in the source code. This view is a hierarchical representation that illustrates the distribution of personal data types. The tree-like structure is automatically generated based on our static analysis findings, categorizing personal data identifiers by their nature such as *account* information or *personal ID*. For instance, all source names with a stem-identifier like `email_addr`, `email_id`, `e-mail`, and `email` are grouped together. This overview provides users with a clear image of the types of personal data and their respective identifiers present in the source code. See Figure 3 for an example.

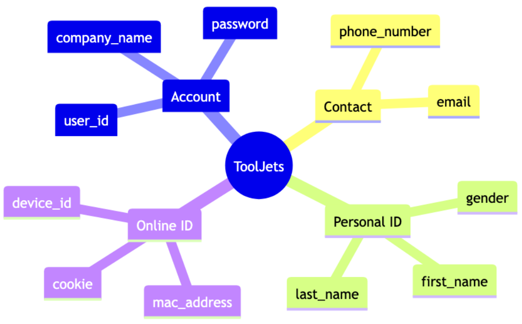


Fig. 3: Personal Data Type View of ToolJet

Detailed Flow View For tasks requiring a deeper understanding of specific personal data flows, we provide the Detailed Flow View. This view presents comprehensive information about each identified flow, including the file path, source and sink names and types, and abstract flow patterns. Furthermore, it offers the ability to link to the actual location of the code fragments, supporting a deeper contextual analysis when necessary. Features for filtering and ranking the identified flows are also included, allowing reviewers to focus on specific flows or those of high potential risk. An example of the Detailed Flow View is provided in Table 4.

These specialized views are designed to be flexible and adaptable to various GDPR compliance tasks. We incorporate grouping functionalities based on feedback from potential users, allowing results to be organized by specific source identifiers, file locations, or other relevant criteria. This flexibility is designed to make our approach practical, user-friendly, and adaptable to various GDPR compliance tasks.

Table 4: Detailed Flow View grouped by identifier `'email'` in ToolJet (top 7 results displayed)

Path	**Source**	**Sink**	**Sink Type**	**Flow Pattern Instance**
server/src/services/organizations.service.ts	users.email_addr	createQueryBuilder	DB	users.email_addr $\overset{\text{createQueryBuilder}}{\rightarrow}$ query
server/src/services/group_permissions.service.ts	users.email	createQueryBuilder	DB	users.email $\overset{\text{createQueryBuilder}}{\rightarrow}$ query
server/src/services/users.service.ts	email	this.usersRepository.findOne	DB	email+_ $\overset{\text{findOne}}{\rightarrow}$ UserInfo
server/src/services/organizations.service.ts	email_addr	this.usersService.create	C/D	email_addr+_ $\overset{\text{create}}{\rightarrow}$ UserInfo
server/ee/services/oauth/oauth.service.ts	email	this.usersService.findOrCreateByEmail	C/D	UserInfo+email $\overset{\text{findOrCreateByEmail}}{\rightarrow}$ UserInfo
server/src/services/users.service.ts	email	user.organizationUsers.sendData	T	email $\overset{\text{sendData}}{\rightarrow}$ sendData(email)
server/src/services/organizations.service.ts	email_addr	this.usersService.update	M	UserInfo+email_addr $\overset{\text{update}}{\rightarrow}$ UserInfo

In a Detailed Flow View, we present key information such as file path, source and sink names and types, and flow pattern instances, which are displayed by default. For example, Table 4 shows the Detailed Flow View for all source identifier `email` identification results. This information can be used to construct a ROPA using the official template provided by data protection authorities or a research semantic model like CSM-ROPA [13]. By traversing flow pattern instances in Table 4, users can generate a list of processing related to personal data "`email`" and identify their location.

To better assist GDPR compliance, our Detailed Flow View should not be limited to a fixed presentation style. Based on feedback and recommendations from potential users, we incorporate grouping to simplify GDPR compliance. We also refer to the ICO's ROPA template [8], which specifies the information required for GDPR compliance mentioned in Section 2.1, such as the categories of individuals and personal data, categories of recipients of personal data, transfer details to third countries, retention schedules, and technical and organizational security measures. Thus, we group the results by various attributes such as source and sink categories and their distinct identities, which can help users answer ROPA queries.

Moreover, users can experiment with different grouping criteria while examining the code. For example, they can group the results by a specific source identifier or file name/location of interest. We aim to incorporate GDPR-related grouping criteria to ease GDPR compliance and make our approach more useful.

3.4 Implementation

In recognizing the contextual nature of personal data, we have designed a flexible system allowing users to customize pattern-matching rules via a Python script. This script simplifies the definition of flow syntax and the addition of new identifiers. All our identification rules are open-source to encourage collaborative enhancement of the system's capabilities. The results of our analysis are produced in the Standard Static Analysis Results Interchange Format (SARIF), facilitating easy filtering and sorting based on parameters such as rules or file locations.

For visualizing the findings, we employ the Mermaid diagramming tool to automatically generate a personal data distribution graph. This visual representation aids in the identification and selection of specific source identifiers for further examination. Finally, we've adopted a SARIF viewer for filtering and sorting functionalities, and designed an interactive interface for the Specialized Views, offering users an adjustable and task-specific data exploration experience.

4 Experiment

To assess our approach, we focus on open-source software that processes personal data. The experiment consists of two parts. In the first part, we analyze popular open-source projects from GitHub, manually evaluating the true positives (TP) and false positives (FP) in the results to validate our approach. We illustrate the usefulness of our two specialized views by aligning them with the published privacy statements of four validation applications, showcasing how these views can guide the creation of a ROPA. In the second part, we apply our approach to 15 popular Android applications on Google Play, scrutinizing the accuracy of the information given in the "Data collected" section of Google Play's "Data safety" section.

4.1 Validation: Analyzing Trending GitHub Applications

Since we have used ToolJet in the previous sections as an example, here we select our analysis target from the *trending repositories on GitHub* list. Among the repositories captured from the GitHub monthly trending list (accessed on 6/Dec/2022) under three different languages: Java[3], JavaScript[4], and TypeScript[5], we selected the top four complete applications, that are not a framework, an add-on, or a tutorial: Rocket Chat[6], Telegram[7], Odoo[8], and Joplin[9]. We now refer to them as validation applications.

We display the time taken to complete the identification task on validation applications and the number of identified flows in Table 5.

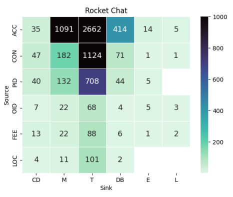
Table 5: Identified code snippet count and the time consumed for each application

	Time	**Code Snippet Count**
Rocket Chat (TypeScript)	397 s	6,935
Telegram (Java)	562 s	16,963
Odoo (JavaScript)	916 s	25,653
Joplin (TypeScript/JS)	694 s	17,299

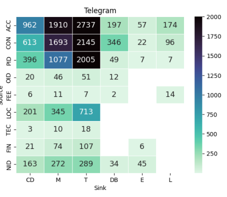
For large projects such as Odoo and Telegram, our analysis takes about 15 minutes to analyze 2,285 files (Odoo) using 70 static analysis rules. Odoo is the business content management system with the highest number of identified personal data flows among the four validation applications, indicating that it processes a substantial amount of personal data.

Next, we validate our approach in two steps. First, we manually examine the precision of identified code snippets, assessing whether they are related to personal data processing. Figure 4 and

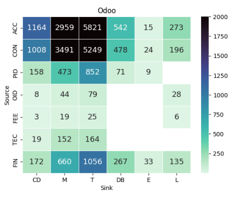
[3] `https://github.com/trending/java?since=monthly`
[4] `https://github.com/trending/javascript?since=monthly`
[5] `https://github.com/trending/typescript?since=monthly`
[6] `https://github.com/RocketChat/Rocket.Chat`
[7] `https://github.com/DrKLO/Telegram`
[8] `https://github.com/odoo/odoo`
[9] `https://github.com/laurent22/joplin`



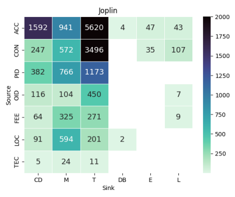
(a) Rocket Chat



(b) Telegram



(c) Odoo



(d) Joplin

Fig. 4: The overview statistics of identified flow under all possible combinations of source and sink types as illustrated in heatmaps.

a Personal Data Type View figure (akin to Figure 3) form the initial perspective of our generated views. The four heatmaps present overview statistics of the different categories of personal data and its processing in the four validation applications. The overview statistics indicate that all four applications collect a significant quantity of account and contact information, which is understandable as they function as identifiers for system users and communication mediums. However, we observe that certain applications gather more categories of personal data from users, such as Telegram, which collects nine categories of personal data, with the majority stored in databases.

Table 6 summarizes the results of the code snippet detection in each example application, as well as the precision calculated through manual inspection for each identified flow provided in the Detailed Flow View. Given that this evaluation requires manual inspection and we need to avoid false negatives, we examine the code snippets identified by the static analysis using the following criteria to determine their precision score[10]:

– Do the source and sink identifiers match their respective categories? For instance, if the analysis intended to match "`log`" but instead matched "`login`", which is irrelevant to the context.
– Do the source and sink matches qualify as personal data processing? For instance, "`noreply@test.org`" was detected as a clear-text result for explicit personal data in the system, although it is not personal data.

Our main objective is to lessen the manual workload for code reviewers in identifying and analyzing personal data flows, though we recognize that manual scrutiny continues to play a crucial role. A high level of precision in our flow identification technique indicates that we can conserve resources by directing code reviewers to potential areas of concern, and offering guidance on the potential data flow paths and locations within their applications. This aligns with our specialized views approach, enhancing the efficiency and efficacy of GDPR compliance tasks.

For the manual analysis, we conducted a thorough evaluation of a representative sample of the identified code snippets to estimate the precision, rather than individually checking all code snippets identified. Table 6 shows that the majority of flow types have a precision of at least 0.8, with an average of over 0.9 for categories with more than 500 identified flows. Some categories of flows that have a smaller sample size have a lower precision, ranging from 0.65 to 0.75. Although a precision of 0.8 might result in 5K false positives for an application with 25K identified snippets (e.g., Odoo), our approach aims to assist developers and code reviewers in finding possible directions and locations for personal data processing, thereby reducing the overall time and effort spent on manual analysis.

Note that we do not include the number of occurrences of literal personal data identified by the pattern matching, as their precision is typically less than 0.6. A precision of less than 0.6 indicates that developers and code reviewers still need to devote considerable effort to exclude

[10] Precision = TP/(TP+FP), where True Positives (TP) are identified results that meet both of our criteria, while False Positives (FP) are identified results that do not meet one of our criteria.

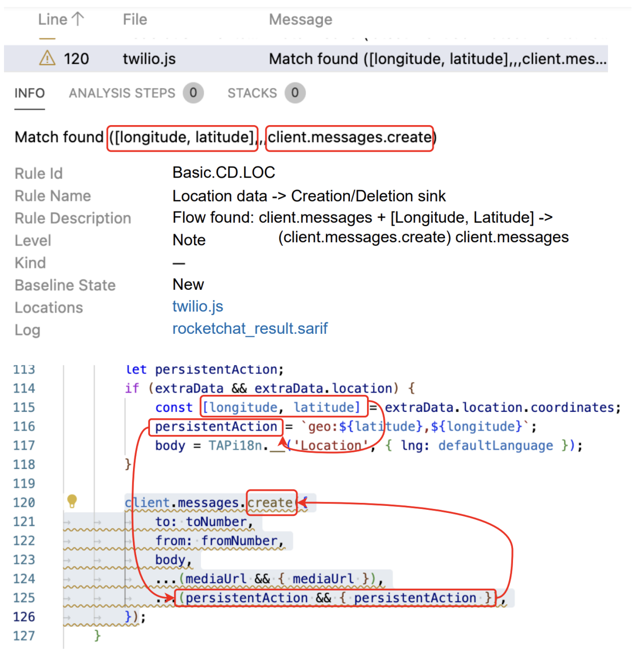
Table 6: Statistics showing the precision of different types of flows detected. '-' marks the labels for which our approach detected less than 20 results. Sink types are: *data creation/deletion* (**C/D**), *data manipulation* (**M**), *data transportation* (**T**), *database* (**DB**), *encryption* (**E**) and *log* (**L**). Source types are: *account* (**ACC**), *contact* (**CON**), *personal ID* (**PID**), *online identifier* (**OID**), *location* (**LOC**), *feedback* (**FEE**), *health* (**HEA**), *national ID* (**NID**), *technical* (**TEC**), and *financial* (**FIN**).

Application	**Source**	**Basic Sink**			**Special Sink**		
		C/D	**M**	**T**	**DB**	**E**	**L**
Rocket Chat	ACC	0.80	0.95	0.94	0.97	-	-
	CON	0.85	0.88	0.95	0.96	-	-
	PID	0.83	0.92	0.95	0.96	-	-
	OID	-	0.73	0.88	-	-	-
	FEE	-	0.82	0.91	-	-	-
	LOC	-	-	0.89	-	-	-
Joplin	ACC	0.88	0.93	0.96	-	0.89	0.95
	CON	0.93	0.81	0.92	-	0.92	0.98
	PID	0.81	0.89	0.95	-	-	-
	OID	0.79	0.87	0.84	-	-	-
	FEE	0.84	0.95	0.8	-	-	-
	LOC	0.91	0.86	0.89	-	-	-
	TEC	-	0.71	-	-	-	-
Telegram	ACC	0.89	0.92	0.96	0.87	0.86	0.96
	CON	0.90	0.96	0.94	0.93	0.91	0.92
	PID	0.84	0.93	0.89	0.86	-	-
	OID	0.70	0.73	0.81	-	-	-
	FEE	-	-	-	-	-	-
	LOC	0.92	0.90	0.95	-	-	-
	TEC	-	-	-	-	-	-
	FIN	-	0.94	0.97	-	-	-
	NID	0.86	0.96	0.94	0.88	0.91	-
Odoo	ACC	0.91	0.86	0.98	0.94	-	0.95
	CON	0.84	0.87	0.93	0.92	0.92	0.96
	PID	0.82	0.95	0.97	0.96	-	-
	OID	-	0.66	0.63	-	-	0.82
	FEE	-	-	0.72	-	-	-
	TEC	-	0.89	0.95	-	-	-
	FIN	0.88	0.92	0.94	0.95	0.90	0.95

literal identification results that are not relevant. Identifying literal personal data is difficult due to its ambiguous and highly contextual nature.

Next, we leverage the attributes available in the detailed Detailed Flow View to evaluate existing privacy statements. Figure 5 depicts an instance of a flow identified through our Detailed Flow View (similar to Table 4). This flow was identified by a rule pinpointing the flow of location data into a transportation sink. When a flow is selected for deeper scrutiny, crucial information such as the file location, rule type and name, abstract flow pattern (displayed under "Rule Description"), source/sink identifiers, and the original code are readily accessible.

By integrating the Detailed Flow View with the provided statistics and heatmaps from the Personal Data Type View, we can guide ROPA development and verify the accuracy of existing privacy statements. For mobile applications, we additionally examine the accuracy of personal data processing disclosures made to platforms like Google Play or Apple's App Store. In situations where a comprehensive privacy statement is absent, such as with Joplin[11], we illustrate how our views can be employed for ROPA creation.



Line ↑	File	Message
⚠ 120	twilio.js	Match found ([longitude, latitude],,,client.mes...

INFO ANALYSIS STEPS 0 STACKS 0

Match found ([longitude, latitude],,client.messages.create)

Rule Id	Basic.CD.LOC
Rule Name	Location data -> Creation/Deletion sink
Rule Description	Flow found: client.messages + [Longitude, Latitude] -> (client.messages.create) client.messages
Level	Note
Kind	—
Baseline State	New
Locations	twilio.js
Log	rocketchat_result.sarif

```
let persistentAction;
if (extraData && extraData.location) {
    const [longitude, latitude] = extraData.location.coordinates;
    persistentAction = `geo:${latitude},${longitude}`;
    body = TAPi18n.__('Location', { lng: defaultLanguage });
}

client.messages.create({
    to: toNumber,
    from: fromNumber,
    body,
    ...(mediaUrl && { mediaUrl }),
    ...(persistentAction && { persistentAction }),
});
}
```

Fig. 5: Example of an identified flow in the Detailed Flow View for Rocket Chat

In order to establish a connection between ROPA requirements and our specialized views, we reference Table 8 which addresses four critical requirements outlined in Section 2.1. These can subsequently be incorporated into a comprehensive ROPA. Furthermore, we juxtapose our results with the published privacy statements of the chosen four applications.

Rocket Chat's detailed and well-structured privacy statement[12] aligns closely with our Personal Data Type View and Detailed Flow View. Telegram and Odoo also offer comprehensive and lucid privacy policies. However, our experiment uncovered that Telegram collects user feedback and financial data, an activity not explicitly mentioned in their privacy statement. On the other hand, Joplin's privacy statement[13] is brief and lacks specificity on the types of personal data collected, except for geolocation data. We also discovered instances where some personal data is temporarily stored without being disclosed in the statement.

Our approach demonstrated its effectiveness in covering personal data and its processing in the selected validation applications, as compared to their published privacy statements. Nonetheless, we also found instances where the applications collected personal data beyond their disclosed policies. Personal data logging is a sensitive matter and should be meticulously inspected and documented.

[11] https://joplinapp.org/privacy/
[12] https://docs.rocket.chat/legal/privacy
[13] https://joplinapp.org/privacy/

Table 7: Identified personal data types, their quantity, and how they are covered by Google Play's privacy policies. In column 'Coverage', '-' indicates the types of personal data identified by our approach but not covered in the privacy statement; '+' implies that there is coverage of all identified personal data types in the privacy statement.

