MagnetDroid: a Bridge between Security, Privacy, and the Law for Android Applications

Emanuele Uliana

Submitted in fulfillment for the degree of Doctor of Philosophy

Department of Computer Science

Royal Holloway, University of London

December 18, 2021

Declaration of Authorship

I, Emanuele Uliana, hereby declare that this thesis and the work presented in it are entirely my own. Where I have consulted the work of others, this is always clearly stated.

December 18, 2021

Emanuele Uliana

Acknowledgements

I wish to express my gratitude to everyone who made my PhD experience and this thesis possible. First of all, my primary supervisor Kostas Stathis who pushed me when I needed a push, listened to me when I needed to talk, and guided me thanks to his knowledge, experience, and wisdom. There are no words to describe the support I got from you when I was plagued by technical problems, and in a terrible mood, due to the still ongoing pandemic. Thank you!

I am grateful to my co-supervisor from the School of Law Robert Jago who inspired me to dive into the world of legislation on data protection, and provided an invaluable support when I was confused in the world of legal systems, rules, and regulations.

I am grateful to Leverhulme and Royal Holloway University of London for the Magna Charta scholarship that allowed me to get by while working on my PhD research.

I am grateful to my fellow PhD students of the DICELab for supporting me on my journey since they joined the lab, exchanging ideas on challenging subjects, and providing feedback both on my research, and this thesis. In rigorous alphabetical order: thank you Pallavi Bagga, Joel Clarke, Nausheen

Saba Shahid, and Benedict “Ben” Wilkins.

I am grateful to Royal Holloway University of London for accepting me as a PhD student, therefore allowing me to leave Italy, which was a long-time goal of mine. I am grateful to my old supervisors at Politecnico di Milano - Federico Maggi and Stefano Zanero - for letting me develop part of my Master thesis at Royal Holloway under Lorenzo Cavallaro whom I also owe much. That was the start of the process that brought me here.

I am grateful to all the members of the “Phish’n’Chips” and “TU6PM” CTF teams. We shared some amazing moments, including the privilege of testing our collective security knowledge and skills by competing in several Capture The Flag event teams. Among them, I must highlight my long-time friend Claudio Rizzo who was also an invaluable source of inspiration for my research. We spent some amazing days together! Special thanks to Roberto Jordaney who also helped my in my research during his time as a PhD student at Royal Holloway. I am sure I am going to forget someone (I apologise for that), yet thank you Giovanni Cherubin, Feargus Pendelbury, Giulio De Pasquale, Eamonn Postlethwaite, James Patrick-Evans, and Fernando Virdia.

I am also grateful to a number of PhD students with whom I had time to exchange ideas of various subjects: Duncan Mitchell, Blake Loring, Thomas Van Binsbergen, Marie-Sarah Lacharité, Mateo Torres, Diego Galeano Galeano, Juan Cáceres Silva, and Nery Riquelme Granada. Once again, I apologise, if I forgot some of you.

I am grateful to those who still live in Italy, yet remained in frequent contact with me during my PhD years. Thank you Giulia Desteffani, Luca Fanton, Dino Desteffani, and Antonella Ziliani.

Finally, I am grateful to my parents Mario Uliana and Ornella Gavinelli, and to my grandparents Aldo Gavinelli and Amedea Sarasini for accepting my life choices, and encouraging me to pursue my objectives.

Abstract

We study the problem of privacy of user data in relation to the behaviour of Android applications. The problem is of interest not only to common users who regularly use Android applications, but also to application developers who aim to provide users with a secure enough final product. It is also relevant to legal professionals, who are involved in litigations about privacy of data with mobile devices. In that regard, we believe it is also relevant to developers and publishers, even though it is often overlooked by them.

We view privacy as intrinsically dependent on security. As a result, the thesis discusses how insufficient security impacts the private data of Android users at the application level and granularity. To assess the problem in a concrete scenario, we develop MagnetDroid, a novel agent-based framework that infers privacy-related legal violations and consequences that stem from insufficient security in the context of Android applications. Our main goal is to provide a fact-based interpretation capable of linking security with the relevant law on data protection.

We first present a unifying framework on the application analysis side, where agents translate and aggregate reports from different analysis tools under a common ontology defining the security issues that Android applications can

trigger. As a result, the framework delivers a structured collection of identified security issues (contained in a so called final report) which we refer to as Technological Knowledge Base (TKB). The application-dependent TKB is used in conjunction with a general model of the relevant law on data protection, which we call Legal Knowledge Base (LKB). We then reason on the union of TKB and LKB by performing queries to derive legal violations and infer legal consequences. The results are then presented in natural language format to link security issues with legal violations and consequences.