Android Apps	**ACC**	**CON**	**PID**	**OID**	**LOC**	**FEE**	**HEA**	**NID**	**TEC**	**FIN**	**Coverage**	**Logging**
Discord	1621	791	31	399	159	107	-	-	87	9	+	41
Kik	2065	1923	17	817	51	39	-	-	46	31	-ACC, -LOC	133
Viber	1823	2093	101	740	219	42	-	-	71	16	-ACC, -OID, -FEE, -TEC	82
Amazon	905	2762	1025	1623	634	198	42	-	231	129	-ACC	162
AliExpress	1244	2381	1730	2094	841	217	-	-	309	261	-LOC, -FEE	217
Shein	1072	1967	965	978	409	96	-	-	160	373	-LOC	62
The Weather Channel	239	2384	1291	1328	2623	172	-	-	16	22	+	76
AccuWeather	176	2137	1102	1736	3321	98	-	-	5	9	-PID, -OID, -ACC, -CON	41
Windy	312	246	567	803	1196	-	-	-	-	14	-LOC	22
SamSung Health	273	182	31	98	685	44	1027	-	28	21	+	10
Google Fit	105	49	14	6	271	19	580	-	9	4	+	7
My Fitness Pal	328	209	19	113	902	75	721	-	47	69	-PID	89
Uber	1497	1054	298	835	2094	511	-	-	814	427	+	294
Trainline	469	107	244	54	626		-	4	82	562	-NID	104
Omio	724	151	457	72	729		-	12	205	710	-ACC, -NID, -OID	71

4.2 Application: Assessing Google Play Data Safety Statements

To further highlight the applicability of our approach, we examined the top three Android applications from Google Play ([14]) in five different categories: communication, online shopping, weather, fitness/health, and transportation. As the decompilation process generated some noise, potentially distorting the count of data processing flows, we focused on assessing the categories and distributions of personal data collected by the applications.

In our evaluation, we verified the "Data Safety" statements of 15 applications against the Personal Data Type View and Detailed Flow View generated by our approach. Table 7 presents the results, including the number of identified flows for each personal data category, the coverage of their data safety statements in Google Play, and the amount of personal data logged.

We found that our approach detected certain types of personal data not disclosed in the Google Play data safety section. While Google allows exceptions for "on-device access/processing" and "end-to-end encryption", we found instances where data left the device without being disclosed. For example, location data in the online chatting app Kik and national ID data in the transportation app Omio are transmitted outside the device, but this was not stated in their respective Google Play disclosures.

Interestingly, we observed that applications across various domains commonly collected account and contact data, with additional domain-specific types. For instance, weather applications gathered significant location data, while transportation/ticketing applications acquired national ID data from users. Consistent with the GitHub projects evaluated earlier, all the apps logged personal data. This assessment underscores the utility of our approach in helping apps improve the accuracy of their data safety statements and increase transparency in their data handling practices.

4.3 Threats to Validity

Our approach does not offer a fully automated solution for GDPR compliance tasks, such as generating a ROPA, due to the lack of a natural language processing module. This limitation makes it challenging to directly generate or verify statements based on the output of our approach, which consists of code snippets, fragments, labels, and additional details. As a result, manual effort is required in our experiments, which consequently restricts the number of applications we

[14] `https://play.google.com/store/apps?gl=US`, accessed on 8/Dec/2022, from the U.S. store

Table 8: Fact-check the published privacy statements of the four applications using the information supplied by our two specialized views. The missing information is **highlighted**.

App	Data Captured via Views Aligned with ROPA	Published Privacy Statement/Policy
Rocket Chat	– Categories of personal data: ACC, CON, PID, OID, FEE, LOC. – Categories of processing: basic processing: C/D, M, and T. Minor logging personal data identified. – Transfer to a database or third-party APIs: own database access identified on ACC, CON, and PID data, minor on OID, FEE, and LOC data, no third-party database API detected. – Data encryption or anonymization: Encryption on ACC, CON, PID, OID, FEE data. No anonymization was identified.	– Categories of personal data: personal ID (PID, CON), account data (ACC), usage data (OID, FEE), location data (LOC), cookie data (OID) – Categories of processing: for contact/identification: PID, CON; for market/communication: PID, CON; for registration: ACC; for maintenance/tech support/monitoring: OID, FEE; for functionalities: OID, LOC, OID. – Transfer to a database or third-party APIs: no third-party services mentioned, data outside of the USA might be transferred to services in the USA. – Data encryption or anonymization: relevant security measures were taken into account.
Telegram	– Categories of personal data: ACC, CON, PID, OID, **FEE**, LOC, TEC, **FIN**, NID – Categories of processing: basic processing: C/D, M, and T. Major logging ACC and CON data identified. – Transfer to a database or third-party APIs: there are database calls for ACC, CON, PID, and NID data identified to both internal and external databases. – Data encryption or anonymization: there is major encryption on NID, ACC, and CON data.	– Categories of personal data: account data (ACC, PID), contact data (CON, ACC, NID), location data (LOC), chats (OID), and cookies (TEC, OID) – Categories of processing: for identification/account purposes (ACC, PID, NID); for communication (CON, ACC); for improving services (TEC, OID); for functionalities (LOC, TEC, OID). – Transfer to a database or third-party APIs: data is saved in third-party provided data centers in the Netherlands for European users, and end-to-end chats are not transmitted out of the device – Data encryption or anonymization: "All data is stored heavily encrypted so that local Telegram engineers or physical intruders cannot get access."
Odoo	– Categories of personal data: ACC, CON, PID, OID, FEE, TEC, FIN – Categories of processing: basic processing: C/D, M, and T. Major logging ACC, CON, and FIN data identified. – Transfer to a database or third-party APIs: there are database calls for ACC, CON, PID, and FIN data identified to both internal and external databases. – Data encryption or anonymization: there is encryption on ACC, CON, PID, and FIN data.	– Categories of personal data: account & contact Data (ACC, CON), job application data (CON, PID), browser data (FEE, TEC), customer databases (ACC, CON, PID, FIN), free trial session recording (FIN, TEC, OID, FEE), In-App Purchase transaction data (FIN) – Categories of processing: for the recruitment process (ACC, CON, PID); for maintaining and improving services (FEE, TEC, OID); for providing services (ACC, PID), answering requests (CON), and for billing management (FIN, CON, ACC) – Transfer to a database or third-party APIs: "customer databases are hosted in the Odoo Cloud Region closest to where they are based, and can request a change of region" – Data encryption or anonymization: "info is securely processed, stored and preserved from data loss and unauthorized access".
Joplin	– Categories of personal data: **ACC**, **CON**, **PID**, **OID**, **FEE**, LOC, **TEC** – Categories of processing: basic processing: C/D, M, and T. Major logging contact data identified. – Transfer to a database or third-party APIs: Almost none, only less than 5 identified for ACC and LOC data to be passed into a local temporary data model. – Data encryption or anonymization: There are encryption measures on ACC and CON data identified.	– Categories of personal data: It is not mentioned in the privacy policy, only explicitly mentioned geo-location data. – Categories of processing: Not mentioned. – Transfer to a database or third-party APIs: Only mentioned: "Any data that Joplin saves, such as notes or images, are saved to your own device and you are free to delete this data at any time." – Data encryption or anonymization: Not mentioned.

can feasibly validate (four in this study). Each application must be open-source, widely used, gather diverse types of personal data, and possess a publicly available privacy statement.

The primary threat to the validity of our experiments is the difficulty in establishing a ground-truth set of actual data processing activities within the source code. This is due to the necessity of domain knowledge from the original development team, which is typically inaccessible. Consequently, it is not feasible to accurately measure the recall value (TP/(TP+FN)) for our approach.

In the context of fact-checking Google Play data safety statements, we are unable to verify which types of personal data are sent outside of the device. This limitation prevents us from fully assessing the accuracy of these statements.

5 Related Work

Existing research on incorporating privacy-by-design (PbD) into the software development life cycle lacks concrete tools to assist software developers in designing and implementing GDPR-compliant systems [3]. Moreover, such PbD-created systems lack standards for mapping particular legislative data protection obligations, such as the GDPR.

Ferrara et al. propose a framework to adopt static analysis to assess GDPR compliance [4]. However, this approach requires compiled bytecode, which may not always be readily available during the development process. Arfelt et al. provide a formal logic for monitoring GDPR compliance [2] and a functional tool was developed by [6] for analyzing cross-border data transmission in Android applications, which is limited in scope to a specific platform.

Some existing research focuses on the identification of personal data but does not consider broader GDPR compliance aspects. Fugkeaw et al. [5] design a technique to let enterprises automatically detect and handle personal data stored in the local file system. ReCon [12] uses machine learning to detect possible breaches of personal data by monitoring network traffic, requiring a pre-trained ML model. van der Plas [11] identifies personal data in Git commits using CodeBERT, a transformer model similar to RoBERT, but this approach only focuses on the presence of personal data.

Our work aims to address these research gaps by providing a comprehensive solution that does not rely on compiled bytecode, pre-trained ML models, or platform-specific tools, and goes beyond merely identifying the presence of personal data to ensure broader GDPR compliance during the software development process.

6 Conclusion and Future Directions

Ensuring GDPR compliance demands detailed information on personal data processing, often requiring significant manual effort. Our work strives to lessen this burden by offering two specialized views — Personal Data Type View and Detailed Flow View, facilitating code reviewers in identifying potential data processing locations and providing necessary information. Our approach, leveraging static analysis, has shown an average precision of 0.87 in our experiments, demonstrating its effectiveness.

However, our approach has limitations. Currently, it is based on Semgrep for static analysis, which captures intra-procedural data flows, leaving inter-procedural flows unaccounted for. The adoption of the Semgrep Pro Engine, offering inter-procedural analysis, could enhance our approach's precision.

In conclusion, our approach presents specialized views, aiding in GDPR compliance tasks, such as ROPA production, by reducing manual effort. With further improvements, such as incorporating the Semgrep Pro Engine and refining our identification rules, we aim to make the process of privacy analysis more efficient and manageable for code reviewers.

Acknowledgement

This paper is an extended version of work published in [15]. We would like to extend our sincere gratitude to Rob van der Veer for his valuable insights and contributions to this research. This work is part of the Privacy Matters (PriMa) project. The PriMa project has received funding from European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 860315.

References

1. Alharthi, A., Krotov, V., Bowman, M.: Addressing barriers to big data. Business Horizons **60**(3), 285–292 (2017)
2. Arfelt, E., Basin, D., Debois, S.: Monitoring the GDPR. In: European Symposium on Research in Computer Security. Springer (2019)
3. Baldassarre, M.T., Barletta, V.S., Caivano, D., Scalera, M.: Integrating security and privacy in software development. Software Quality Journal **28**(3), 987–1018 (2020)
4. Ferrara, P., Spoto, F.: Static analysis for gdpr compliance. In: Italian Conference on Cybersecurity (2018)
5. Fugkeaw, S., Chaturasrivilai, A., Tasungnoen, P., Techaudomthaworn, W.: Ap2i: Adaptive PII scanning and consent discovery system. In: 2021 13th International Conference on Knowledge and Smart Technology (KST). pp. 231–236. IEEE (2021)
6. Guamán, D.S., Del Alamo, J.M., Caiza, J.C.: Gdpr compliance assessment for cross-border personal data transfers in android apps. IEEE Access **9**, 15961–15982 (2021)
7. Huth, D., Tanakol, A., Matthes, F.: Using enterprise architecture models for creating the record of processing activities (Art. 30 GDPR). In: 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC). pp. 98–104. IEEE (2019)
8. ICO: How do we document our processing activities? information commissioner's office. `https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/documentation/how-do-we-document-our-processing-activities/`, (Accessed on 04/29/2023)
9. Naik, A., Mendelson, J., Sands, N., Wang, Y., Naik, M., Raghothaman, M.: Sporq: An interactive environment for exploring code using query-by-example. In: The 34th Annual ACM Symposium on User Interface Software and Technology. pp. 84–99 (2021)
10. Pandit, H.J., Polleres, A., Bos, B., Brennan, R., Bruegger, B., Ekaputra, F.J., Fernández, J.D., Hamed, R.G., Kiesling, E., Lizar, M., et al.: Creating a vocabulary for data privacy. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 714–730. Springer (2019)
11. van der Plas, N.: Detecting PII in Git commits. Master's thesis - TU Delft (2022)
12. Ren, J., Rao, A., Lindorfer, M., Legout, A., Choffnes, D.: Recon: Revealing and controlling PII leaks in mobile network traffic. In: Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services. pp. 361–374 (2016)
13. Ryan, P., Pandit, H.J., Brennan, R.: A common semantic model of the gdpr register of processing activities. arXiv:2102.00980 (2021)
14. Solove, D.J.: Access and aggregation: Public records, privacy and the constitution. Minn. L. Rev. **86**, 1137 (2001)
15. Tang., F., Østvold., B., Bruntink., M.: Identifying personal data processing for code review. In: Proceedings of the 9th International Conference on Information Systems Security and Privacy - ICISSP. pp. 568–575. INSTICC, SciTePress (2023). `https://doi.org/10.5220/0011725700003405`

PAPER 5

Transparency in App Analytics: Analyzing the Collection of User Interaction Data

Tang, F. and Østvold, B. (2023). Transparency in App Analytics: Analyzing the Collection of User Interaction Data. In Proceedings of the 20th Annual International Conference on Privacy, Security & Trust (PST2023).

Transparency in App Analytics: Analyzing the Collection of User Interaction Data[⋆]

Feiyang Tang and Bjarte M. Østvold

Norwegian Computing Center, Oslo, Norway
{feiyang,bjarte}@nr.no

Abstract. The rise of mobile apps has brought greater convenience and many options for users. However, many apps use analytics services to collect a wide range of user interaction data, with privacy policies often failing to reveal the types of interaction data collected or the extent of the data collection practices. This lack of transparency potentially breaches data protection laws and also undermines user trust. We conducted an analysis of the top 20 analytic libraries for Android apps to identify common practices of interaction data collection and used this information to develop a standardized *collection claim* template for summarizing an app's data collection practices wrt. user interaction data. We selected the top 100 apps from popular categories on Google Play and used automatic static analysis to extract *collection evidence* from their data collection implementations. Our analysis found that a significant majority of these apps actively collected interaction data from UI types such as View (89%), Button (76%), and Textfield (63%), highlighting the pervasiveness of user interaction data collection. By comparing the collection evidence to the claims derived from privacy policy analysis, we manually fact-checked the completeness and accuracy of these claims for the top 10 apps. We found that, except for one app, they all failed to declare all types of interaction data they collect and did not specify some of the collection techniques used.

Keywords: Mobile apps · User interaction data collection · Transparency · Trust · Privacy.

1 Introduction

The rapid rise of mobile apps has revolutionized how we interact with technology, providing developers with a treasure trove of user interaction data through analytics services such as AppsFlyer [1], Flurry [2], and Firebase Analytics [3]. This data, encompassing user actions like button taps, page scrolls, and video views, is invaluable for enhancing app functionality and user experience. However, the vague terminology often used in privacy policies, such as "user's interaction with the service", raises concerns about transparency. The lack of specificity leaves users uncertain about the extent and nature of the data being collected and its usage, potentially leading to mistrust and diminished app usage.

Transparency in data collection is a crucial factor influencing user trust [8]. It empowers users to make informed decisions about the data they share and its intended usage [19].

An example of this ambiguity can be found in the Yr app, Norway's most popular weather app developed by the Norwegian Broadcasting Corporation (NRK). Despite collecting user interaction data to understand commonly used features, the app's privacy policy[4] is vague regarding the collection of such data, as quoted below in the blue box. Our examination of NRK's privacy policy revealed no explicit information about Yr's data collection practices, leading to concerns about user trust in both the app and NRK.

⋆ Published at *the 20th Annual International Conference on Privacy, Security & Trust (PST2023)*
[1] https://www.appsflyer.com/
[2] https://www.flurry.com/
[3] https://firebase.google.com/docs/analytics
[4] https://hjelp.yr.no/hc/en-us/articles/360003337614-Privacy-policy

Analyze tools
"We use different tools to track the use on our app and website. This information gives us valuable information such as most popular pages and on what times Yr is being used the most. No information that can identify persons are available for Yr."

Our examination of NRK's privacy policy[5] revealed no specific information regarding Yr's data collection practices. The policy mainly focuses on NRK's news services and their "interaction with the services" collection practices. This obscurity concerning Yr app's data collection practices raises concerns, as it might undermine user trust in both the app and NRK as a whole.

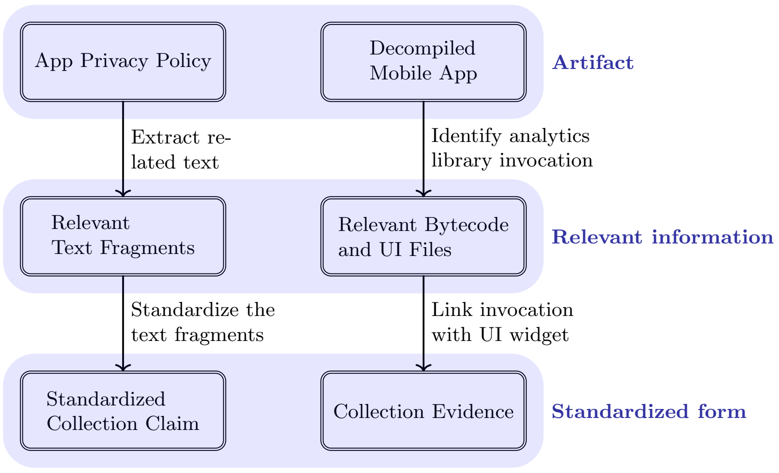

App Privacy Policy
Decompiled Mobile App
Artifact
Extract related text
Identify analytics library invocation
Relevant Text Fragments
Relevant Bytecode and UI Files
Relevant information
Standardize the text fragments
Link invocation with UI widget
Standardized Collection Claim
Collection Evidence
Standardized form

Fig. 1. Overview of the approach for analyzing collection claims and evidence in apps.

Recent research has shown that even seemingly harmless user interaction data can reveal sensitive information about individuals. For instance, data like emoji usage or pages visited can be used to infer a user's pool preference or political orientation [13]. Moreover, mobile biometric data related to keystrokes and touchscreen gestures can help estimate attributes like age, gender, and operating hand [7, 15]. This underscores the potential risks associated with user interaction data collection, which, while not typically considered personal data, can be utilized to deduce sensitive information about individual users, leading to user profiling. The lack of transparency in these data collection practices could potentially erode user trust in the app.

Most current research on the privacy implications of analytic services has concentrated on determining whether personally identifiable information (PII) is being collected and transmitted to external analytics services [17, 28]. Studies have also scrutinized log data to understand user behavior [10], and high-level analyses of user behavior data collection in mobile apps have been performed [26]. It is essential to clarify that the interaction data discussed in this study is not part of traditionally defined PII or personal data, emphasizing the need for better transparency in data collection practices.

[5] https://info-nrk-no.translate.goog/personvernerklaering/?_x_tr_sl=no&_x_tr_tl=en&_x_tr_hl=en-US&_x_tr_pto=wapp

1.1 Objective

The aim of this paper is to address the issue of lack of transparency in the collection of user interaction data in mobile apps. To achieve this, we propose a standardized collection claim template that can be compared to collection evidence determined through static analysis.

1.2 Research Questions

To guide our investigation, we formulated the following research questions:

RQ1 What are the common practices of user interaction data collection in mobile apps?
RQ2 How are these practices reflected in apps' privacy policies?
RQ3 What types of user interaction data do apps actually collect in their implementations, and how is this data collected?
RQ4 To what extent do the collection claims in privacy policies align with the actual data collection practices as observed in their implementations?

1.3 Contributions

In this paper, we make several contributions towards understanding and promoting transparency in user interaction data collection practices:

- We propose a *standardized collection claim template* for user interaction data. *Collection claims*, in this context, refer to the descriptions of common practices of user interaction data collection in mobile apps, as stated in privacy policies. The template reflects common phrasing and vocabulary derived from Android documentation and popular Android analytic libraries (Section 3).
- We present an automatic static analysis method to identify *collection evidence* from Android apps, which involves analyzing data types, relevant code, and techniques of collection in layout files and bytecode (Section 4).
- We provide an overview of user interaction data collection practices in the top 100 popular apps on Google Play across the top 10 categories (Section 5). Our analysis reveals common patterns and offers useful statistics for both app developers and users to better understand the current state of data collection practices in mobile apps.
- We conduct *fact-checking* by manually comparing privacy policy collection claims against the actual collection evidence found in the ten most popular apps from the top ten categories (Section 5). Our findings reveal that none of these apps were completely accurate and complete in their collection claims, highlighting the importance of our proposed approach in promoting transparency and trust in user interaction data collection practices.

We believe that our proposed method addresses the problem of lack of transparency by providing a standardized collection claim template for describing user interaction data collection practices. The template allows app developers to offer clearer and more accurate information about their data collection practices, which in turn can help users make informed decisions about app usage and data sharing. Our method also enables researchers and app developers to assess the alignment between stated data collection practices in privacy policies and the actual practices found in app implementations, facilitating better transparency and ultimately enhancing user trust.

Fig. 1 provides a high-level overview of our method, illustrating the process of acquiring artifacts (privacy policies and decompiled mobile apps), deriving knowledge through text analysis and static analysis (relevant text fragments and bytecode/UI files), and standardizing the information into collection claims and evidence. This systematic and transparent approach can contribute to promoting trust and fostering greater transparency in user interaction data collection practices in mobile apps.

2 Motivation

The transparency of mobile app data collection practices is a critical issue that stems from several significant factors, all of which play an essential role in the complex relationship between user trust and app adoption. Interaction data, a key asset for understanding user behavior, can raise serious concerns if it is collected non-transparently.

Transparency serves a dual purpose in this scenario. Firstly, it acts as an ethical commitment, assuring users that they are informed about their interaction data's collection and use. This principle not only respects user autonomy but also fosters an environment of openness and accountability. Secondly, transparency plays a pivotal role in building user trust, a significant factor influencing user satisfaction and continued app usage. When users understand and control their interaction data, their trust in the app increases, leading to more consistent engagement. Conversely, a lack of transparency can breed mistrust and privacy concerns, potentially causing user dissatisfaction or even app abandonment.

The correlation between transparency and user trust is well-documented in the academic world. Studies have consistently highlighted the positive relationship between increased transparency and elevated levels of user trust [12]. Conversely, an absence of transparency can obstruct the success and widespread adoption of mobile apps [27]. Thus, transparency is not merely about informing users, but is essential for facilitating informed decisions and granting consent.

Another critical aspect to consider is the rise of analytics services like Firebase Analytics and Flurry Analytics. These services provide developers with tools to gather data on user behavior, engagement, and preferences. However, they have raised data protection concerns as they often automatically collect user data, thereby creating privacy issues. Several countries, including France, Italy, Austria, Denmark, and Norway, have explicitly stated that the use of Google Analytics violates GDPR [23]. Android apps can utilize these services either by directly invoking third-party APIs or by customizing their analytics service by extending these APIs. The first approach involves calling third-party API methods directly in activities to log user engagement events. The second approach allows developers to tailor data collection to their specific needs.

The importance of transparency is also acknowledged by mobile app developers, who have a vested interest in prioritizing it alongside user control in their data collection practices. Research supports this notion; for instance, Almuhimedi et al. [1], for instance, discovered that many smartphone users are not fully aware of the data collected by their apps. Providing users with an app permission manager and sending notifications to increase their awareness of data collection can enable them to better manage their privacy. Moreover, users' concerns about data collection can negatively impact their perception of the app, potentially leading to its uninstallation [9]. Hence, promoting transparency and user control can foster trust, resulting in improved user experiences, increased app engagement, and higher adoption rates.

In the following sections, we will explore the mutual benefits of transparency for both users and app developers and illustrate how our proposed method can address the current shortcomings in the transparency of interaction data collection practices.

3 Standardizing Data Collection Claims

To address **RQ1**, we analyze the common practices of user interaction data collection in mobile apps, specifically the types of data collected and the techniques of collection. To achieve this, we conducted an analysis of the top 20 analytic libraries for Android apps. To answer **RQ2**, we examine how these practices are reflected in the privacy policies of mobile apps. We refer to the descriptions of data collection practices in privacy policies as *collection claims*, which we define as a single sentence in a standardized template using a restricted vocabulary to convey the essence of interaction data collection.

Table 1. The top five most frequent terms used to describe user interaction data in the APP-350 corpus

Term	Count
interact(∼ion,∼ing) with service/app	1,049
analytic(s)	886
us(∼age, ∼ing) of service/app	397
statistic(s)	315
input(s) of user	173

Table 2. The top five most frequent verbs used to describe such collection in the APP-350 corpus

Verb	Count
collect	1,386
track	548
use	202
log	86
gather	46

3.1 Collection Vocabulary

Our restricted collection vocabulary was developed by analyzing the Android system implementation documentation, as well as the APIs of the top 20 analytic services for Android apps listed on AppBrain [2].

Terms for Types of User Interaction Data The user interface of an Android app collects a variety of data types, such as touch events, sensor data, and text input. Based on a manual inspection of every single type of Android UI widget, we identified the following six types of interaction data and named them:

– *App presentation data*: This data arise from the consumption of content provided by the app. For example, the user plays a certain video for a period of time, spends minutes reading one specific page of the news. These interactions are often recorded by a logging system to keep track of the user's consumption habits.
– *Binary data*: This data arise from discrete user actions, such as tapping on a button or icon, or selecting a checkbox.
– *Categorical data*: This data arise from a selection from a set of predefined options or categories, such as choosing a value from a dropdown menu, selecting a radio button, or rating a product.
– *User input data*: This data arise from user input through an on-screen keyboard or another input method, such as entering text or numbers into a form field, or using voice input to perform a search or command.
– *Gesture data*: This data arise from gesture inputs and smooth and continuous movements of the user's finger on the screen, such as scrolling through a list, swiping left or right, pinching or zooming, or shaking the device.
– *Composite gestures data*: This data arise from a combination of multiple gestures, such as tapping and holding, double tag, or drag and drop.

Terms for Collection Techniques We use the following terms to describe the techniques of user interaction data collection.

– *Frequency*: This technique involves logging the frequency of the occurrence of a particular interaction. For example, an app might log the number of times a user taps on a specific button or selects a certain option from a drop-down menu.
– *Duration*: This technique involves tracking the time a user spends engaging in a particular interaction. For example, an app might log the amount of time a user spends watching a particular video or reading a specific article.
– *Motion details*: This technique involves monitoring the specific details of a user's interaction, such as the speed, direction, or angle of their finger movements on the screen. This type of data can be collected for interactions such as scrolling, swiping, or dragging.

3.2 From Policies to Standardized Collection Claims

In this section, we study privacy policies of publicly accessible mobile apps, aiming to identify and standardize collection claims related to user interaction data. Utilizing the APP-350 Corpus, a pre-trained language model, and manual checks, we extract and validate common terminologies employed in these policies. This comprehensive process allows us to establish a standardized vocabulary for user interaction data collection claims, providing a solid foundation for subsequent analysis.

Identifying Relevant Policy Parts To distinguish sentences related to user interaction data collection in privacy policies, we adopt a simple pre-trained language model. This model sifts through HTML files of privacy policies and singles out sentences containing specific keywords and their synonyms. Our focus is to ensure that our privacy policy claim vocabulary aligns with the most common terminology used in the industry to describe user interaction data.

After processing the privacy policies, we conduct manual checks to eliminate any false positives. The end result is a selection of common phrases used in these policies to describe the collection of user interaction data. From the sentences identified, we isolate the most relevant verbs and nouns to form a list of keywords.

Experimental Details and Validation In conducting our analysis, we use the APP-350 Corpus [24], a set of 350 mobile app privacy policies that are annotated with privacy practices. Although the main focus of the APP-350 Corpus is on identifying sentences related to personally identifiable information (PII), we utilize the raw HTML files of the privacy policies for our examination.

The natural language processing is carried out using the spaCy [14] library with the `en_core_web_sm` model. This model, pre-trained on web text, which includes web forums, web pages, and Wikipedia, is capable of identifying named entities, parts of speech, dependency parsing, and more. We also employ the WordNet module from the Natural Language Toolkit (NLTK [6]) to discover synonyms for the extracted keywords.

To authenticate the effectiveness of the model, we manually annotate 50 randomly selected privacy policies from the APP-350 dataset. This helps us identify sentences containing relevant information, the verbs used to describe data collection (e.g., “collect”, “track”), and the terms used to describe user interaction data (e.g., “usage of the app”, “interaction with the service”).

The model successfully recognized sentences related to user interaction data collection in 37 out of the 38 files that contained such sentences, using keywords such as interaction, usage, statistics, experience, and analytics. Identifying the verbs used to describe data collection was a more complex task, with a recall of 92% but a precision of only 84% [6] due to the presence of similar verbs in sentences that were not related to the context.

Upon running the model on the 350 privacy policies, we identified 1,411 sentences. The relevant verbs and nouns from these sentences are shown in Table 1 and Table 2 and then compiled to form the list of keywords.

Template for Standardized Collection Claims In privacy policies, it is common for apps to use convoluted language to describe how user data is collected. To make these collection claims in privacy policy easier to read and compare across different apps, we created a standardized template that utilized the most frequently used verb, “collect”, and the most frequently used noun phrase, “user interaction data”. The resulting structure is as follows:

[6] Recall is calculated as TP / (TP + FN), while precision is calculated as TP / (TP + FP), where TP is true positives, FN is false negatives, FP is false positives, and TN is true negatives.

Template for Standardized Collection Claims
We collect the following types of user interaction data: ⟨*types of data collected*⟩, along with their ⟨*techniques of collection*⟩.[7]

This standardized collection claim template can be combined with the collection evidence gathered through static analysis to check and the accuracy of privacy policy collection claims made by various apps. Also, the standardized language facilitates transparency and comparison between policies. We return to the subject of fact-checking collection claims in Section 5.

4 Data Collection Evidence

In this section, we analyze mobile apps to understand the types of user interaction data collected and the techniques employed (**RQ3**). We conduct static analysis of the Android application package (APK) to identify data collection methods (DCMs) and extract collection evidence, which highlights the gap between privacy policy claims and actual practices (**RQ4**).

Our analysis is divided into two parts. First, we identify DCMs from the top 20 Android analytic services and customized analytics services. Second, we extract collection evidence by focusing on invocations to analytics services, associated UI widgets, and the callbacks triggered by registered listeners.

4.1 Identifying Data Collection Methods

Data collection methods (DCMs) are methods defined by analytics services, such as Firebase Analytics, that allow app developers to log user interaction data. DCMs provide a standardized way for app developers to collect user interaction data and track app usage in order to analyze and understand user behavior.

For example, the Firebase Analytics API provides the `logEvent()` method to log user events, such as button clicks or screen views. Suppose we have a button `myButton` in the app's UI, and we want to track when the user clicks on it. We can do this using Firebase Analytics by adding the following code to the button's `OnClickListener`:

```
myButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        FirebaseAnalytics.getInstance(this).
            logEvent("button_click", null); }
});
```

Here `FirebaseAnalytics.getInstance(this)` returns an instance of the Firebase Analytics object, and `logEvent("button_click", null)` collects the button click interaction data with the string `"button_click"` to Firebase Analytics.

To determine how Android apps use analytics services, we identified DCMs from the top 20 Android analytic services, cf. Section 3.1. Matching the full signature of these methods in bytecode allows us to find direct invocations to analytics services. However, some apps use customized analytics services to do a more fine-grained collection, such as collecting motion details and duration. To do this, the apps implement their own analytics classes by extending the analytic services.

To identify customized analytics, we use static analysis to identify the classes that invoke external DCMs. We then check whether these classes are invoked in any of the app's declared activities. If they are, we mark these classes as customized analytics services classes.

[7] Refer to the claim vocabularies in Section 3.1

4.2 Extracting Collection Evidence

Next, we extracted evidence of actual data collection from the APK. Specifically, we analyze three types of information: (1) invocations to analytics services that logged user interaction data collection, (2) associated UI widgets, and (3) the callbacks triggered by registered listeners on these UI widgets.

We utilized static analysis with FlowDroid [3] to associate DCM invocations with callbacks, listeners, and activities in the bytecode. We then compared the layout IDs of the associated UI widgets defined in the layout XML files to identify the relevant collection data types and techniques.

The relationships between different parts of the extracted collection evidence in an Android app are shown in Fig. 2. The UI-related parts, such as layout files and defined UI widgets, provide information on the types of user interaction data (red section), while the bytecode provides details on the techniques of collection (blue section)[8].

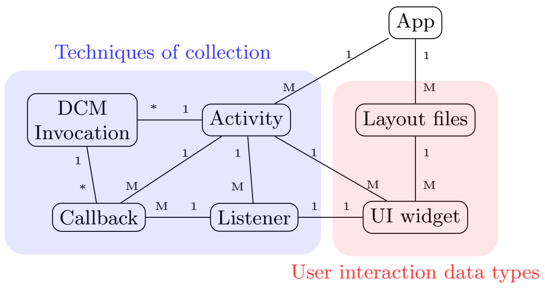


Fig. 2. Relationships between different parts of the extracted collection evidence in an Android app

We return to the Yr weather app, the example app from Section 1. Based on the collection evidence extracted from Yr's bytecode and layout files, we discovered that it collects detailed user interaction data using various types of UI widgets such as `SearchView` and `Textfield`. This data collection is linked to features such as changes in location, enabling forecast summary notifications, and opening the forecast graph. Building on this finding from static analysis, we propose the following more specific checked standardized collection claim:

Checked Standardized Collection Claim for Yr
We collect the following types of user interaction data: *app presentation, binary and categorical interactions, and user input interactions*, along with their *frequency.*

5 Findings

To address **RQ2**, we conduct a manual inspection of 1411 sentences that described user interaction data collection in all 350 privacy policies within the APP-350 corpus, as outlined in Section 3.2.

We examine whether the sentences in a privacy policy provide clear descriptions of the types of user interaction data collected and the techniques of collection.

We find that only 37% of the identified sentences contained clear statements on both the data types and techniques of collection, while 41% only discussed the techniques of collection and 22% mentioned only the data types.

[8] Note: The figure notation is as follows: 1-M means one-to-many, 1-$*$ means one-to-any (zero or more), and 1-1 means one-to-one.

Here are the relevant sentences from two policies in the corpus. DAMI[9] states: "*We may work with analytics companies to help us understand how the Applications are being used, such as the frequency and duration of usage.*" Wish[10] states: "*We may collect different types of personal and other information based on how you interact with our products and services. Some examples include: Equipment, Performance, Websites Usage, Viewing and other Technical Information about your use of our network, services, products or websites.*"

DAMI's privacy policy only discloses the techniques of collection, such as the frequency and duration of usage, without clearly explaining which type of user interaction data is collected. In contrast, Wish's privacy policy does mention some specific types of data collected, such as equipment and performance data, but it is unclear about which techniques of collection are used.

The majority of identified sentences discuss the techniques of collection rather than specific data types, suggesting that organizations use the tactic of avoiding or minimizing disclosures about the types of user interaction data they collect in order to collect more data than users are aware of or comfortable with.

To investigate **RQ3**, we performed a static analysis on a sample of 100 free Android apps downloaded from the top 10 most popular categories on the German Google Play store[11], as identified by AppBrain. In cases where the same app appeared in several categories, we moved to the next popular app in the second category to get a total of 100 distinct apps.

Table 3. Statistics of the user interaction data collection for the top 100 Android apps.

UI type (types of interaction data)	**Top 2 techniques of collection**	**Top 3 app categories**	**Percent collected**	**Avg # collected**
View (Presentation)	Frequency (100%), Duration (52%)	Entertainment, Shopping, Travel	89%	12
Button (Binary)	Frequency (94%), Motion (8%)	Social, Utility, Gaming	76%	26
Textfield (Input)	Frequency (100%), Duration (4%)	Social, Shopping, Utility	63%	5
Checkbox & Spinner (Categorical)	Frequency (97%), Motion (16%)	Shopping, Travel, Utility	32%	7
GestureDetector (Gesture)	Motion (94%), Duration (40%)	Gaming, Entertainment, Social	16%	38

Our analysis of the top 100 Android apps revealed that app developers placed a great deal of emphasis on understanding how frequently users interacted with different UI elements (which may correspond to different features or functionalities in the app), as frequency was the top techniques of collecting user interaction data across all UI types. We also found that the average number of interaction data collected varied significantly across different types of UI. It was also interesting to see that the high number of interaction data collected for the button UI type (also found in 76% of the apps), indicated that understanding button usage was a particularly important metric for app developers.

Table 3 presents an overview of the user interaction data collection practices across various app categories, focusing on the top UI type for each type of interaction data. We have selected the most frequently occurring UI types from each category for this analysis, which are listed in the first column.

The second column indicates the top two techniques linked with each UI type. The percentages in parentheses, for instance, 100% and 52% for the "View" UI type, represent the proportion of apps that use a particular technique in tracking the UI type. For example, 100% of apps tracked "View" interactions use the frequency technique, while 52% also use the duration technique.

[9] https://play.google.com/store/apps/details?id=com.blappsta.damisch
[10] https://play.google.com/store/apps/details?id=com.contextlogic.wish
[11] https://play.google.com/store/apps?gl=DE

The "Percent collected" column indicates the proportion of the top 100 apps that collect data related to a specific UI type. For instance, "View" data is collected by 89% of the analyzed apps.

Finally, the "Average # collected" column represents the average number of distinct DCMs detected in each app associated with a particular UI type. For example, on average, 12 distinct DCMs were detected for "View" data collection across the analyzed apps.