MagnetDroid takes full advantage of the communication and coordination abilities of a multi-agent platform, which we experimentally evaluated on a dataset of applications from AndroZoo. We conclude by discussing how our approach can be applicable to non-Android and non-mobile scenarios as well, given the appropriate ontologies.

Glossary of terms

Android application analysis tool: Any software that analyses an Android application and produces a report.

APK: The file of an Android application.

ASO: Short for Android Security Ontology.

LKB: Short for Legal Knowledge Base.

(Raw) report: The product of an application analysis tool.

Technological: Anything related to Android application analysis, as opposed as related to the law.

TKB: Short for Technological Knowledge Base.

Contents

1 Introduction **13**

1.1 Background 14

1.1.1 The privacy problem 16

1.2 Thesis Objectives 18

1.3 Thesis Contributions 19

1.4 Structure of the thesis 21

1.5 Previous publications 21

2 Background and State of the Art **22**

2.1 Privacy Background 23

2.1.1 Philosophical privacy 23

2.1.2 Privacy in legal systems 29

2.1.3 Privacy in Computer Science 37

2.1.4 Working definition of Privacy 41

2.2 Security Background for Android 43

2.2.1 Platform level security 45

2.2.2 Application level security 45

2.2.3 Security of network communications 47

2.2.4 Security of data at rest 57
2.2.5 Security and data leaks 58
2.2.6 Defense-in-depth techniques 59
2.3 Security analysis for privacy 60
2.4 Limitations of the State-of-the-Art 63
2.4.1 Limits of security analysis 63
2.4.2 Limits of the law 66
2.4.3 Obstacles to integration 66
2.5 Summary 67

3 MagnetDroid: Design and Architecture 69
3.1 Platform assumptions 70
3.2 Goals and aims 74
3.2.1 Cyberlegal privacy 74
3.2.2 Prerequisites for cyberlegal privacy 76
3.3 MagnetDroid design 80
3.3.1 High level architecture 80
3.3.2 A platform for integrative analysis and synthesis 82
3.3.3 A reasoning platform 96
3.3.4 A webapp to close the ring 102
3.4 Summary 103

4 MagnetDroid Agent Platform 105
4.1 The opportunity of an agent Platform 106
4.1.1 Key Agent Properties 106
4.1.2 Practical examples 109

4.2 Agent Model and Architecture 111
4.2.1 Agent capabilities 113
4.3 High level structure 114
4.3.1 MAP entities 114
4.3.2 MAP agent protocol 125
4.4 Phase 1: parallel analysis 128
4.4.1 The flow 128
4.4.2 The algorithm 129
4.4.3 Raw reports 131
4.5 Phase 2: translation 132
4.5.1 Parsing 133
4.5.2 ASO instantiation 137
4.6 Phase 3: aggregation 140
4.6.1 The flow 141
4.6.2 The algorithm 141
4.6.3 Conflict solving mechanism 144
4.6.4 The final report 146
4.7 MAP extensibility and agility 147
4.8 Summary 148

5 MagnetDroid Reasoning Module 149
5.1 Phase 4a: creation of a TKB 150
5.1.1 From leaves and state variables to Prolog rules 151
5.1.2 The algorithm 153
5.2 Phase 4b: creation of a LKB 153
5.2.1 The 2018 UK Data Protection Act 154

5.2.2 From the articles to a Legal Knowledge Base 157
5.3 Phase 5: reasoning 160
5.3.1 Queries 160
5.4 MWA: Interacting with MagnetDroid 163
5.4.1 Parameterised Analysis and Queries 164
5.4.2 Visualisation of the Results 164
5.5 Summary 166

6 Discussion 167

6.1 Justification of the High Level Approach 168
6.2 The significance of ASO 169
6.2.1 Absolute Relevance Index 169
6.2.2 The Impact Indexes 170
6.2.3 The Nature of DNS Issues 172
6.3 Experimental Evaluation: MAP 174
6.4 Experimental Evaluation: MRM 176
6.5 Summary 179

7 Conclusions and Future Work 181

7.1 Summary of the Thesis 181
7.2 Future work 183

Appendix A: MAP agents actions 185

Appendix B: Raw Report Example 190

Appendix C: Pseudo-grammar Example 193

Appendix D: Example of a TKB complementing a LKB **196**

Chapter 1

Introduction

At the centre of our Internet dependent lifestyles, we rely more and more on a large network of devices that we use to carry out our daily activities. The software application running on these devices often need to know *personal identifying data* [1], which may violate our privacy if exposed. As a result, there is a growing interest over privacy. Moreover, the increasingly shifting demographics perceive practices such as the abuse of digital systems for the sake of non-explicitly authorised user profiling, selling of personal information to advertisers, and mass surveillance [2] as unacceptable. All of these are consistently regarded as detrimental to individuals, whether persons or organisations, in terms of their rights and freedoms. In particular, there seems to be a common understanding of a universal right to retain exclusive control of certain pieces of information about oneself, whether digital or not [3].