Upon comparing these user interaction data collection practices with the declarations in privacy policies, we observe a larger mismatch in certain app categories. Gaming apps, despite their high prevalence of Gesture data collection to optimize user experience, often lack comprehensive disclosure of such practices in their privacy policies. Similarly, Entertainment, Shopping, and Travel apps extensively collect "View" data, but their policies rarely match the extent of this data collection, indicating a transparency gap in these visually-centric applications.

Social and Utility apps, which heavily rely on "Button" and "TextField" data, also demonstrate a significant disparity between their actual data collection practices and policy disclosures. These observations highlight that while app developers tailor their data collection strategies to their specific objectives and requirements, they often fail to mirror this granularity in their privacy policies.

This mismatch is consequential as it affects the transparency of these apps and the users' ability to make informed decisions. Hence, addressing these discrepancies becomes crucial, and our findings provide valuable insights for developers aiming to improve their privacy disclosures, ultimately fostering trust and success in the app ecosystem.

To address **RQ4**, we manually inspected the privacy policy claims of the most popular app in each of the 10 categories on Google Play. We generated our checked collection claims by analyzing the actual data collection practices of each app and comparing them to the privacy policy claims published by the app. Our checked collection claims are made by combining the evidence gathered through static analysis and the proposed standardized claim template. The results are fact-check collection claims presented in Table 4.

Our findings uncovered inconsistencies between the claims made in privacy policies and the actual data types and techniques of collection used by popular apps on Google Play. Many apps do not fully disclose the types of data collected or the techniques of collection, often using vague language such as "collecting user interactions to improve the service".

Notably, some apps that may be perceived as having questionable data collection practices, such as TikTok and Amazon Prime Video, actually provided more detailed information on the types of data collected and the techniques of collection used. TikTok and Duolingo even provided specific examples of their data collection practices.

However, we found that some apps from less controversial categories, such as the photography editing app Picsart and the payment platform PayPal, used opaque language in their privacy policies, leaving a large gap between their claims and our findings. The most extreme example was Booking.com, which extensively collects user interactions within the app, yet discloses almost no information in its privacy policy. These findings highlight the need for clearer and more comprehensive disclosures in privacy policies, particularly for apps that collect sensitive user data.

5.1 Threats to Validity

Potential threats to the validity of our experiment may impact the interpretation of our findings. A primary limitation of our experiment is the number of apps we manually fact-checked for data collection practices. Due to the complexity of accommodating varying UI types and callbacks into our predefined six data types and three techniques of collection, we were only able to manually fact-check one app in each category, totaling ten apps. This sample size, though limited, may not encapsulate the full diversity of data collection practices across all apps.

Furthermore, measuring the recall of our analysis posed a considerable challenge, given the absence of a comprehensive ground truth detailing all interaction data collection practices in each app. Consequently, our findings may not wholly represent the full range of data collection practices.

Table 4. Fact-checked data collection claims w.r.t. evidence for the most popular app from each of the top 10 categories of Google Play.
The red text indicates types of user interaction data missing from the privacy policy/collection claims, while the blue text indicates undisclosed techniques of collection.

Checked Collection Claim	**Related Text in the Published Privacy Policy**
[**TikTok**] We collect the following types of user interaction data: app presentation, binary, categorical, user input, gesture and composite gesture interactions, along with their frequency, duration and motion details.	[**TikTok**] We collect information about how you engage with the Platform, including information about the content you view, the duration and frequency of your use, your engagement with other users, your search history and your settings.
[**SHEIN**] We collect the following types of user interaction data: app presentation, binary, categorical, user input interactions, along with their frequency and duration.	[**SHEIN**] Data about how you engage with our Services, such as browsing, adding to your shopping cart, saving items, placing an order, and returns for market research, statistical analysis, and the display of personalized advertising based on your activity on our site and inferred interests; Collect your device information, and usage data on our website or app for fault analysis, troubleshooting, and system maintenance, as well as setting default options for you, such as language and currency. The display of information you choose to post on public areas of the Services, for example, a customer review.
[**Booking.com**] We collect the following types of user interaction data: app presentation, binary, categorical, user input interactions, along with their frequency and duration.	[**Booking.com**] We collect data that identifies the device, as well as data about your device-specific settings and characteristics, app crashes and other system activity.
[**PayPal**] We collect the following types of user interaction data: app presentation, binary, categorical, user input interactions along with their frequency.	[**PayPal**] When you visit our Sites, use our Services, or visit a third-party website for which we provide online Services, we and our business partners and vendors may use cookies and other tracking technologies to recognize you as a User and to customize your online experiences, the Services you use, and other online content and advertising; measure the effectiveness of promotions and perform analytics; and to mitigate risk, prevent potential fraud, and promote trust and safety across our Sites and Services.
[**Duolingo**] We collect the following types of user interaction data: app presentation, binary, categorical, user input, gesture interactions, along with their frequency and duration.	[**Duolingo**] We do record the following data: Patterns, Clicks, Mouse movements, Scrolling, Typing, Pages visited, Referrers, URL parameters, Session duration.
[**Amazon Prime Videos**] We collect the following types of user interaction data: app presentation, binary, categorical, user input, gesture interactions, along with their frequency, duration and motion details.	[**Amazon Prime Videos**] We automatically collect and store certain types of information about your use of Amazon Services including your interaction with content and services available through Amazon Services. List of examples: search for products or services in our stores and download, stream, view, or use content on a device, or through a service or application on a device.
[**Yazio**] We collect the following types of user interaction data: binary and user input interactions, along with their frequency.	[**Yazio**] The Firebase Analytics service helps to determine the interactions of App users by recording, for instance, the first time the App is opened, deinstallations, updates, system crashes and how often the App is used. The service also records and analyses certain user interests.
[**Fasion Famous**] We collect the following types of user interaction data: app presentation, binary, user input, gesture and composite gesture interactions, along with their frequency, duration and motion details.	[**Fasion Famous**] Information that may be collected automatically: Data and analytics about your use of our Services. Data we collect with cookies and similar technologies: Data about your use of our Services, such as game interaction and usage metrics.
[**Picsart**] We collect the following types of user interaction data: app presentation, binary, gesture and composite gesture interactions, along with their frequency, duration and motion details.	[**Picsart**] Our servers passively keep an electronic record of your interactions with our services, which we call "log data". We collect and combine data about the devices you use to access Picsart, and data about your device usage and activity.
[**Dezor**] We collect the following types of user interaction data: app presentation, binary, categorical, user input interactions, along with their frequency.	[**Dezor**] The information collected by log files include internet protocol (IP) addresses, browser type, Internet Service Provider (ISP), date and time stamp, referring/exit pages, and possibly the number of clicks.

6 Related Work

The related work can be categorized into three primary themes: (1) privacy policy analysis using NLP and policy compliance check, (2) static analysis for security and privacy in apps, and (3) analytics services analysis.

6.1 Privacy Policy Analysis

Numerous studies have focused on analyzing and improving privacy policies in mobile apps. Researchers have explored various NLP approaches to automatically process and understand privacy policy texts, as well as to assist users in comprehending these policies more effectively [21, 22, 25]. However, these studies do not specifically address the issue of user interaction data collection, which is a significant gap that our research addresses. Tools like PrivacyFlash Pro [30] and AutoCog [20] have been developed to audit privacy policy compliance by comparing disclosed policies with actual app behavior, but they primarily focus on personal data, not user interaction data. A recent study by Bardus et al. [5] systematically mapped existing contact-tracing apps and evaluated the permissions required and their privacy policies, but it did not delve into the specifics of user interaction data collection.

6.2 Static Analysis for Security and Privacy

The static analysis approach has been used to enhance security and privacy in mobile apps. This involves analyzing app bytecode, identifying data leaks, and detecting privacy violations [4, 11, 29]. Despite the progress in this field, there remains an underrepresentation of studies targeting user interaction data, a type of data often overlooked in privacy policies and their corresponding analyses. A novel system, LocationScope, was presented by Lu et al. [18] to detect and measure aggressive location harvesting in mobile apps at scale, but it did not specifically target user interaction data.

6.3 Analytics Services Analysis

Another line of research has concentrated on the role of analytics services in capturing user data, primarily focusing on PII. Alde [17], for example, proposed a method employing both static and dynamic analysis to detect the key information gathered by analytics libraries, which are largely device-level data. PAMDroid [28] takes a similar approach, identifying personal data funneled into analytics services and treating it as a misconfiguration. The domain of user interaction data collection, however, remains relatively untouched in these studies. A recent study by Laperdrix et al. [16] presented a privacy analysis of free and paid games in the Android ecosystem, but it did not specifically focus on user interaction data collection.

These studies have contributed to the understanding of privacy policies and data collection practices in mobile apps. However, there is a lack of research specifically on the practices of user interaction data collection and the transparency of related claims in privacy policies. Our work extends the scope of previous research by focusing on user interaction data collection practices and providing an analysis on comparing privacy policy disclosures with actual app behavior. This approach aims to enhance transparency and trust in the mobile app ecosystem, addressing the research gaps in the existing literature.

7 Conclusion and Future Work

In conclusion, our analysis of the top 100 apps uncovers the widespread collection of user interaction data, while the detailed examination of the top 10 apps reveals that privacy policies often inadequately disclose such practices. To address this lack of transparency, we introduced a standardized collection claim template that aids app developers in accurately detailing their data collection practices. This approach fosters informed decisions by users and enhances transparency by allowing assessments of alignment between declared and actual data collection practices for the manually analyzed apps. Our findings lay the groundwork for improving data collection transparency in mobile apps and highlight the need for automating the policy-to-claims analysis. This insight could potentially guide future research and policy-making to foster a more secure and trustworthy app ecosystem.

Our approach has limitations that can be addressed in future research to improve the analysis of data collection practices. The current analysis only covers the top 20 analytics services and is confined to Android apps. Furthermore, the manual fact-checking of the top 10 apps relies on our interpretation of their policies. To overcome these limitations, machine learning models could be employed to automatically identify and categorize data collection methods (DCMs) within app code, reducing the need for manual analysis. This would involve training models to detect DCMs and categorizing them based on the data types and collection techniques they employ. Additionally, a more precise and fine-grained policy analysis could be developed to automatically extract interaction data types and collection techniques from privacy policies. By combining these advancements, we could create a fully automated approach to fact-check collection claims against the collection evidence, thereby increasing the efficiency and accuracy of analyzing data collection practices in mobile applications.

Another potential area for future work is the exploration of user studies to understand users' perceptions of interaction data collection practices and their impact on users' trust and app usage. Extending the analysis to include other platforms and analytics services could also contribute to a more holistic understanding of user interaction data collection practices across the mobile app ecosystem.

Acknowledgement

We appreciate the valuable insights provided by Professor Staal Vinterbo. This work is part of the Privacy Matters (PriMa) project. The PriMa project has received funding from European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 860315.

References

1. Almuhimedi, H., Schaub, F., Sadeh, N., Adjerid, I., Acquisti, A., Gluck, J., Cranor, L.F., Agarwal, Y.: Your location has been shared 5,398 times! a field study on mobile app privacy nudging. In: Proceedings of the 33rd annual ACM conference on human factors in computing systems. pp. 787–796 (2015)
2. AppTornado: AppBrain: Android analytics libraries. https://www.appbrain.com/stats/libraries/tag/analytics/android-analytics-libraries, (Accessed on 03/04/2023)
3. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., McDaniel, P.: Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. ACM SIGPLAN Notices **49**(6), 259–269 (2014)
4. Avdiienko, V., Kuznetsov, K., Gorla, A., Zeller, A., Arzt, S., Rasthofer, S., Bodden, E.: Mining apps for abnormal usage of sensitive data. In: 2015 IEEE/ACM 37th IEEE international conference on software engineering. vol. 1, pp. 426–436. IEEE (2015)
5. Bardus, M., Al Daccache, M., Maalouf, N., Al Sarih, R., Elhajj, I.H.: Data management and privacy policy of covid-19 contact-tracing apps: Systematic review and content analysis. JMIR mHealth and uHealth **10**(7), e35195 (2022)
6. Bird, S., Klein, E., Loper, E.: Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc. (2009)
7. Buriro, A., Akhtar, Z., Crispo, B., Del Frari, F.: Age, gender and operating-hand estimation on smart mobile devices. In: 2016 International Conference of the Biometrics Special Interest Group (BIOSIG). pp. 1–5 (2016)

8. Cysneiros, L.M., Werneck, V.: An initial analysis on how software transparency and trust influence each other. In: Workshop em Engenharia de Requisitos (2009)
9. Degirmenci, K., Guhr, N., Breitner, M.: Mobile applications and access to personal information: A discussion of users' privacy concerns. In: Proceedings of the 34th International Conference on Information Systems (ICIS 2013). pp. 1–21. Association for Information Systems (AIS) (2013)
10. Dumais, S., Jeffries, R., Russell, D.M., Tang, D., Teevan, J.: Understanding user behavior through log data and analysis. Ways of Knowing in HCI pp. 349–372 (2014)
11. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS) **32**(2), 1–29 (2014)
12. Fischer-Hübner, S., Angulo, J., Karegar, F., Pulls, T.: Transparency, privacy and trust–technology for tracking and controlling my data disclosures: Does this work? In: Trust Management X: 10th IFIP WG 11.11 Conference, IFIPTM 2016, Darmstadt, Germany, July 18-22, 2016, Proceedings 10. pp. 3–14. Springer (2016)
13. Gadotti, A., Houssiau, F., Annamalai, M.S.M.S., de Montjoye, Y.A.: Pool inference attacks on local differential privacy: Quantifying the privacy guarantees of apple's count mean sketch in practice. In: USENIX Security 22. pp. 501–518 (2022)
14. Honnibal, M., Montani, I.: spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing (2017)
15. Jain, A., Kanhangad, V.: Gender recognition in smartphones using touchscreen gestures. Pattern Recognition Letters **125**, 604–611 (2019)
16. Laperdrix, P., Mehanna, N., Durey, A., Rudametkin, W.: The price to play: a privacy analysis of free and paid games in the android ecosystem. In: Proceedings of the ACM Web Conference 2022. pp. 3440–3449 (2022)
17. Liu, X., Liu, J., Zhu, S., Wang, W., Zhang, X.: Privacy risk analysis and mitigation of analytics libraries in the android ecosystem. IEEE Transactions on Mobile Computing **19**(5), 1184–1199 (2020). https://doi.org/10.1109/TMC.2019.2903186
18. Lu, H., Zhao, Q., Chen, Y., Liao, X., Lin, Z.: Detecting and measuring aggressive location harvesting in mobile apps via data-flow path embedding. Proceedings of the ACM on Measurement and Analysis of Computing Systems **7**(1), 1–27 (2023)
19. Morey, T., Forbath, T., Schoop, A.: Customer data: Designing for transparency and trust. Harvard Business Review **93**(5), 96–105 (2015)
20. Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T., Chen, Z.: Autocog: Measuring the description-to-permission fidelity in android applications. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 1354–1365 (2014)
21. Ramanath, R., Liu, F., Sadeh, N., Smith, N.A.: Unsupervised alignment of privacy policies using hidden markov models. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics. pp. 605–610 (2014)
22. Ravichander, A., Black, A.W., Norton, T., Wilson, S., Sadeh, N.: Breaking down walls of text: How can nlp benefit consumer privacy? In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing. vol. 1 (2021)
23. Rzhevkina, A.: Several EU countries banned Google Analytics - here are some alternatives. https://www.contentgrip.com/eu-countries-ban-google-analytics/ (September 2022), (Accessed on 03/12/2023)
24. Story, P., Zimmeck, S., Ravichander, A., Smullen, D., Wang, Z., Reidenberg, J., Russell, N.C., Sadeh, N.: Natural language processing for mobile app privacy compliance. In: AAAI Spring Symposium on Privacy-Enhancing Artificial Intelligence and Language Technologies (2019)
25. Tesfay, W.B., Hofmann, P., Nakamura, T., Kiyomoto, S., Serna, J.: Privacyguide: Towards an implementation of the eu gdpr on internet privacy policy evaluation. In: Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics. p. 15–21. IWSPA '18, Association for Computing Machinery, New York, NY, USA (2018)
26. Verkasalo, H.: Analysis of smartphone user behavior. In: 2010 Ninth International Conference on Mobile Business and 2010 Ninth Global Mobility Roundtable (ICMB-GMR). pp. 258–263. IEEE (2010)
27. Vorm, E., Combs, D.J.: Integrating Transparency, Trust, and Acceptance: The Intelligent Systems Technology Acceptance Model (ISTAM). International Journal of Human–Computer Interaction **38**(18-20), 1828–1845 (2022)

28. Zhang, X., Wang, X., Slavin, R., Breaux, T., Niu, J.: How does misconfiguration of analytic services compromise mobile privacy? In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). pp. 1572–1583 (2020)
29. Zhang, X., Wang, X., Slavin, R., Breaux, T., Niu, J.: How does misconfiguration of analytic services compromise mobile privacy? In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. pp. 1572–1583 (2020)
30. Zimmeck, S., Goldstein, R., Baraka, D.: Privacyflash pro: Automating privacy policy generation for mobile apps. In: NDSS (2021)

PAPER 6

User Interaction Data in Apps: Comparing Policy Claims to Implementations

Tang, F. and Østvold, B. (2023). User Interaction Data in Apps: Comparing Policy Claims to Implementations. Published at the 18th IFIP Summer School on Privacy and Identity Management 2023 (IFIPSC 2023).

User Interaction Data in Apps: Comparing Policy Claims to Implementations[⋆]

Feiyang Tang[0000−0002−8720−6743] and Bjarte M. Østvold[0000−0001−6922−4027]

Norwegian Computing Center
N-0314 Oslo, Norway
{feiyang,bjarte}@nr.no

Abstract. As mobile app usage continues to rise, so does the generation of extensive user interaction data, which includes actions such as swiping, zooming, or the time spent on a screen. Apps often collect a large amount of this data and claim to anonymize it, yet concerns arise regarding the adequacy of these measures. In many cases, the so-called anonymized data still has the potential to profile and, in some instances, re-identify individual users. This situation is compounded by a lack of transparency, leading to potential breaches of user trust.
Our work investigates the gap between privacy policies and actual app behavior, focusing on the collection and handling of user interaction data. We analyzed the top 100 apps across diverse categories using static analysis methods to evaluate the alignment between policy claims and implemented data collection techniques. Our findings highlight the lack of transparency in data collection and the associated risk of re-identification, raising concerns about user privacy and trust. This study emphasizes the importance of clear communication and enhanced transparency in privacy practices for mobile app development.