1.1 Background

Consider a metaphorical enclosure where everything inside needs to remain private, while, on the other hand, we assume that what remains outside could become public, at some point. Such a definition highlights our first issue with the notion of privacy: there is no universal agreed perimeter for the enclosure, because it is a subjective matter [4]. Therefore, for instance, it is difficult to write any kind of legislation with precise limits on the matter [5]. In practice, what happens is that societies derive the extent of the enclosure from the particular set of moral values developed over time within them. This process is not unique to privacy. In fact, arguably that is how the law originates and gets updated [6]. The law reflects, at least to some extent, the set of moral values that are accepted (either spontaneously, or by coercive means, such as a dictatorship) by the majority of the population of a certain society. It is apparent that, due to the proliferation of different human societies, many different (often incompatible) sets of moral values exist today in the world. As such, it is not surprising, therefore, to find widely different laws on privacy in different countries.

Another important issue is that new disruptive technologies generate both new kinds of "information", and new ways to access, use, and abuse it. Thanks to computing devices such as smartphones, more and more aspects of our lives are encoded into data. Some data reflect pieces of personal information that we may want to retain exclusive knowledge of. Some other apparently harmless data (e.g. our location at a specific time of date) can be aggregated into a "profile" from which more personal information can be inferred (e.g. whether we play tennis every Sunday) [7]. Those data and information are powerful. On average, whoever has more data on a subject makes better decisions related to that subject. Therefore, there is a strong push towards the collection of the largest amount possible of data [8], even though the obtained information might not be immediately useful. Think about a nation state spying on its own citizens or foreign nationals. For a political party, having access to the profiles of a large number of voters allows the development a more effective electoral strategy. In a cynical way, we could say that exploiting the fears of a large group of voters while campaigning becomes notably easier when the profiles of those people are available. Or perhaps a corporation whose revenue depends on customers buying their products. Having access to the profiles of a large number of perspective customers is an obvious advantage for such corporation, as it allows tailored advertising, for instance. In other words, there are strong incentives from different angles to reduce the size of the enclosure as much as possible. However, the push is not infinitely powerful. The abuse of private data in different digital domains has generated a reaction, namely, demanding for a better protection for private information [9].

Our thesis constitutes a part of the push-back against what we perceive as an excessive push towards a smaller enclosure. However, we aim to be a line of defense against a more subtle threat against privacy. If the enclosure has holes or weak sections, no push-back will save private data, because an external attacker could just mount a targeted effort against the weak point(s), and still gain access to the private data. Out of metaphor, our thesis is based on the basic assumption that privacy is the first victim of insufficient security. Even more so, privacy is *impossible* in absence of a sufficient level of security in the context of whoever and whatever is managing our private data. By now, it is abundantly clear that *privacy of individuals* nowadays is almost synonymous with *privacy of their data*. Therefore, this kind of privacy is our main focus.

1.1.1 The privacy problem

Having established privacy of data as our main focus, and insufficient security as its biggest threat, we recognise that the problem is still open. We choose to focus our attention on mobile operating systems, because we carry mobile devices with us for a significant amount of time, and we entrust them with a large amount of personal data. Among the mobile operating systems we restrict the domain even more, and focus on Android, both because its open source nature makes it an easier subject for research than its closed source counterparts, and because it is the most popular and used mobile OS. Within Android we choose to ignore the security issues that are intrinsic to or derive from the Android OS, focusing our attention only on Android applications. Therefore, we study the problem of privacy with respect to Android applications as pieces of software that can perform actions and take decisions.

As already mentioned, insufficient security is a sufficient condition to undermine privacy. In that context, we broadly categorise Android applications in two groups: vulnerable applications and malicious applications. From the technological point of view, once a vulnerable or malicious application has been developed and published, the standard line of defence consists of attempting to detect malicious behaviours and/or vulnerabilities. Unfortunately, as it stands, this approach exhibits some significant shortcomings, which constitute the core of what we characterise as the privacy problem.

First, most of the tools that are designed to detect vulnerabilities and malicious behaviours produce reports that are meant for humans. As such, these reports are not machine-friendly, and hinder a fully automated approach which is necessary for the detection to work at scale, given the enormous amounts of applications and the scarcity of security professionals.