Keywords: Mobile Apps · Transparency · Trust · Interaction Data · Privacy Policy

1 Introduction

Mobile apps have become deeply integrated into daily life, often collecting user interaction data like taps and swipes. While this data is typically anonymized to protect privacy, the effectiveness of this anonymization is increasingly under scrutiny. Anonymized data, when aggregated, can still enable user profiling and potentially lead to identification. This challenges the common practice of labeling such data as “non-personal” to meet less stringent privacy regulations. These practices, under the pretext of anonymization, pose significant privacy risks and can diminish user trust.

To address these issues we propose an automated method to compare privacy policy statements with actual data collection practices in app code. We aim to highlight the discrepancies between policy and practice, thereby enhancing transparency and rebuilding trust. Our focus extends beyond mere regulatory compliance to advocating for stronger data protection and a culture of transparency in the digital domain.

This paper aims to answer the following research questions:

1. What claims do app privacy policies make concerning the collection of user interaction data?
2. What insights can be derived from analyzing app implementations in light of policy claims?
3. How can we automate the examination of the transparency of collection claims in privacy policies based on evidence obtained by static analysis?

Our contributions extend and automate our previously work [12]:

[⋆] *Published at the 18th IFIP Summer School on Privacy and Identity Management 2023 (IFIPSC 2023).*

1. Extending the manual analysis approach [12], we introduce an automated claim extractor and classifier for processing privacy policies. This approach uses natural language processing techniques, enhanced by targeted keyword searches, to automatically extract and categorize claims about user interaction data collection.
2. We develop a static analyzer and an evidence classifier. These components automatically extract and categorize details of user interaction data collection directly from app implementations, streamlining the process.
3. By automating the comparison of labeled collection claims (extracted from privacy policies) with the labeled collection evidence (derived from application code), our approach provides a more efficient and objective assessment of the transparency of data collection practices.
4. Building upon the automated components, we conduct a study of 100 popular mobile apps. This study aims to analyze and identify patterns in user interaction data collection, enhancing the understanding of this practice and its implications for privacy and transparency.

Our two-fold approach, encompassing privacy policy analysis and application code analysis, is depicted in Fig. 1.

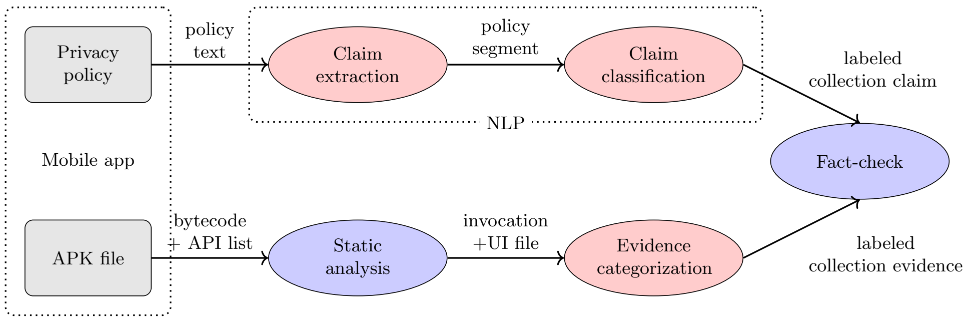


Fig. 1: Overview of the approach

2 Motivation

The capability of anonymized user interaction data to be de-anonymized and thus potentially classified as personal data under regulations such as the GDPR is an emerging concern in user privacy. Studies, including those by Leiva et al. [6], highlight that even data devoid of explicit personal identifiers can be subjected to user profiling and identification of users through the aggregation of interaction data with other contextual information. This complex interplay blurs the distinction between non-personal and personal data, thus challenging the notion that anonymized data is inherently non-sensitive or non-identifiable.

Furthermore, the work of Creţu et al. [2] suggests that stable behavioral patterns within anonymized mobile app data can be leveraged to achieve high re-identification rates. Such findings directly challenge the GDPR's current classification of this data as non-personal. This misinterpretation of what constitutes personal data, especially in the context of user interaction, can significantly undermine privacy risks. It also raises questions about the effectiveness of anonymization techniques and the need for informed consent and transparent data governance, even when data is seemingly anonymized [5].

The ambiguity in categorizing data as personal or non-personal is further complicated in mixed datasets, where differentiating between the two becomes increasingly challenging [7]. This is particularly relevant in the context of mobile apps, where user interaction data is often collected

Table 1: Most frequent bigrams

Bigram	Freq.
Your Information	3,295
Our Service	2,941
Your Data	2,892
Third Party	1,788
Help You	1,214
Improve Service	947
Automatic Collection	422
Tracking Technology	402
Interact With	346
Collect Information	281

alongside other types of data. The risk of re-identification in what is classified as non-personal data underscores the importance of rethinking how such data is treated within legal frameworks. As argued in existing literature, there is a growing necessity to treat non-personal data with the same level of protection as personal data [11].

Transparency in mobile app data collection is critical for user trust and app adoption, as it ensures user autonomy and accountability [12], especially with the rise of analytics services that may infringe upon GDPR guidelines [10]; thus, fostering transparency and user control is imperative for sustaining user satisfaction and promoting app engagement.

3 Analyzing Collection Claims from Privacy Policies

In an attempt to assess the transparency of data collection practices stated in privacy policies, we developed a two-tiered approach. This strategy is specially designed to extract and classify claims related to user interaction data collection, a facet less explored in privacy policy analysis.

This approach aims to answer three primary questions:

- *Does the privacy policy mention user interaction data collection?*
- *If so, what types of user interaction data are claimed to be collected?*
- *What techniques are claimed to be used for this data collection?*

3.1 Claim Extraction

The first phase of our approach identifies whether user interaction data collection is mentioned within a privacy policy. Instead of conventional keyword-based approaches, this extractor utilizes semantic context to accommodate the diverse ways such claims can be articulated.

The APP-350 Corpus was utilized in this stage [16]. This corpus comprises 350 Android app privacy policies annotated for privacy practices. However, the existing annotations primarily focused on personal data collection, which didn't coincide with our emphasis on user interaction data collection. Therefore, we conducted our own manual annotations.

Key Findings: Our review of the 350 app privacy policies yielded several key findings that offered insights into the disclosure practices regarding user interaction data collection.

By doing manual annotation we found that out of the 350 analyzed apps, 294 mentioned the collection of user interaction data at varying detail levels. However, of these 294, only 57% (169/294) of the policies provided more specifics than a mere mention of "data" or "information." Upon segmenting the privacy policies into sentences, we annotated 3,661 sentences as relevant

Table 2: Data collection frequencies (169 apps)

Data Type	**Freq.**
App Presentation	98%
Categorical	60%
User Input	45%
Binary	17%
Gesture/Composed Gesture	2%
Device Data [1]	92%

to user interaction data collection from a total of 42,797 sentences. Table 1 presents the most frequently occurring bigrams within these annotated sentences.

Transparency about user interaction data collection varied significantly across apps. Although 294 policies referenced such data collection, the details were often obscured by general phrases like "*we collect data to improve our service.*" Our bigram analysis highlighted the common use of third-party services in the data collection process. These services, often referred to as "*tracking technology*", are employed to automatically collect data purportedly to enhance services. "*Google Analytics*", a prominent third-party analytics service, was frequently observed in our bigram analysis, underscoring its vital role in user interaction data collection.

3.2 Claim Classification

The second phase of our approach classifies the claims. This model is innovative in its ability to categorize claims according to user interaction data types and collection techniques. Unlike traditional binary classifiers, it acknowledges that a single sentence may convey multiple types of data and collection techniques.

Key Findings: In our examination of the 169 privacy policies that offered more explicit information about user interaction data collection, we found that only 56 policies clearly stated the collection techniques, such as *"the times you click a page"* or *"the time you spend watching content"*.

To standardize vocabularies and taxonomy for classification purposes, we utilized data types and collection techniques from our previous work, known as collection vocabularies [12]. These vocabularies included six types of interaction data and an additional category named device data, which we observed is commonly collected alongside interaction data. The frequencies with which of different data collection types are mentioned in the policies are shown in Table 2.

The descriptions given by the apps about their collection techniques were often vague. Of the 56 apps that vaguely mentioned the techniques used, all referred to frequency, representing 100% of this subgroup. A substantial but smaller portion, 48% (27 out of 56), mentioned duration, using phrases like *"time spent watching"* or *"length of service use"*. However, only a mere 1.8% (1 out of 56) of these apps mentioned motion.

Transparency was lacking in the descriptions of user interaction data collection types and techniques. Of the 294 apps that acknowledged data collection, a majority, 84% (248 out of 294), categorized the collected data as *"non-personal data"*, without providing further details. Such categorization seemed to be used to justify sensitive actions like *"aggregation"*, a method mentioned by 43% (126 out of 294) of these apps, and *"transfer to third-party services"*, an action mentioned

[1] While not part of interaction data, it is a crucial component often collected alongside interaction data. This includes information such as the International Mobile Equipment Identity (IMEI) number, device model, operating system, and other device-specific identifiers.

by 68% (199 out of 294). Furthermore, almost half, 48% (141 out of 294), acknowledged using *"automatic collection"* methods.

Further details on dataset analysis, along with the training and testing procedures for both the claim extractor and claim classifier, can be found in the Appendix under Section A.

4 Analyzing Collection Evidence from Application Code

Once the collection claims from the privacy policy have been extracted, we seek to validate these claims by investigating the application code for tangible signs of data collection. Our attention is primarily devoted to identifying and categorizing the embedded data collection techniques within the mobile app. The approach we adopt for static analysis explicitly targets user interface (UI) elements and the invocations to analytics libraries from these UI elements. Following identification, these elements are classified based on a predefined collection vocabulary that we have introduced in [12]. This vocabulary was generated through a meticulous examination of all Android UI widgets and it captures a broad range of user interaction data types and collection techniques. The detailed terms for types of user interaction data and collection techniques employed in our study are listed in the Appendix.

The collection vocabulary not only allows for a structured classification of data collection instances but also facilitates the mapping between the collection evidence found in the code and the claims made in the privacy policy. The usage of this comprehensive vocabulary ensures that we can conduct a granular comparison later in the fact-checking process.

Through our analysis, we aim to answer the following questions:

- *Which analytics libraries are being utilized by the mobile app?*
- *What types of user interaction data are being collected?*
- *Which techniques are employed for data collection?*

4.1 Analytics Library Identification

In the first stage of our code analysis, we focus on identifying the analytics libraries that are used by the mobile apps. It is common for apps to utilize such libraries to gather and analyze user interaction data, providing developers with valuable insights into user behavior.

To achieve this, we target a set of popular analytics libraries as our initial point of focus. These libraries are often integral to tracking user interactions and facilitating data collection. Hence, recognizing these libraries' invocations serves as an efficient guide to pinpoint locations where user interaction data collection is likely to take place.

Our analysis primarily focuses on the classes that engage with UI elements, carefully examining the imported analytics libraries along with their respective method invocations. We constrain our investigation to a selected set of methods belonging to popular analytics libraries that are frequently utilized for data collection. In this context, we adopted the list of the top 20 analytics services for Android apps listed on AppBrain[2]. Prior understanding of these frequently used analytics libraries and their APIs forms a crucial foundation for this stage of our analysis.

4.2 Categorizing Data Types and Collection Techniques

Following the identification of analytics libraries, our objective is to establish links between the UI elements and the corresponding bytecode that manages user interactions. UI actions such as button presses trigger specific methods within the bytecode. Thus, we delve into both XML files, which define the UI elements, and the bytecode, which dictates the actions corresponding to these elements.

[2] https://www.appbrain.com/stats/libraries/tag/analytics/android-analytics-libraries

Table 3: Types of user interaction data and corresponding UI elements

Interaction Data Types	**Android UI Elements**
App Presentation	View (TextView, VideoView, WebView, etc.)
Binary	Button (ImageButton, CheckBox, etc.)
Categorical	AbsSpinner (Spinner), CompoundButton (RadioButton, Switch), RatingBar
User Input	TextView (EditText, AutoCompleteTextView, SearchView)
Gesture	GestureDetector, ViewPager, SwipeRefreshLayout
Composite Gestures	GestureDetector (ScaleGestureDetector)

The examination of these components often provides insights into the type of user interaction data being collected.

For instance, consider a simple scenario where a Firebase Analytics library is employed in an Android app. A button click in the UI represented as `<Button android:onClick="buttonClick"/>` in the XML file, would trigger a corresponding `buttonClick(View view)` method in the Java code. The interaction with the analytics library within this method could look something like this:

```
public void buttonClick(View view) {
    FirebaseAnalytics mFAnalytics = FirebaseAnalytics.getInstance(this);
    Bundle params = new Bundle();
    params.putString("Button_name", "button1");
    params.putString("Action", "click");
    mFAnalytics.logEvent("ButtonClick", params);
}
```

Here, an invocation to the Firebase Analytics library occurs whenever the button is clicked, recording the button's name and the associated action. This example highlights that click data is collected each time the button is clicked.

Though this method generally proves effective in discerning the types of user interaction data being collected, it is important to note that some complexities in the bytecode may obscure certain data collection events. Additionally, data collected outside of standard UI interactions, such as device-generated data or data from non-UI sources, may not be captured by this approach. Building upon the successful linking of UI elements to their corresponding analytics library invocations, we categorize the extracted data based on predefined interaction data types and collection techniques. Our initial focus is on the types of user interaction data, where we aim to classify the data according to their corresponding UI elements. Table 3 presents a classification of interaction data types associated with common Android UI elements.

In the table, the main Android UI elements represent the core classes or interfaces in the Android UI hierarchy. For instance, View is a fundamental class for UI widgets in Android, and the various UI elements like TextView, VideoView, and WebView are its subclasses, hence included as its subcategories.

In this process, we perform an inspection of each UI element across the XML files, which define the UI, and the code files that handle these UI elements. Accordingly, the type of user interaction data is ascertained based on the functionality attributed to the UI elements.

Identification of Collection Techniques Our approach to identifying the collection techniques for user interaction data consists of two components: rules-based identification using predefined criteria, and criteria obtained from a detailed analysis of popular analytics libraries' documentation.

In rules-based identification, we create a set of heuristics centered on invocations of Android or Java methods, which are associated with different collection techniques. For instance, the "frequency" technique can be inferred from the event logging invocation. Techniques like "duration" collection can be suggested by invocations of methods from the Java `Timer` class or `android.os.SystemClock.elapsedRealtime()`. Similarly, "motion details" collection stem from methods in the `MotionEvent` class, such as `getPressure()`, `getX()`, and `getY()`.

The second component of our approach involves using criteria obtained from the documentation of widely-used analytics libraries, such as Firebase Analytics and Mixpanel. Once the specific API methods used for different collection techniques in these libraries are identified, they are added to our categorization list. For instance, Firebase Analytics' `logEvent()` method, with parameters like `select_content` and `view_item`, can log the frequency of user interactions. On the other hand, Mixpanel uses the `track()` method with event names to record frequency. For recording duration, Firebase Analytics uses the `user_engagement` event, capturing user engagement duration, while Mixpanel provides the `time_event()` method to time events' duration.

While this approach provides a systematic and informed means to identify collection techniques, it also has limitations. For example, if an app uses a custom package without Java or Android method invocations, or if it uses a third-party service not included in our list, our categorization method may not accurately identify the collection technique used. Further details on the performance metrics are provided in the Appendix under Section B.

5 Fact-Checking Privacy Policy Claims

Upon completing the static analysis and organizing the privacy policy collection claims, we have the necessary foundation to perform a fact-checking analysis on these claims. The goal of this process is to detect any inconsistencies between the data collection practices described in the policy and the actual practices observed in the application code. The process unfolds in two stages:

5.1 Mapping Interaction Data Types and Collection Techniques

In the first stage, we create a mapping between the types of data outlined in the privacy policy and the equivalent interaction data types pinpointed during our static analysis. A similar mapping is constructed for each collection technique stated in the policy and the corresponding technique identified within the application code.

For instance, suppose a privacy policy declares, "*We collect the content you provide*", implying the collection of user-input data. During our static analysis, we identify the invocation of `EditText` elements in the application code, which signifies user input in Android. We then form a mapping between the phrase "*We collect the content you provide*" from the privacy policy and the `EditText` elements found in the code.

In another case, if the policy statement indicates, "*We track how long you spend on our services*", this suggests the usage of a duration-based collection technique. Suppose we identify the invocation of `android.os.SystemClock.elapsedRealtime()` in the code, which measures elapsed time, a mapping is established between the policy phrase "*We track how long you spend on our services*", and the this invocation in the code.

These mappings provide a basis for comparing the privacy policy's claims to the actual evidence in the code, allowing us to assess the consistency between policy declarations and the application code's actual practices.

5.2 Interaction Consistency Analysis

Having established the mappings, we can compare the data types and collection techniques from the privacy policy to those discovered in the code. This allows us to calculate the **Interaction Consistency Rate**, which measures the extent of consistency between the collection evidence identified in the static analysis (categorized by data type and collection technique) and the corresponding claims in the privacy policy. This rate represents the proportion of collection evidence found in the code that is accurately claimed in the policy.

An inconsistency may arise if, for example, our static analysis uncovers `EditText` invocations, but there is no mention of "user input data" in the app's privacy policy. Note that our analysis focuses on correlating claims made in the privacy policies with evidence gleaned from our static analysis. This means that if data collection is linked with a UI element that falls outside the scope of our static analysis, such collection will not be included in our investigation.

5.3 Context Consistency Analysis

The second stage of our analysis involves a context-based examination to comprehend when user interaction data is collected. Our motivation for conducting a context-based analysis is based on our preliminary observation from the APP-350 dataset, where 74% of the policy sentences related to user interaction data collection also described the context, for example, "We collect information on how you interact with our service *when you are making a purchase*."

To accomplish this, we review the app's code and identify unique contexts under which data collection takes place. The contexts we consider here are confined to those directly linked with identifiable criteria in the bytecode, thereby limiting our scope to certain discernible contexts.

Our approach for constructing a context catalog began with a careful selection of Android apps. We chose a representative sample of 25 apps from five distinct categories within the Google Play Store in Germany. [3] The categories selected were: "Social Networking", "Health & Fitness", "Entertainment", "Productivity", and "Finance". These categories were chosen for their popularity and the likelihood that they would handle a decent amounts of user interaction data. Each of these apps underwent a detailed static analysis. We scanned their bytecode for instances of user interaction data collection, focusing on the specific contexts in which this collection occurred.