Second, there is an apparent lack of coordination between the technological response to the privacy problem (i.e., the analysis tools), and the legal response. As much as technology can help identifying security issues, without an appropriate legislative framework, it becomes all but impossible to force developers to think about security when designing and building their applications. Likewise, a deterrent is required to at least partially prevent the damage that malicious actors cause when they publish malicious applications. The main reason that hinders the cooperation between technology and the law in terms of mitigating the insufficient security of Android application is a crucial mismatch. The law is written in natural language by legal experts with little knowledge of security. On the other hand, analysis tools (and Android application, for what matters) are developed by professionals with little knowledge of the law. As such, given the current differences between the two domains, the lack of a bridge between the two is not surprising.

The apparent lack of a bridge between technology and the law is the main motivation for this thesis. As we explore the causes of it, the mostly uncoordinated and machine-unfriendly nature of the technological response becomes another driving force. It is clear to us that, before a bridge can be built, we need to lay a solid foundation on both sides. As such, we take interest in creating unity in the analytical response to insufficient security, and in ensuring that such unity enables a machine friendly toolchain to be built, as well as the aforementioned bridge. On the legal side, our interest is captivated by how the inherently technological terminology (e.g., data confidentiality, etc.) is defined and/or referenced within the law. Once both sides' shortcomings have been addressed, we take interest in building a bridge that provides an interpretation linking security issues in Android applications with legal violations and legal consequences.

1.2 Thesis Objectives

Motivated by the privacy problem described above, our hypothesis is that it is possible to develop a bridge between application analysis and the relevant law. The bridge will be developed via process of raising the level of application analysis to legal concepts, and by formulating the law as a logic program, so to check individual applications for compliance. Given this hypothesis, the main aim of the thesis is to develop a generic framework for evaluating the privacy of Android applications, in the light of insecure practices. In this context, the key objectives of our work are to:

- Use existing analysis tools and identify new ones to provide more thorough technical analyses for Android applications.
- Formalise computationally a subset of privacy legislation and link them to concepts of the technical analysis.

- Identify suitable integration mechanisms that allow us to coordinate results from analysis tools, reasoning with privacy legislation, and running applications to test whether privacy is compromised within experiments.
- Perform a series of experiments to demonstrate the use of the framework.

1.3 Thesis Contributions

Given the privacy problem, and our motivation to address it, our contributions to its mitigation are as follows:

- We designed an ontology that specifies a standard syntax and semantics for Android application analysis reports. Named *Android Security Ontology* (ASO), it encompasses the relevant security issues that users of Android applications can face, and analysis tools are able to identify and report.
- We designed a framework that, given an Android application, a set of available analysis tools, and ASO:
  - Runs the tools in parallel on the application, and collects the reports.
  - Translates the reports into an ASO compatible structure via a standard procedure that concentrates agility in a few points (in order to minimise the additional coding needed to accommodate for new tools).

  - Aggregates the translated reports into an ASO compatible final technological report that can be used for further analysis.
  - Outputs a *Technological Knowledge Base* (TKB) from the final report that can be used for reasoning purposes.

- We implemented the framework as a multi-agent platform, in order to take advantage of the cooperation, coordination, and communication properties that are intrinsic to agent platforms.
- We produced a model of a subset of the relevant law on data protection, and derived a *Legal Knowledge Base* (LKB) from it.
- We designed and implemented a reasoning module that, given the TKB and the LKB, allows for reasoning on their union, as well as querying for legal violations, and legal consequences.
- We designed a human-friendly interface to the system that allows for selective querying, and explains the results in comprehensible terms for three types of users: developers, regular users, and legal experts.

We call the collection of frameworks MagnetDroid. In particular, we call the sub-framework responsible for generating the final report *MagnetDroid Agent Platform* (MAP), and the sub-framework that enables reasoning *MagnetDroid Reasoning Module* (MRM).

1.4 Structure of the thesis

The remainder of the thesis is structured as follows. In Section 2 we discuss the relevant background on privacy, security, Android, and the law. In Section 3 we describe the design of ASO, and our approach to MagnetDroid as a whole system. In Section 4 we describe the implementation of MAP. In Section 5 we describe how we modelled the law, and our MRM. In Section 6 we evaluate our contributions both in terms of compatibility with existing standards, and via two rounds of experiments. Finally, in Section 7 we summarise our work, and discuss the limitations of MagnetDroid, how it can be improved, and future research directions.

1.5 Previous publications

Some of the components of MagnetDroid (namely, *ASO*, *MAP* and *MRM*) have already been published in [10] for what concerns their design, architecture, and proto-implementation.