Through this process, we identified and organized recurring contexts across the different apps. These common contexts, indicative of typical scenarios associated with user interaction data collection are developed into a generalized catalog. While not comprehensive, this catalog, as presented in Table 4, provides an informative overview of the most common user actions and apps states where interaction data collection is likely to occur.

Based on this catalog, we calculate the **Context Consistency Rate**, which measures the degree of consistency between the data collection contexts identified in the static analysis and those outlined in the privacy policy. This rate indicates the proportion of collection contexts found in the code that are accurately represented in the policy.

We recognize that our catalog cannot encapsulate all possible contexts due to the complexity and diversity of user interactions and app functionalities. Furthermore, our policy claim checks rely on language model-assisted vocabulary matching, which might not guarantee absolute precision. These factors should be considered when interpreting the Context Consistency Rate.

6 Experiment

In this section, we present the results of a large-scale analysis conducted on a set of 100 Android apps. Through this comprehensive examination, we aim to gain insights into the landscape of user interaction data collection practices as reflected in their privacy policies and underlying code. This analysis forms the basis of our discussion on the consistencies and discrepancies between policy claims and actual code execution.

[3] The German Google Play Store was selected for its adherence to the GDPR, ensuring that the apps included in the study would have well-constructed privacy policies. https://play.google.com/store/apps?hl=en_US&gl=DE

Table 4: Catalog of contexts for user interaction data collection

Context	**Identifiable Criteria in Code**
Viewing Content	Invocation of certain View UI elements (e.g., TextView/ImageView).
Making Purchase	Calls to Android Google Play payment service APIs.
Location-Based Services	Invocation of Android Location APIs.
Interacting with Media	Calls to media-related APIs (e.g., Media Player, Media Recorder).
Search	Invocation of SearchView UI elements.
Notifications	Interactions with NotificationManager API.
Accessing User Profile	Invocation of User Profile related APIs (e.g., AccountManager).
Sensor-based Features	Use of Android Sensor APIs.
Communication Features	Use of communication-related APIs (e.g., TelephonyManager).
Gameplay Interactions	Calls to APIs related to gameplay, typically seen in game apps.
Customization Features	Invocation of APIs related to customization (e.g., changing theme).

6.1 Setup

Our experimental analysis is based on a set of Android apps obtained from the Google Play Store in Germany. To ensure a comprehensive and varied dataset, we selected the top 100 apps from 10 distinct popular categories. These categories included varied domains such as “Lifestyle”, “Education”, “Travel”, and ’Entertainment’ among others, chosen for their relevance to a broad spectrum of users and potential data collection diversity.

We employed two key criteria for selecting these apps: (1) The categories and apps should be disjoint to avoid overlap and redundancy in our dataset. This approach was crucial to ensure that each app provided unique insights into user interaction data collection practices. (2) Every app must have a corresponding English privacy policy webpage linked in its “Data Safety” section. This criterion was essential to facilitate the analysis of privacy policies against actual app behaviors, and to ensure that all apps adhered to the GDPR. The chosen apps represented a mix of global popularity and regional relevance.

In this experiment, we assess the data’s consistency from privacy policies against static analysis results using two primary metrics: the Interaction Consistency Rate and the Context Consistency Rate, detailed in Section 5. These metrics measure the alignment of data types and collection contexts between policy claims and code evidence. We also introduce the **Interaction Consistency Coverage Rate** and the **Context Consistency Coverage Rate** to determine the completeness of our analysis, identifying any potential gaps in our static analysis method. These coverage rates help pinpoint areas not covered by our analysis, ensuring a thorough evaluation of privacy policy claims.

6.2 Overview of User Interaction Data Collection Practices

In our overview of 100 apps, illustrated in Fig. 2, indicate that 14% of the apps do not mention any form of user interaction data collection in their privacy policies. Approximately a third of the apps (29) acknowledge data collection but do not specify the type of data collected or the method of collection. These policies often contain general statements such as, “we use statistical tools to collect non-personal data such as usage details.” It is important to note that the more detailed policies tend to describe the types of data collected more than the methods of collection.

Our analysis showed an Interaction Consistency Rate of 58%, indicating how often app behaviors matched their policy claims regarding collected data types. The Context Consistency Rate was 32%, reflecting how well the context of data collection in the app code aligns with policy claims. These rates reveal a clear gap in transparency, with our static analysis capturing most data collection instances and contexts, evidenced by an Interaction Consistency Coverage Rate of 86% and a Context Consistency Coverage Rate of 71%.

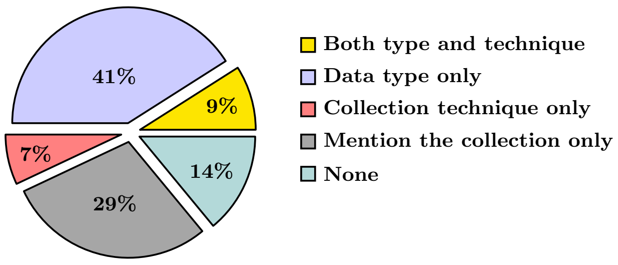


Fig. 2: Policy claims completeness regard to interaction data collection

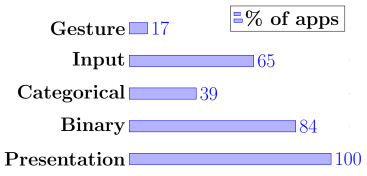


Fig. 3: Data type distribution

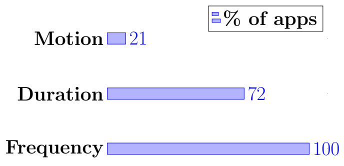


Fig. 4: Collection technique distribution

Figs. 3 and 4 display the distribution of interaction data types and collection techniques identified in our static analysis of the apps' code. The analysis shows that app presentation data and binary data, such as screen content and button clicks, are commonly collected. Moreover, the collection of user input data, particularly in relation to user preferences and surveys, is a frequent practice.

In terms of collection techniques, frequency and duration emerge as common methods. Notably, many apps do not disclose duration-based data collection in their privacy policies. This lack of mention further emphasizes the transparency issues in the way apps communicate their data collection practices.

Our analysis identified five categories of apps that most frequently engage in user interaction data collection: *social, entertainment, shopping, gaming*, and *lifestyle*. The extent of data collection in these categories can be attributed to two key factors. First, the intrinsic characteristics of the category, such as social networking and entertainment, necessitate understanding user behavior for the personalization of services. Second, the complexity of functionality in certain categories, like gaming, often requires learning from user interactions to optimize user experiences. Likewise, lifestyle apps might need to track user actions within the app to function effectively.

Note that almost all apps, across categories, engage in some form of user interaction data collection. However, the level of transparency in detailing such practices in their privacy policies varies widely. The majority of these policies lack completeness, indicating a trend of incomplete disclosure about user interaction data collection practices. This highlights the urgent need for more transparent and detailed communication about these practices in app privacy policies.

6.3 Case Study: In-depth Analysis of Four Popular Apps

We conducted an in-depth case study on four popular apps from the German Google Play Store to evaluate their privacy policies against actual data collection practices. Our selection included WetterOnline, Temu, Poe, and Plant App. Notably, except for Temu, the expected data collection scope for these services should be minimal. Yet, our static analysis revealed discrepancies in policy transparency, particularly in specifying data collection contexts. Table 5 provides an examination of their policy claims alongside our fact-checking results.

The analysis showed that while Temu was fairly transparent, the other apps were vague, often using broad terms such as "interaction with our service", which lacks detail. WetterOnline and Plant App were particularly limited in disclosing their data collection methods, and Poe's policy was almost silent on its data collection specifics. These findings highlight the critical need for clarity in privacy policies, especially since vague policies can mask practices that might lead to user profiling or re-identification, compromising user trust.

Moreover, the categorization of all user interaction data as non-personal and solely for commercial use is concerning. The extensive behavioral data collected could, when linked with unique identifiers, potentially be used to re-identify users. This underscores the urgent need for policies to more accurately reflect data use, aligning with our aim to ensure user privacy and trust in mobile apps.

7 Related Work

The analysis of mobile app privacy policies, particularly focusing on user interaction data, forms the core of our research, extending beyond the typical scope of existing studies. While prior research employing NLP techniques has significantly contributed to understanding privacy policies [13,9], our work uniquely concentrates on the nuanced aspects of user interaction data collection. Tools like PrivacyFlash Pro [15] and AutoCog [8] have laid the groundwork in aligning privacy policy claims with app behaviors, but their focus on personal data leaves a gap in addressing user interaction data, which our study aims to fill.

In the field of static analysis for app security and privacy [1,4,14], existing efforts have primarily analyzed app bytecode for data leaks and privacy violations without specifically targeting user interaction data. Our research contributes to this field by introducing an automated method that not only evaluates privacy policy disclosures but also correlates them with actual app behaviors concerning user interaction data. This approach not only enhances transparency but also fosters trust in the mobile app ecosystem, addressing a critical area that has been previously overlooked. Our work, thus, adds a dimension to the current understanding of mobile app privacy and automated data collection practices.

8 Conclusion and Future Work

Through this research, we investigated the collection of user interaction data in mobile apps, often claimed as anonymized but raising privacy concerns. Recent studies suggest that even anonymized data could be re-identified, challenging the idea of complete privacy protection. We developed a method to compare privacy policy statements with app behaviors, highlighting the need for transparent data collection practices. Our initial results demonstrate effectiveness in identifying the gap between stated practices and actual policy claims regarding interaction data collection, a discrepancy that could erode user trust.

Looking forward, expanding the research to include more apps and platforms will deepen our understanding of data collection practices. Future studies should also examine the classification of user interaction data, typically considered non-personal, and its potential impact on user profiling and privacy.

Table 5: Fact-checking data collection claims wrt. evidence for 4 popular apps. Red text means types of user interaction data missing from the privacy policy/collection claims, while blue text means undisclosed techniques of collection.

App	Policy Claims	Collection Evidence
Wetter Online	The goal of usage measurement is to determine the intensity of use, the number of uses and users of our application, and their surfing behavior statistically. The information about the use (..., the site visited, date and time of your visit. The event-driven data collection ... is triggered by activities such as installation and start of the app, ..., and in-app purchases as well as the receipt, the swipe and the opening of push-messages and the opening and updating of the app by means of a dynamic link. For each of these events the number of visits, the number of users triggering the event and, if available, the value of the events is collected.	Interaction Consistency Rate: data type 3/4; collection technique 1/2. Context Consistency Rate: 1/6. Data types: presentation, categorical, binary, user input. Collection techniques: frequency, duration. Context: viewing content, location, search, notification, sensor-based, customization.
Temu	Online activity data, such as pages or screens you viewed, how long you spent on a page or screen, the website you visited before browsing to the Service, navigation paths between pages or screens, information about your activity on a page or screen, access times and duration of access, and whether you have opened our emails or clicked links within them.	Interaction Consistency Rate: data type 3/5; collection technique 3/3. Context Consistency Rate: 1/10. Data types: presentation, categorical, binary, user input, gesture. Collection techniques: frequency, duration, motion. Context: viewing content, purchase, location, media, search, notification, user profile, sensor-based, communication, customization.
Poe	Our third party LLM providers and third party bot developers may receive details about your interactions with Poe (including the contents of your chats, upvotes, etc.) to provide and generally improve their services, which they may process in their legitimate business interests.	Interaction Consistency Rate: data type 1/3; collection technique 0/2. Context Consistency Rate: 1/6. Data types: presentation, binary, user input. Collection techniques: frequency, duration. Context: viewing content, location, search, notification, communication, customization.
PlantApp	During your visits, we may use software tools such as JavaScript to measure and collect session information including page response times, download errors, length of visits to certain pages, page interaction information (such as scrolling, clicks, and mouse-overs), and methods used to browse away from the page.	Interaction Consistency Rate: data type 2/5; collection technique 3/3. Context Consistency Rate: 1/8. Data types: presentation, categorical, binary, user input, gesture. Collection techniques: frequency, duration, motion. Context: viewing content, purchase, location, media, search, notification, sensor-based, customization.

Acknowledgement This paper is an extended version of work published in [12]. This work is part of the Privacy Matters (PriMa) project. The PriMa project has received funding from European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 860315.

References

1. Avdiienko, V., Kuznetsov, K., Gorla, A., Zeller, A., Arzt, S., Rasthofer, S., Bodden, E.: Mining apps for abnormal usage of sensitive data. In: The 37th IEEE international conference on software engineering. vol. 1, pp. 426–436. IEEE (2015)
2. Creţu, A.M., Monti, F., Marrone, S., Dong, X., Bronstein, M., de Montjoye, Y.A.: Interaction data are identifiable even across long periods of time. Nature Communications **13**(1), 313 (2022)
3. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding (2019)
4. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS) **32**(2), 1–29 (2014)
5. Grünewald, E., Pallas, F.: TILT: A GDPR-aligned transparency information language and toolkit for practical privacy engineering. In: Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency. pp. 636–646 (2021)
6. Leiva, L.A., Arapakis, I., Iordanou, C.: My mouse, my rules: Privacy issues of behavioral user profiling via mouse tracking. In: Proceedings of the 2021 Conference on Human Information Interaction and Retrieval. pp. 51–61 (2021)
7. Marda, V.: Non-personal data: the case of the Indian Data Protection Bill, definitions and assumptions (2020), https://www.adalovelaceinstitute.org/blog/non-personal-data-indian-data-protection-bill/, (Accessed on 28/11/2023)
8. Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T., Chen, Z.: Autocog: Measuring the description-to-permission fidelity in android applications. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 1354–1365 (2014)
9. Ravichander, A., Black, A.W., Norton, T., Wilson, S., Sadeh, N.: Breaking Down Walls of Text: How Can NLP Benefit Consumer Privacy? In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing. vol. 1 (2021)
10. Rzhevkina, A.: Several EU countries banned Google Analytics - here are some alternatives. https://www.contentgrip.com/eu-countries-ban-google-analytics/ (September 2022), (Accessed on 03/11/2023)
11. Singh, A., Raghavan, M., Chugh, B., Prasad, S.: The Contours of Public Policy for Non-Personal Data Flows in India (2019), https://www.dvara.com/research/blog/2019/09/24/the-contours-of-public-policy-for-non-personal-data-flows-in-india/, (Accessed on 28/11/2023)
12. Tang, F., Østvold, B.M.: Transparency in app analytics: Analyzing the collection of user interaction data. In: 2023 20th Annual International Conference on Privacy, Security and Trust (PST). pp. 1–10 (2023). https://doi.org/10.1109/PST58708.2023.10320181
13. Tesfay, W.B., Hofmann, P., Nakamura, T., Kiyomoto, S., Serna, J.: PrivacyGuide: Towards an Implementation of the EU GDPR on Internet Privacy Policy Evaluation. In: Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics. p. 15–21. IWSPA '18 (2018)
14. Zhang, X., Wang, X., Slavin, R., Breaux, T., Niu, J.: How does misconfiguration of analytic services compromise mobile privacy? In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. pp. 1572–1583 (2020)
15. Zimmeck, S., Goldstein, R., Baraka, D.: PrivacyFlash Pro: Automating Privacy Policy Generation for Mobile Apps. In: NDSS (2021)
16. Zimmeck, S., Story, P., Smullen, D., Ravichander, A., Wang, Z., Reidenberg, J.R., Russell, N.C., Sadeh, N.: Maps: Scaling privacy compliance analysis to a million apps. Proc. Priv. Enhancing Tech. **2019**, 66 (2019)

A Implementation of Claim Analysis Using BERT

We opted for BERT over GPT-3 for its bidirectional architecture, enabling a thorough contextual understanding of privacy policies, essential for our analysis. BERT's capacity to analyze both left and right sentence contexts is particularly effective for interpreting complex privacy policy language [3]. In our implementation, BERT was tailored to privacy policy language, involving pre-processing steps like tokenization and normalization, and trained on a specialized dataset to identify binary claims and data collection methods. We also employed bigram analysis to recognize common word pairs, augmenting the model's proficiency in interpreting policy language and thereby enhancing its precision and recall.

The model achieved a precision of 95% and a recall of 98% for claim extraction. For data types and collection methods, we observed precision and recall rates of 82% and 74%, respectively, and for collection techniques, precision and recall stood at 92% and 78%, showcasing the model's robust performance.

B Code Analysis and Performance Metrics

We selected 20 popular apps from the German Google Play Store, meticulously identifying each instance of user interaction data collection to establish a ground truth. Our static analysis method was then evaluated against this dataset.

Our method demonstrated high accuracy (91%), precision (92%), and recall (79%), with an overall F1-score of 85.5%, indicating effectiveness in accurately identifying and classifying user interaction data collection in mobile apps.

C Types of User Interaction Data & Collection Techn.s

We identified six types of user interaction data based on our analysis of Android UI widgets: App Presentation Data, Binary Data, Categorical Data, User Input Data, Gesture Data, and Composite Gestures Data. For a detailed explanation of these types, refer to [12]. Similarly, our study categorizes collection techniques as Frequency, Duration, and Motion Details. Each technique's specifics are also elaborated upon in [12].

PAPER 7

Finding Privacy-relevant Source Code

Tang, F. and Østvold, B. (2024). Finding Privacy-relevant Source Code. Accepted by The 2nd International Workshop on Mining Software Repositories Applications for Privacy and Security (MSR4P&S 2024).

Finding Privacy-relevant Source Code*

Feiyang Tang and Bjarte M. Østvold

Norwegian Computing Center
N-0314 Oslo, Norway
{feiyang,bjarte}@nr.no

Abstract. Privacy code review is a critical process that enables developers and legal experts to ensure compliance with data protection regulations. However, the task is challenging due to resource constraints. To address this, we introduce the concept of *privacy-relevant methods* — specific methods in code that are directly involved in the processing of personal data. We then present an automated approach to assist in code review by identifying and categorizing these privacy-relevant methods in source code.
Using static analysis, we identify a set of methods based on their occurrences in 50 commonly used libraries. We then rank these methods according to their frequency of invocation with actual personal data in the top 30 GitHub applications. The highest-ranked methods are the ones we designate as privacy-relevant in practice. For our evaluation, we examined 100 open-source applications and found that our approach identifies fewer than 5% of the methods as privacy-relevant for personal data processing. This reduces the time required for code reviews. Case studies on Signal Desktop and Cal.com further validate the effectiveness of our approach in aiding code reviewers to produce enhanced reports that facilitate compliance with privacy regulations.

Keywords: Personal Data Protection · Privacy · GDPR · Static Analysis · Code Review

1 Introduction

In the realm of software development, privacy code reviews have become indispensable, especially with the advent of stringent data protection regulations like the General Data Protection Regulation (GDPR). Unlike security code reviews, which focus on existing security flaws or vulnerabilities, privacy code reviews are concerned with the ethical and lawful handling of personal data. Although there may be overlaps, such as in access control, the primary objectives of these two types of reviews are distinct: security reviews aim to prevent unauthorized access, while privacy reviews aim for compliance with data protection principles.

Privacy code reviews involve a systematic process where source code is inspected to trace the flow of personal data. Equipped with program analysis tools, reviewers categorize these flows and detail how personal data is processed. This analysis serves as a comprehensive guide for compliance checks and aids Data Protection Officers (DPOs) in fulfilling their responsibilities. The process is illustrated in Figure 1. However, the challenge arises from the complexity and sheer volume of modern codebases, making it difficult to identify instances where personal data is processed.

Recent studies [6, 7] have examined tools for identifying personal data, but less focus has been placed on data that is dynamically changing or in active use. While categorizations exist for personal data itself, taxonomies of the processing code are lacking. Developing a understanding of the diverse ways data can be handled would illuminate processing activities and facilitate compliance reporting like records of processing activities (ROPA) and data protection impact assessments (DPIA). Since reviewing entire codebases is time-consuming, targeting reports to highlight the most relevant aspects could better serve reviewers and streamline the compliance process. The goal should be

* Accepted by *The 2nd International Workshop on Mining Software Repositories Applications for Privacy and Security* (MSR4P&S 2024).

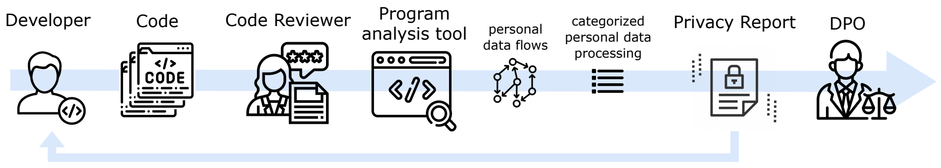


Fig. 1. Privacy code review process

providing clarity on key data handling activities without getting lost in an elaborate labeling framework.

In light of these challenges, we propose an automated approach to enhance the efficiency and effectiveness of privacy code reviews. Our approach focuses on identifying *privacy-relevant methods* — specifically, Java methods or JavaScript functions commonly found in popular libraries — that are involved in the processing of personal data. By doing so, we can pinpoint instances in real-world applications where these privacy-relevant methods are invoked to handle personal data.

This paper addresses the following research questions:

1. How to identify privacy-relevant methods in commonly used libraries that potentially process personal data?
2. How to categorize such privacy-relevant methods based on their actual usage in real-world applications?

To answer these questions, we make the following contributions:

1. We present a novel static analysis technique specifically designed to identify methods in source code that are involved in the processing of personal data. (Section 4)
2. We develop a set of labels for categorizing personal data and the methods that process them, thereby providing a structured approach to understanding how personal data is processed in code. (Sections 5 and 6)
3. We apply our approach to a set of popular open-source applications. Through this, we rank privacy-relevant methods based on their frequency of occurrence, thereby identifying those that are most critical for privacy considerations. (Section 7)
4. We provide insights to code reviewers by highlighting frequently used methods relevant to privacy, based on our large-scale study and specific case studies. This approach streamlines the review process, enabling a more focused and efficient identification of potential privacy risks. (Section 8)

Our evaluation of 100 open-source applications indicates that our approach identifies fewer than **5%** of methods involved in personal data processing as privacy-relevant methods.This enables reviewers to focus only on the identified relevant code, thereby expediting privacy code reviews.

2 Background

Code review, originally aimed at ensuring software quality by identifying bugs and performance issues [11], has expanded to address security vulnerabilities and, more recently, privacy concerns under data protection laws like the GDPR. Privacy-focused reviews add the complexity of ensuring personal data is handled lawfully and ethically, a challenging task due to the often ambiguous nature of data protection guidelines [10].

Static analysis tools are pivotal in code reviews, aiding in the identification of data flows, security risks, and compliance issues. The effectiveness of a review is measured by its ability to pinpoint critical problems and offer actionable solutions. Privacy code reviews, however, struggle with identifying personal data due to unclear definitions and varied contexts, increasing reliance on these tools despite their limitations in recognizing diverse personal data types [9].

These reviews also play a key role in creating essential compliance documents like Records of Processing Activities (ROPA) and Data Protection Impact Assessments (DPIA). The proposed automated approach in this paper focuses on improving the efficiency and accuracy of privacy code reviews, specifically in categorizing personal data processing in large-scale code projects.

3 Privacy-Relevant Methods

To streamline the process of privacy code review, we introduce the concept of *privacy-relevant methods*. These are specific methods that play a direct role in the processing of personal data. Such methods can be part of standard libraries or third-party libraries, making them critical focal points for personal data processing in software applications.

Native libraries are foundational because they offer the only pathways to device resources like files and networks. Consequently, any operation involving data storage or transfer must go through these native methods. Native privacy-relevant methods are those found in standard libraries of programming languages like JavaScript and Java. These methods act as the origins (sources) for all personal data entered by users via devices. They are also the exclusive methods that directly transmit this data to other devices or services. We categorize these native methods into domains such as *I/O, Database, Network, Security*, following the guidelines of existing research [8]. We identify these methods through a systematic manual review that includes an examination of documentation, source code, and actual usage patterns.

To facilitate the identification and categorization of native privacy-relevant methods, we conducted an in-depth analysis of key modules like `java.io`, `java.security`, and `java.util` for Java, and their equivalents in JavaScript. This analysis helps us compile a complete set of native privacy-relevant methods, denoted as *Native*, that are involved in personal data processing.

4 Identifying API Privacy-relevant Methods

Native privacy-relevant methods form the basis for identifying what we refer to as API privacy-relevant methods. These are methods found in third-party libraries and frameworks that are likely to process personal data by calling upon native privacy-relevant methods. Understanding the relationship between API and native methods is crucial for a complete review of how personal data is processed in a codebase. The identification process is iterative and takes into account the dependencies between libraries and codebases, as depicted in Fig. 2. The goal is to assemble a list of API privacy-relevant methods that have the potential to handle personal data. Understanding the relationship and dependency hierarchy among these libraries is essential for accomplishing this task.

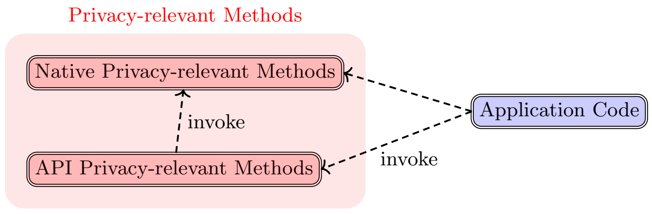


Fig. 2. The relationships between privacy-relevant methods and application code

4.1 Dependency Sorting and Identification of Privacy-relevant Methods

To manage library dependencies, we focus on import statements within each library's source code. We organize the libraries in a sequence such that each library is evaluated only after all its dependencies have been assessed. This ensures a logical and efficient evaluation process.

For the identification of API privacy-relevant methods, we define a set denoted as API. This set includes methods from our organized list of libraries that invoke native privacy-relevant methods at some point during their execution. These methods are significant as they interact with native methods, either directly or through a chain of calls, making them critical for privacy code review.

5 Labels for Personal Data Processing

Compliance with data protection regulations like GDPR necessitates a nuanced understanding of how personal data is processed within code. While GDPR outlines various processing activities such as collection, recording, and organization, the four native privacy-relevant method categories [8] we previously discussed (*I/O*, *security*, *database*, and *network*) lack the granularity needed for comprehensive understanding. For instance, the *security* category encompasses both authentication and encryption, warranting a more detailed labeling system.

After analyzing top labels from Maven and NPM that pertain to personal data processing, we identified 20 labels that closely align with both GDPR's definitions and our native privacy-relevant method categories. This shows how libraries handle data processing in different ways. For example, *OAuth* combines *network* and *security* functionalities, while *Object-Relational Mapping (ORM)* bridges *database* and *I/O operations.* These overlaps underscore the necessity for a detailed set of labels tailored for privacy reviews. We present these labels and their alignment with GDPR requirements in Table 1.

These labels serve a dual purpose: they categorize methods involved in data processing activities like collection, storage, and encryption, and they map these activities to GDPR compliance requirements. This streamlined mapping simplifies the task of identifying code sections that need to comply with legal standards. In our later approach, we use these labels to prioritize privacy-relevant methods, enabling a focused review on areas critical for data protection.

6 Process of Identifying Personal Data

Before delving into the approach, it is crucial to differentiate between personal data and personally identifiable information (PII). While both are subsets of information that relate to an individual, PII is a category of data that directly identifies a person. Examples include account information, contact details, personal IDs, and national IDs. Not all the 10 categories of personal data we consider below fall under PII. The exposure of PII is especially concerning as it could lead to personal or psychological harm, such as identity theft.

Our primary aim is to identify the flow of personal data within a codebase, focusing on its cruicial implications for privacy. To achieve this, we use a pattern-matching technique inspired by Tang et al. [**?**]. This technique effectively identifies data from 10 categories, including *Account*, *Contact*, *Personal ID*, *Location*, and *National ID.* We employ Semgrep, a tool tailored for pattern matching in code, to facilitate this process. Semgrep's rules are specifically designed for Java and JavaScript languages.

6.1 Static Analysis for Personal Data Identification

The initial phase of our approach involves using static analysis to locate code fragments that contain personal data. We use Semgrep for this task, given its efficiency and flexibility in analyzing large codebases. We rely on Semgrep's support for multiple languages and its capabilities for local data flow analysis.

Table 1. Alignment of the labels with GDPR requirements

Category	GDPR Alignment
Identity and Access Management (IAM): managing users' identities and regulating their access to resources. It is based on the libraries that perform authentication and access control.	Article 32: Robust measures, including authentication.
Data Encryption and Cryptography (DEC): cryptographic operations enhancing the security and privacy of personal data. DEC specifically targets data encryption.	Article 32: Data pseudonymization and encryption.
Data Storage, Management, and Deletion (DSMD): handling the storage, retrieval, and deletion of personal data. DSMD extends the "*database*" category to incorporate data deletion methods, focusing on the lifecycle management of personal data in storage systems.	Article 5(1)(e): Data retention only as long as necessary.
Data Processing and Transformation (DPT): carry out transformations or processing on personal data, including anonymization, aggregation, and other forms of data manipulation. DPT also includes Object/Relational Mapping methods that facilitate the conversion between incompatible type systems in object-oriented programming languages and databases.	Article 30: Mandatory record of processing activities under responsibility.
Network Communication (NC): methods that send or receive personal data over a network, focusing on personal data transmission.	Article 44: Controlled data transfer.
Logging and Monitoring (LM): methods that handle the recording, monitoring, and retrieval of log entries that may contain personal data. It emphasizes the tracking and auditing activities that involve personal data.	Artical 5(1)(c): Principle of data minimization. Article 5(1)(e): Data retention only as long as necessary.

6.2 Defining Sources of Personal Data

In the context of our analysis, sources refer to instances where personal data appears. We identify personal data in two ways: 1) as literal text present in the source code, and 2) as variables, based on their name identifiers. Our identification rules are designed to support Java, JavaScript, and TypeScript but can be extended to other languages that Semgrep supports.

6.3 Rule Crafting for Identification

To pinpoint literal personal data, we use regular expression (regex) matching. This comes into play, for example, when identifying the format of national ID numbers. For variable sources, we maintain a default list of identifiers that correspond to the 10 categories of personal data. These identifiers help us formulate Semgrep rules. To reduce false positives, we impose specific conditions on these regex rules. For instance, to capture all human names in the code, we use a regex pattern that accommodates variations like first, last, and full names: `(?i).(?:first|given|full|last|sur(?!geon)) [s/(;)|,=!>]name)`.

7 Data-based Ranking of Privacy-relevant Methods

Our data-based ranking is designed to identify and prioritize privacy-relevant methods in Java and JavaScript applications. This ranking process comprises several stages, as depicted in Fig. 3, using

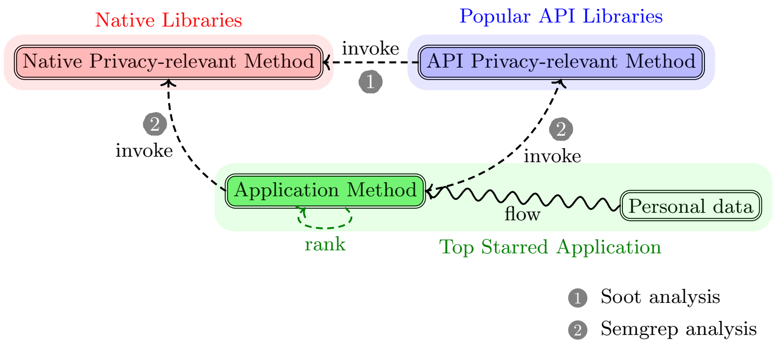


Fig. 3. Overview of the Java ranking. The circled numbers represent different static analysis tools used for the analysis step. Soot was applied to Java bytecode, while Semgrep was used for source code analysis.

the Java ranking as an example. By analyzing data from real-world applications, we aim to provide a practical guide for identifying methods that are most relevant for privacy concerns.

7.1 Library Selection for Data-based Ranking

To focus our data-based ranking on the most relevant libraries, we selected the top 25 libraries from NPM for JavaScript and Maven for Java, shown below in Table 2. Our selection criteria were based on the libraries' relevance to personal data processing, as aligned with our set of labels for personal data processing activities. This selection was made through a systematic review of each library's documentation, specifically targeting functionalities that are related to personal data processing.

7.2 Method Invocation Analysis

We employed static analysis tools to identify method invocations and analyze data flows within the code. For Java, we used Soot [14] to construct call graphs and trace method invocations. In the case of JavaScript, we used ESLint [1] for its capabilities in Abstract Syntax Tree (AST) analysis. Our analysis matched these invocations to our list of native privacy-relevant methods, providing a view of how these methods are used in practice.

7.3 Selecting Open-source Applications

To rank privacy-relevant methods, we selected 30 popular open-source GitHub projects with over 100 stars in Java and JavaScript. We focused on applications processing personal data rather than frameworks and libraries.

The selection included 15 Java applications such as the e-commerce software Shopizer, and 15 JavaScript applications like the chat application RocketChat. We also included projects predominantly in Java/JavaScript that use other languages like TypeScript for some modules. Criteria were: popularity (applications with high stars, indicating broader relevance), data sensitivity (applications processing personal or sensitive data, highly relevant for privacy reviews), diversity (applications from different domains and languages, showing wide applicability), and public availability (open source code enables reproducibility and transparency). The details of these selected projects are provided in Table 4.

[1] `https://eslint.org`

Table 2. Selected popular libraries: 25 for each language

Category	Maven Libraries (Java)	NPM Libraries (JavaScript)
Identity and Access Management (IAM)	Keycloak, Apache Hadoop Auth, GRPC Auth, Identity API, CAS Server Core Authentication API	Google Identity Platform, @azure/identity, Passport.js, jsonwebtoken, bcrypt.js
Data Encryption and Cryptography (DEC)	Bouncy Castle, Jasypt, Apache Shiro, Nimbus JOSE+JWT, Cryptacular	scrypt-js, Bcryptjs, Jsonwebtoken, node-rsa, openpgp
Data Storage, Management, and Deletion (DSMD)	H2 Database Engine, Spring Data MongoDB Core, PostgreSQL JDBC Driver, Apache Cassandra, MongoDB Driver	Sequelize, Mongoose, Knex.js, nedb, pg (node-postgres)
Data Processing and Transformation (DPT)	Hibernate, MyBatis, Apache Spark, Spring Batch, MapStruct	Prisma, Ramda, Immutable.js, async, moment
Network Communication (NC)	Netty, Apache HttpComponents, OkHttp, Retrofit, HttpClient	Axios, Request, Socket.IO, node-fetch, WebRTC
Logging and Monitoring (LM)	Log4j, slf4j, Logback, Apache Commons Logging, jboss-logging	Log4js, Morgan, Winston, Bunyan, Pino

7.4 Efficient Analysis of Library Imports

To make the analysis efficient, we first identified the libraries imported by each application. For standard libraries, we assumed their presence in most applications. For API libraries, we examined import statements and configuration files to narrow down our focus to the top 50 pre-selected libraries, 25 each for Java and JavaScript.

7.5 Ranking Privacy-relevant Methods in Top 30 Applications

We employed Semgrep to monitor the flow of personal data into privacy-relevant methods invoked by application code. Utilizing Semgrep's DeepSemgrep [2] capability for cross-file analysis, we were able to comprehensively analyze data flows across entire applications, as opposed to only examining isolated code snippets. This provided a holistic perspective of how personal data propagates across different components.

Using Semgrep's taint analysis and the rules outlined in Section 6, we traced personal data flows to privacy-relevant methods.

To assess the practical relevance of our identified privacy-relevant methods, we introduce the following usage-based metrics, presented in Table 3:

We ranked privacy-relevant methods by analyzing their usage in the 30 popular GitHub projects introduced above, with an average of 358 application methods processing personal data per application. This varied by language and type: Java applications averaged 288 methods, while JavaScript had 363. The higher average in JavaScript was likely due to its more diverse front-end processing, reflecting the complexity and multifaceted nature of these applications.

[2] `https://semgrep.dev/blog/2022/introducing-deepSemgrep/`

Table 3. Usage-Based Metrics for Ranking Privacy-relevant Methods

Metric	Description
Method Occurrence	Enumerates the overall instances a method is invoked across applications, offering insights into its regularity of processing.
PII-Related Method Frequency	Measures the proportion of times a method interacts with PII, highlighting its involvement with sensitive data.
Category Occurrence	Captures the aggregate appearances of a particular category across applications, revealing the ubiquity of distinct processing modalities.
PII-Related Category Frequency	Assesses the percentage of methods within a category that engage with PII, reflecting its sensitivity quotient.

To better focus our approach, we calculated the proportion of application methods that both invoke a privacy-relevant method and process a concrete flow of personal data (there is confirmed personal data flow into the method). This is relative to the total number of methods in the application. This metric indicates the level of focus in identifying privacy-relevant methods, allowing developers to narrow their efforts to a more relevant subset of the code. In essence, our approach aims to minimize the code sections that need scrutiny, saving both time and resources. For more details on these proportions in selected open-source Java and JavaScript/TypeScript applications, see Table 4.