Chapter 2

Background and State of the Art

In this Chapter we explore the relevant background to the thesis contributions, and we establish the common ground terms and concepts required in order to understand our contributions. We start by exploring privacy from different points of view. In Section 2.1.1 we describe how the concept of privacy has been shaped by philosophical thought throughout history. In Section 2.1.2 we describe how privacy is currently framed within different legal systems, and we establish what for us constitutes the relevant law on data protection. In Section 2.1.3 we explore what privacy means in the context of Computer Science, and we establish security as a necessary condition for privacy. Then, in Section 2.1.4 we provide our working definition of privacy. We explore security (and therefore privacy) in the context of Android in Section 2.2 by concentrating on Android applications (Section 2.2.2), their network communications (Section 2.2.3), and their treatment of data at rest

(Section 2.2.4). In Section 2.2.5 we discuss the practical consequences of insufficient security of Android applications on privacy. We continue in Section 2.3 with the presentation of the current state of the art techniques to analyse Android applications in the context of security. Finally, we identify the limitations of the state of the art in Section 2.4, especially the limits of analysis, the limits of the law, and the obstacles to the integration of the two.

2.1 Privacy Background

In this Section we discuss the background surrounding the concept of privacy from different points of view. In Section 2.1.1 we focus on the philosophical view. In Section 2.1.2 we focus on legal systems. In Section 2.1.3 we focus on Computer Science.

2.1.1 Philosophical privacy

Privacy has no universally agreed definition. Multiple attempts have been made throughout history to characterise it from different points of view. Early attempts predated both written laws and computer technology. Consequently, they were mainly focused on social behaviours and philosophical disquisitions. The former case mainly consists in attempts at identifying what constitutes a normal social behaviour. In the latter case, topics for debate typically include what may or may not be considered private, under which conditions, and with which exceptions. When written laws came into existence, lawmakers attempted to formalise some of the results of the philosophical discussions into written clauses. History brought technology advancements, at various rates, and in different periods. However, in the 20th and 21st centuries this pace experienced a notable acceleration. Many technological advancements in the last 200 years have demonstrated that the previous definitions of privacy are crumbling, and no more applicable. In particular, new technologies can either allow behaviours that were physically impossible before (e.g., remote connections from home with far away places in the world), or create new objects that may fall under the span of privacy (e.g., digital data). On the other hand, advancements may be used to improve the level of privacy, once a definition consistent with the historical period has been adopted. Along with technology, the philosophical discussion on the meaning of privacy and on its subjects of applicability has accelerated too, in the attempt to keep apace. The whole process consists of a cycle which repeats itself. When technology improves, the definition and the span of privacy changes. Whenever the definition and/or the span of privacy evolve, new philosophical discussions arise. Consequently, lawmakers and legislators try to incorporate the results of those debates. New laws contribute to new advancements in technology, and so on. Effectively, there is a perpetual cycle of philosophy, law, and technology taking their turn in defining privacy, working with its definition, and expanding it.

Understanding the past to predict the future The current iteration of the aforementioned cycle consists of a world where most of the daily routine would be impossible as it is without mobile devices. As mobile devices contain more data about people than people often realise and remember, it seems natural to consider any breach in the management of those data as a privacy-related problem. Philosophical discussions about privacy in a technological world are already frequent and widespread [11]. The next expected step is for lawmakers to incorporate the results of those discussions into relevant rules and regulations. In order to understand how this process works and might work in the future, we need to explore how that happened in the past.

Before the modern era From the beginning of the modern human (pre-)history, around 200,000 years ago, to the beginning of the modern era, privacy was in its infancy. We know that primitive humans already understood the concept of dignity [12], especially with respect to actions like sleeping. In Ancient Greece the desire to be let alone in determinate circumstances played a role in the construction of residential dwellings. Greeks had a sophisticated understanding of Geometry, and they built their houses by leveraging their knowledge, so that rooms had the maximum exposure to sunlight, while being minimally exposed to the sight of external viewers. In Ancient Sparta the first known attempt at providing a private form of communication took place: the Scytale [13]. Within the ancient roman civilisation, we can find another example of an attempt at securing communications against unauthorised third parties: the so called Caesar's Cipher [14]. The advent of Christianity in Europe, first within the Roman Empire, then during the Middle Ages, brought the concept of seclusion: isolation from the society, in order to fight against the inner demons, so not to sin anymore. The most evident effect of this line of thought was the creation of the monastic orders.