Table 4. List of 30 selected open-source applications written in Java and JavaScript (JS)/TypeScript (TS), along with their descriptions. And the calculated percentages of application methods that invoke identified privacy-relevant methods and are involved in the concrete flow of personal data, relative to the total number of methods in each application.

Lang.	Project Name	Description	|AM|/|Total|	Prop.
Java	Apache James	A mail server fully written in Java.	531/18,332	2.9%
Java	Apache OFBiz	A product for the automation of enterprise processes.	376/10,448	3.6%
Java	DSpace	A turnkey institutional repository application.	141/5,769	2.4%
Java	Broadleaf	An eCommerce platform based on the Spring Framework.	591/11,586	5.1%
Java	Shopizer	A web-based Java eCommerce software.	492/10,318	4.7%
Java	OpenMRS	A platform that enables the design of a medical records system.	336/8,621	3.9%
Java	Apache Nutch	A highly extensible and scalable web crawler software project.	18/2,194	0.8%
Java	JabRef	An open source bibliography reference manager.	154/9,621	1.6%
Java	Apache Roller	A Java-based full-featured, multi-blog, multi-user server.	112/1,983	5.6%
Java	Apache Camel	A framework integrates systems consuming/producing data.	198/20,471	0.9%
Java	Keycloak	An identity and access management for apps and services.	843/17,562	4.8%
Java	OpenCms	A professional level website content management system.	446/13,932	3.2%
Java	Waltz	A web app managing the architectural landscape of enterprises.	11/2,093	0.5%
Java	H2O	A distributed, fast, and scalable ML and analytics platform.	20/14,231	0.1%
Java	RapidMiner	A data science platform with an integrated environment.	58/7,949	0.7%
JS	Ghost	A fully adaptable platform for building online publications.	400/6,452	6.2%
TS	Jitsi Meet	A WebRTC JS application for scalable video conferences.	226/3,905	5.8%
TS	KeystoneJS	A scalable platform and CMS to build Node.js applications.	145/1,882	7.7%
JS	Reaction	A commerce platform built using Node.js and GraphQL.	462/4,921	9.4%
TS	Rocket.Chat	A free open-source solution for team communications.	490/12,841	3.7%
JS	Strapi	An open-source Headless CMS Front-End.	347/6,796	5.1%
JS	Gatsby	A framework based on React helping build websites and apps.	241/10,042	2.4%
JS	Etherpad	A modern real-time collaborative document editor.	82/2,175	3.7%
TS	Vue Storefront	An open-source frontend for any eCommerce.	428/6,291	6.9%
TS	Mattermost	A platform for secure collaboration across the entire SDLC.	784/18,231	4.3%
JS	Apostrophe CMS	An in-context CMS built on Node.js and MongoDB.	99/1,896	5.2%
JS	Expensify	An app of financial collaboration centered around chat.	511/6,721	7.6%
JS	Wiki.js	A modern and powerful wiki app built on Node.js.	36/1,194	3.0%
TS	AFFiNE	A knowledge base that enables planning, sorting and creating.	1066/12,845	8.3%
TS	Boostnote	A note-taking app made for programmers.	134/4,956	2.7%

7.6 Findings

Our study reveals that, on average, only **4.2%** of the total codebase is made up of methods that are privacy-relevant and involved in personal data processing. This result highlights the precision of our approach in pinpointing privacy-relevant methods in applications.

Usage Patterns of Privacy-Relevant Methods In Java applications, we observed a more conservative use of privacy-relevant methods, particularly those from popular Maven libraries. Native Java methods, along with methods from Apache Commons and the Spring framework, were frequently used for handling personal data. Libraries such as `slf4j` for logging and `auth0` for authentication were also commonly used, indicating their importance in the flow and protection of personal data.

In contrast, JavaScript applications exhibited a diverse range of library usage. While `lodash` was commonly used, frameworks like Angular, React, and Vue.js played a significant role in personal data processing, particularly in front-end applications.

Table 5 presents the top five packages in both Java and JavaScript that contain methods relevant to privacy concerns.

Table 5. Top 5 packages defining privacy-relevant methods

Java	JavaScript
1 java.*	lodash
2 auth0	mongoose
3 slf4j	React
4 Spring (security, http)	Angular
5 Hibernate	axios

Categories of Privacy-relevant Methods We categorized privacy-relevant methods into types to gain insights into their roles in personal data processing. Our analysis identified several Java classes and categories that are frequently involved in personal data processing. For example, common Java classes like `org.slf4j.Logger` and `auth0.client.Auth0Client` are often used in operations that handle personal data.

In terms of categories, *Data Processing and Transformation*, *Network Communication*, and *Logging Methods* were most prevalent. These categories indicate areas where privacy-relevant methods are most commonly used, suggesting that they are key to understanding how personal data is processed in codebases (Table 6).

Identity and Access Management, *Data Encryption and Cryptography*, and *Data Storage and Database Management* were also highly involved in personal data flows, with involvement percentages of 92%, 78%, and 85%, respectively. Conversely, categories like *Data Processing and Transformation*, *Network Communication*, and *Logging Methods* were less involved, with percentages of 67%, 44%, and 28%. Table 7 lists Java classes that are frequently involved in personal data processing, serving as key indicators for identifying privacy-relevant methods in applications.

8 Application to Privacy Code Review

This section outlines how our approach can be applied to privacy code reviews across a diverse set of 100 open-source applications. We then delve into detailed case studies of two popular software applications to illustrate the utility of our approach.

Table 6. Top 3 categories of privacy-relevant methods and PII-related privacy-relevant methods

Java LM	**Count**	**JavaScript LM**	**Count**
1 DPT	1,946	DPT	2,455
2 LM	1,422	NC	1,871
3 NC	860	LM	1,019
Java PII-LM	**Count**	**JavaScript PII-LM**	**Count**
1 DPT	769	DPT	1,032
2 DSMD	351	DSMD	318
3 NC	307	IAM	596

Table 7. Top classes in Java for personal data processing with example privacy-relevant methods

Library	**Top Classes**	**Top Privacy-relevant Methods**
Commons	org.apache.commons.io.IOUtils	`IOUtils.read(InputStream i, byte[] b)`
Commons	org.apache.commons.io.FileUtils	`FileUtils.readFileToString(File f, String name)`
Auth0	auth0.jwt.JWT	`JWT.decode()`
Auth0	auth0.client.Auth0Client	`Auth0Client.login()`
SLF4J	org.slf4j.Logger	`Logger.info(String format, Object... arguments)`
SLF4J	org.slf4j.LoggerFactory	`LoggerFactory.getLogger(Class<?> class)`
Spring Sec	*.security.core.Authentication	`Authentication.getPrincipal()`
Spring Sec	*.security.web.FilterChainProxy	`FilterChainProxy.doFilter()`
Spring HTTP	*.http.HttpEntity	`HttpEntity.getBody()`
Spring HTTP	*.http.client.ClientHttpRequestFactory	`ClientHttpRequestFactory.createRequest()`
PostgreSQL	*.core.Connection	`Connection.createStatement()`
PostgreSQL	*.jdbc.PreparedStatement	`PreparedStatement.executeQuery()`

8.1 Large-scale Analysis

To understand the prevalence and types of personal data processing in real-world applications, we analyzed 100 open-source applications. These were equally divided between Java and JavaScript/TypeScript and were selected from GitHub's daily top-starred repositories list [3].

We selected applications that are popular (top-starred), non-trivial (over 300K lines of code), and predominantly written in Java or JavaScript/TypeScript (constituting over 60% of the codebase). Additionally, we ensured these applications differed from the 30 popular libraries analyzed previously and that their primary documentation language was English for easier identification of functionalities. This selection process resulted in a dataset that is representative of real-world software applications and suitable for our analysis of personal data processing practices. We then examined the proportion of methods in these applications that invoke privacy-relevant methods and are involved in the flow of personal data and Personally Identifiable Information (PII). The result of statistics of our findings are listed below in Table 8.

Table 8. Percentage of application methods invoking privacy-relevant methods and processing personal data and PII

Language	**Personal Data**	**PII**
Java	3.6%	1.9%
JavaScript	5.1%	3.8%

Our findings indicate that our approach can make the privacy code review process more efficient. By identifying methods that are critical for personal data and PII processing, we help reviewers focus their efforts, enabling a more targeted review.

[3] `https://github.com/EvanLi/Github-Ranking` (captured on 18/06/2023)

8.2 In-Depth Case Studies

We validate the effectiveness of our approach through two open-source projects: Signal Desktop[4] and Cal.com[5]. Each offers unique insights for privacy code review.

Both projects were chosen due to their popularity, sensitivity, and public availability. Their open codebases ensure transparency and reproducibility, making them ideal candidates to validate our approach. By applying our approach to these carefully selected real-world projects, we provide concrete examples that demonstrate practical value in identifying key areas to focus on during privacy code reviews.

Signal Desktop Signal Desktop is a famous end-to-end encrypted messaging application, primarily written in TypeScript (79.5%) and JavaScript (15.6%), covering about 360K lines of code. Its reputation for enhanced security and privacy features showcases the depth of our approach. While the application has limited use of popular libraries, our approach highlighted a minor number of privacy-relevant methods invocations (48, approximately 0.5% of total methods) from our selected APIs and native libraries potentially linked to personal data processing.

In our analysis, Signal stands out for using its own encryption protocol (Signal Protocol) and message transmission services, minimally relying on external libraries. This underscores Signal's commitment to end-to-end encryption. Our categorization highlights the primary areas of Data Processing and Transformation (DPT), Network Communication (NC), and Data Encryption and Cryptography (DEC), with most encryption methods used for local encryption of profiles and group data. Signal's proprietary protocol, used for encrypting chats and attachments, falls outside our analysis scope.

Our findings show that Signal rarely transmits PII directly to the internet. Instead, encrypted system data or anonymized IDs are mainly used, reflecting Signal's dedication to user privacy.

For privacy code reviewers examining Signal Desktop, our approach underscores Signal's limited use of popular libraries for PII processing, aligning with its privacy-focused design philosophy. This categorization helps reviewers understand how Signal handles personal data, aiding in a more streamlined review process.

Cal.com Cal.com, a scheduling application, is designed to grant users comprehensive control over their schedules. Written entirely in TypeScript, it spans about 126K lines of code. Our method identified 371 (approximately 3.8% of total methods) privacy-relevant methods that might engage in personal data processing.

Applications such as Cal.com often employ diverse frameworks for specific functionalities. For instance, Cal.com's utilization of the popular ORM framework, Prisma, for handling user profiles and credentials, aligns with our library list. In terms of categories, Data Processing and Transformation (DPT) topped the list at 26%, followed by Identity and Access Management (IAM) at 17%, and Network Communication (NC) at 15%. Unlike Signal Desktop, Cal.com heavily leverages libraries like `Prisma`, `next-auth`, and `nodemailer` for processing personal data, mirroring its primary functions of user registration, email interaction, and scheduling.

Approximately 97% of privacy-relevant methods invoked by Cal.com handle PII. This attests to the capability of our method in identifying PII processing methods and subsequently guiding code reviewers efficiently.

Our approach highlights the extensive use of specific libraries in applications like Cal.com, aligning with their core features. This correlation boosts reviewers' confidence and precision. By categorizing processing activities, it provides an overview of how the application handles personal data, helping reviewers prioritize effectively. This makes the review process time-efficient and thorough.

[4] `https://github.com/signalapp/Signal-Desktop`
[5] `https://github.com/calcom/cal.com`

8.3 Threats to Validity

Our study's validity may be affected by several factors. The project selection based on GitHub trends could bias towards popular topics, potentially overlooking a broader range of applications. The use of Semgrep for static analysis, though efficient, hasn't been thoroughly validated for precision, which could impact the accuracy of our results. Reliance on regular expression matching for identifying personal data risks introducing false positives and negatives, thus affecting result reliability. Additionally, the absence of manual validation for each instance of personal data processing identified might lead to inaccuracies. Furthermore, focusing only on the top 25 libraries for Java and JavaScript due to resource constraints limits the generalizability of our findings, as other privacy-relevant methods in lesser-known libraries may have been missed.

9 Related Work

Research in source code analysis for privacy is extensive, yet specific approaches for identifying personal data processing are limited. Ullah et al. [13] introduced an approach for extracting control and data dependencies in source code, potentially applicable for locating personal data processing methods, but not directly designed for this purpose. Hjerppe et al. [2] proposed an annotation-based static analysis for data protection, but its effectiveness is contingent on accurate developer annotations, a challenge in large projects.

Dynamic analysis has been explored for sensitive data flow detection, with DAISY [15] focusing on Android apps and ConDySTA [16] combining dynamic taint analysis with static analysis. However, these methods have limitations, such as platform specificity or the need for executing projects. Automated assistance in code review has been explored by Li et al. [3] with their pre-trained model CodeReviewer, but it lacks a focus on personal data processing.

SWANAssist [5] offers a semi-automated approach for identifying security-relevant Java code methods, which could potentially be adapted for privacy purposes. Other studies, like [1, 12], attempt to align GDPR compliance with static analysis. Novikova et al. [4] provided insights into privacy-enhancing technologies but did not focus on personal data processing in source code.

These studies mark great progress in source code analysis, yet a gap exists in automated identification and categorization of personal data processing. Our work addresses this by proposing an automated approach for identifying personal data processing in real-world applications, enhancing efficiency in privacy code reviews.

10 Conclusion and Future Work

In conclusion, our study introduces a method for identifying and categorizing privacy-relevant methods in source code, focusing on personal data processing. We have successfully narrowed the analysis scope to just 4.2% of methods across 100 popular open-source applications, offering a practical starting point for developers, data protection officers, and reviewers. This approach not only simplifies code reviews but also facilitates compliance with data protection regulations like GDPR, helping organizations align their software development with legal requirements.

For future work, we aim to enhance the precision of our privacy-relevant method identification algorithms, possibly integrating machine learning for more accurate predictions of personal data processing activities. Expanding our approach to additional programming languages and integrating it into common development tools for real-time feedback are also key goals. These advancements will broaden the impact and applicability of our approach. Ultimately, our research paves the way for more focused and efficient privacy assessments in software development, contributing to the creation of software that is efficient, robust, and respectful of user privacy.

Acknowledgement

This work is part of the Privacy Matters (PriMa) project. The PriMa project has received funding from European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 860315.

References

1. Ferrara, P., Olivieri, L., Spoto, F.: Tailoring taint analysis to GDPR. In: Privacy Technologies and Policy: 6th Annual Privacy Forum, APF 2018, Barcelona, Spain, June 13-14, 2018, Revised Selected Papers 6. pp. 63–76. Springer (2018)
2. Hjerppe, K., Ruohonen, J., Leppänen, V.: Annotation-based static analysis for personal data protection. In: Privacy and Identity Management. Data for Better Living: AI and Privacy, pp. 343–358. Springer International Publishing (2020)
3. Li, Z., Lu, S., Guo, D., Duan, N., Jannu, S., Jenks, G., Majumder, D., Green, J., Svyatkovskiy, A., Fu, S., Sundaresan, N.: Automating code review activities by large-scale pre-training (2022)
4. Novikova, E., Fomichov, D., Kholod, I., Filippov, E.: Analysis of privacy-enhancing technologies in open-source federated learning frameworks for driver activity recognition. Sensors **22**(8), 2983 (2022)
5. Piskachev, G., Do, L.N.Q., Johnson, O., Bodden, E.: SWANAssist: Semi-Automated Detection of Code-Specific, Security-Relevant Methods. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering. p. 1094–1097. ASE'19, IEEE Press (2020). https://doi.org/10.1109/ASE.2019.00110
6. van der Plas, N.: Detecting PII in Git commits (2022), `http://resolver.tudelft.nl/uuid:fe195c17-ecf5-4811-a987-89f238a6802f`
7. Ren, J., Rao, A., Lindorfer, M., Legout, A., Choffnes, D.: ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic. In: Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services. p. 361–374. MobiSys '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2906388.2906392
8. Tang, F., Østvold, B.M.: Assessing Software Privacy Using the Privacy Flow-Graph. In: Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security. p. 7–15. MSR4P&S 2022, Association for Computing Machinery, New York, NY, USA (2022)
9. Tang., F., Østvold., B., Bruntink., M.: Identifying Personal Data Processing for Code Review. In: Proceedings of the 9th International Conference on Information Systems Security and Privacy - ICISSP. pp. 568–575. INSTICC, SciTePress (2023). https://doi.org/10.5220/0011725700003405
10. Tang, F., Østvold, B.M., Bruntink, M.: Helping Code Reviewer Prioritize: Pinpointing Personal Data and Its Processing. IOS Press (Sep 2023). https://doi.org/10.3233/faia230228
11. Thongtanunam, P., Hassan, A.E.: Review dynamics and their impact on software quality. IEEE Transactions on Software Engineering **47**(12), 2698–2712 (2020)
12. Tokas, S., Owe, O., Ramezanifarkhani, T.: Static checking of GDPR-related privacy compliance for object-oriented distributed systems. Journal of Logical and Algebraic Methods in Programming **125**, 100733 (2022)
13. Ullah, F., Wang, J., Jabbar, S., Al-Turjman, F., Alazab, M.: Source code authorship attribution using hybrid approach of program dependence graph and deep learning model. IEEE Access **7**, 141987–141999 (2019)
14. Vallée-Rai, R., Co, P., Gagnon, E., Hendren, L., Lam, P., Sundaresan, V.: Soot: A java bytecode optimization framework. In: CASCON First Decade High Impact Papers, pp. 214–224 (2010)
15. Zhang, X., Heaps, J., Slavin, R., Niu, J., Breaux, T., Wang, X.: DAISY: Dynamic-Analysis-Induced Source Discovery for Sensitive Data. ACM Trans. Softw. Eng. Methodol. **32**(4) (May 2023)
16. Zhang, X., Wang, X., Slavin, R., Niu, J.: ConDySTA: Context-Aware Dynamic Supplement to Static Taint Analysis. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 796–812 (2021). https://doi.org/10.1109/SP40001.2021.00040

PAPER 8

Software Privacy and Program Analysis: Insights, Methods, and Opportunities

Tang, F. and Østvold, B. (2024). Software Privacy and Program Analysis: Insights, Methods, and Opportunities. Submitted to the Springer Handbook: Privacy and Security Matters in Biometric Technologies.

ISBN 978-82-326-7758-0 (printed ver.)
ISBN 978-82-326-7757-3 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)