Another notable contribution of Christianity to the philosophical concept of privacy was the mandatory secretiveness of the confession sacrament [15]. Every single word needed to remain between the person and the priest. This is likely because Christians then believed and still believe that a third party, the christian divinity, was listening to the entire conversation as the only intended recipient, while the priest was only an intermediary. Towards the end of the Middle Ages, another privacy-related practice, the one-person-one-bed accommodation, became common. Albeit this can be regarded as a privacy matter, the real reason behind that is probably the need to improve the hygiene in hospitals when the Black Death surged in 1348. We do not know about relevant laws about privacy that were written before the modern era. However, the customs that we described can be considered part of the foundations of the laws on privacy that would have been written in the subsequent centuries.

The modern era before the computer revolution In the Modern Era, technology started to raise concerns for the secrecy of certain activities. The beginning of the 17th century saw the invention of the telescope, which could be used to gain a better view of far away objects and activities. One of the consequences was and is that, in case of espionage, the victim could and can be totally unaware of it. The 19th century saw the invention of the telegraph. For the first time, a reliable transmission of data between distant endpoints became possible. With every technology advancement, people started to value more the right not to be subjected to intrusions in their private properties.

By the advent of the two industrial revolutions, when products and services became available to the masses, the need for private spaces and private activities became a major and widespread line of thought across Europe. The belief was that the common well-being could not be achieved without the right for individuals and/or family to live in a separate and healthy, possibly owned, house in which they could perform activities without being exposed to external interferences. It is during this period that historians agree the right to privacy (or modern privacy) was born [16]. Privacy was expected to become a political right, and scholars from universities were involved in the discussion. Warren and Brandeis work in 1890 on the right to privacy is an example of an attempt at building a general concept of privacy which could incorporate technological advancements [16]. One of the most significant contributions in the pre-computer era dates back to 1883. In that year the Dutch linguist Auguste Kerckhoff formulated his eponymous principle which is considered one of the foundations of modern cryptography, and which is still valid today [17].

With the increasing pressure for a legal recognition of the right to be involved in secret activities, the matter started to be seriously discussed by lawmakers. In 1710 the British Parliament approved the Post Office Act [18] which made illegal in the North American colonies for post office operators to look at the contents of mails while sorting them. In 1903, in New York a form of identity theft, namely the use of someone's unauthorised likeness for commercial purposes, was outlawed [19].

The computer revolution and the Internet era Soon after World War II, the television and computer era began. It became possible to program machines and to instruct them to perform tasks in a fast way with a lowered effort and risk for humans. The advent of ARPANET before, and Internet after, made possible for computers to transfer large amount of data. With fiber-optics the speed of data transfer can be up to 2/3 of the speed of light in the vacuum. Devices which could capture pictures, sounds and motion events could be connected to computers, and the captured data could travel the world in a matter of seconds. The implications for privacy are huge. Different threads of discussion arose in response of the major technological revolutions of the 20th and 21st Centuries. One of them regards surveillance of people by the public authority. The main element of uncertainty is where to put the threshold between what is considered normal and necessary for crime prevention, evidence discovery, and global well being, and what is considered a useless and unfair access to other people or entities private data, conversations and behaviours. Computer scientists and engineers mostly agree that mass surveillance and any intentional government-mandated weakening of the security of digital devices always do more harm than good. To cite some, Turing Award recipients Ronald Rivest, Adi Shamir, Whitfield Diffie, Martin Hellman, and the main developer of the Signal Protocol Moxie Marlinspike all agree on these matters [20].

Among computer scientists a major concern is to protect data confidentiality and integrity, and availability of/for digital systems, as well as ensuring the authentication of peers in network communications. The fields of modern cryptography, (digital) system security, network security, transport security, and cyberphysical systems security were all born in order to fulfil these needs.

2.1.2 Privacy in legal systems

Among legal scholars, William Prosser had the intuition that, in order to properly include privacy in the law (Tort's law in its case), a more systematic approach was required. In its work in 1960 [21] Prosser identified four different rights to privacy, as he called them:

1. A person's right to seclusion or solitude, and to have private affairs.
2. A person's right not to have personal embarrassing private facts disclosed to the public.
3. A person's right not to be put under a false light in the public light.
4. A person's right not to have someone else appropriating of their likeness for a personal advantage.

While Prosser's attempt at systematically reorganising the definition of privacy is important (for the systematic approach), it groups together concepts that are too different. The first point is indeed relevant for privacy in a technological world, but the others are much less. Points number 2 and 3 are more about defamation than privacy (even though in some jurisdictions only the third point can be regarded as defamation [22]). The last point falls under the identify theft problem.
In response to the philosophical debate, every country has since incorporated into their legal system rules and regulations regarding privacy, along with a definition of it. In the next paragraphs we are going to examine 4 instances:

UK (more specifically, England and Wales, as Scotland and Northern Ireland have their own jurisdictions) in Paragraph 2.1.2.1, the European Union in Paragraph 2.1.2.2, the United States of America in Paragraph 2.1.2.3, and China in Paragraph 2.1.2.4. The word privacy in these sub-paragraphs refers to its meaning in the specified legal context. We mainly focus on the UK and EU, citing U.S.A. and China for comparison purposes. In particular, we chose the U.S.A. as the representative for multi-party democracies, and China as the representative for one-party countries. This approach is modelled after the *archetype vs deviant example* approach described by Francis Pakes in its book Comparative Criminal Justice [23]. China, in our case, is the deviant example.

2.1.2.1 Privacy in the United Kingdom

The United Kingdom (UK) does not have a unique legal system. However, the relevant pieces of legislation on privacy mostly apply throughout the whole country. According to the case of *Wainwright v Home Office* [24], the general concept of invasion of privacy is beyond the scope of UK common law. Therefore, there cannot be a cause of action for it under English common law. However, in 1998, the promulgation of the Human Rights Act resulted in the incorporation into the UK domestic law of the *European Convention on Human Rights* [25] (ECHR). According to Art. 8(1), of the ECHR:
Everyone has the right to respect for his private and family life, his home and his correspondence.
According to Art. 8(2), the public authority cannot interfere with such right, unless to protect higher rights/values. These are also stated in Art. 8(2):

- National Security.
- Public Safety.
- Economic well-being of the country.
- Prevention of disorder or crime.
- Protection of health or morals.
- Protection of the rights and freedom of others.

Also, any interference, as specified in Art. 8(2), must be fully justified in the law at each Country level.

The ECHR is a European Convention. As such, it has been incorporated into the laws of other European countries, and laws are constantly promulgated in accordance with the principles described in both Art. 8(1) and Art. 8(2). For instance, regarding the protection of health, in 2017 both Italy and France passed national laws which prescribe 10 to 11 mandatory vaccinations to all children, in order for them to be able to attend pre-school. A vaccination record is also created and shared between the local and national health systems. This is an example of a lawful interference in individuals' privacy for a greater and global benefit, since those laws were passed in response to a noticeable drop in vaccination rates. This drop lead to the loss of the so called herd immunity, and, consequently, to a widespread outbreak of measles in Italy in 2017 [26].

In the UK laws in accordance with ECHR allow, for example, luggage inspection and body scans at the airport(s). In this case, the matters are National

Security, Public Safety, Prevention of crime, and Protection of health (e.g., in case of illegal transport of hazardous materials).

The problem with Art. 8 of the ECHR is the vagueness of the terms: each of the higher rights/values require a proper definition. To date, there is no universally accepted definition for any of those, which means that each country has to define them in the corresponding domestic law. This is prone to a number of abuses. The limit between useful checks and practices like individuals humiliation (e.g., public searches and seizures) and immoral mass surveillance is all but clear. The facial scanners at Heathrow airport are an instance of the problem: are they an anti-terrorism measure, or a mere mass surveillance tool? Again, the answer is different, depending on who you ask to.

As mentioned in Section 1.1.1, and reinforced in Section 2.1.3 we are mainly interested in data privacy. For this reason, we are particularly interested in the legislation on data protection. In this thesis, we choose to focus on the legal system of England and Wales. Within it, the single most important piece of legislation on data protection is the *2018 Data Protection Act* (2018 UK DPA from now on, or even simply DPA). While we have little reason to doubt that the legislators acted with their best intentions and in good faith, in order to ensure the best legal protection possible to users' data, and to specify reasonable consequences for violations, the DPA has limits. We discuss them in Section 2.4.2.

2.1.2.2 Privacy in the European Union

As of February 2020, the UK is no more a member of the European Union (EU). However, the 40+ years long membership has ensured that EU legislation affected the domestic legal systems within the UK. For this reason, we also examine how privacy is regarded in the EU legal system. Privacy in the member countries of the European Union is regulated by the following two documents proposed by the European Commission in 2012 and approved by the European Parliament in 2016:

- *Regulation* on the protection of natural persons with regard to the processing of personal data and on the free movement of such data [27].
- *Directive* on the protection of natural persons with regard to the processing of personal data by competent authorities for the purposes of the prevention, investigation, detection or prosecution of criminal offences or the execution of criminal penalties, and on the free movement of such data [28].

Both documents officially repeal previous regulations (Directive 95/46/EC) and directives (Council Framework Decision 2008/977/JHA) on the matter of privacy. Regarding the *Regulation*, EU countries' domestic laws cannot be in contrast with it, much like other EU regulations. Additionally, at the time of approval, all EU countries were expected to incorporate the *Directive* by 2018. Since all EU members are European countries, their national laws usually incorporate the ECHR as well.

Article 4 of the *Regulation* states both that the processing of personal data should be designed to serve mankind, and that the right to the protection of personal data is not an absolute right: it must be considered in relation to its function in society and be balanced against other fundamental rights, in accordance with the principle of proportionality. The fundamental rights are described in the *Charter of Fundamental Rights of the European Union* [29] and in the *Treaty on the Functioning of the European Union* [30]. The underlying concept is that there is a fundamental right to the protection of personal data (seen as a component of privacy), but other fundamental rights (see above) can override it in case of necessity. In particular, everything regarding a specific person, from their identity, to the result of their actions, is treated as data. While the EU regulations and directions are among the most conservative on the matter of personal data protection, they suffer from the same vagueness which threatens Art. 8 of ECHR and its applications.

2.1.2.3 Privacy in the United States of America

Begin a federal republic, the United States of America has multiple legal systems at multiple levels [31]. There are a central federal government (led by the President of the U.S.A.) and parliament (the Congress) which promulgate and enact federal laws in accordance with the Constitution of the U.S.A.. At the federal level, laws are either general-purpose or of national interest (e.g., national security). Below the federal level there are the 50 States. Each of them has a State Constitution, a Governor, and a State Parliament. For this reason, it is correct to say that each State has its own legal system. However, the States cannot promulgate laws on subjects of national interest, and cannot declare a federal law invalid. They can, however, legalise something which is illegal at the federal level - e.g., the recreative use of marijuana, etc.

The existence of multiple legal systems is relevant to the subject of privacy, because at the federal level and at the level of the single States it is possible to have discrepancies on the regulation of privacy.
We are mainly interested in the federal level. The Constitution and its amendments, ensure some fundamental rights, such as:

- The right of free assembly (First Amendment).
- The right to be free of unwarranted search or seizure (Fourth Amendment).

Those rights are complemented by the generally accepted principle of *The right to be let alone*, which, in theory, ensures that a free assembly cannot be monitored and arbitrary secret searches and seizures cannot take place [16] [32]. However, the aforementioned general principle has some limitations:

- It does not apply to the matters of public interest.
- It can be overridden by higher interests.
- It is weaker than its European counterparts, since, as already written, single States can override the Federal law to a greater extent than European and EU countries can with respect to European and EU regulations and directives.

The right to be let alone, also, partly ceases to apply when an individual voluntarily agrees to the terms and conditions of a particular service, such that those explicitly allow some degree of intrusion in data in transit and data at rest, defined as the data sent by the user to the service provider and stored by the latter. As long as the terms and conditions are not against the law that applies in the circumstance, the harvesting of users' data after the acceptance of the aforementioned terms and conditions is legitimate. There is a trade-off between having a service and having the exclusive control over personal data. For this reason, and since most of the network-based services are provided by American companies, privacy in the U.S. is usually informally defined as a commercial right.

The United States regulations concerning privacy are not immune from the already described vagueness problem. In particular, the National Security exception has been taken to an extreme level to justify mass surveillance not only of private US citizens, but also of citizens and entities based in foreign countries, as described by the leaked information in the last few years [11].

2.1.2.4 Privacy in China

The final legal system we cite for the sake of comparison is the Chinese one. It is of interest to us because of the inner distinction between privacy with respect to companies, and privacy with respect to the government. According to Pernot-Leplay et al. [33], the Chinese approach to privacy is an example of middle ground between the EU and US approaches, at least for what concerns data privacy. The authors clearly state that the data privacy framework applies to users as consumers, but not as citizens. As such, the relevant law protects Chinese users from abuses by private companies, while the government is free to pursue its notorious and invasive state surveillance program. Additionally, the Chinese law in general tends to be vague, and the privacy law is no different. Therefore, the most relevant fragments of the Chinese approach to data privacy are part of non binding dispositions that private companies are supposed to follow as "best practices.". This is in starch contrast with what users (as citizens) can expect in terms of privacy. Not only individuals are restricted from visiting certain websites, feature achieved thanks to the so called *Great Firewall of China*, but users are often forced to use specific software applications (the others are forbidden) in order to access network-based services. These applications are usually heavily monitored by the central government (or by affiliated entities). The claim is that this is a necessary trade-off between the freedom of people to get services and the control of potential rebellions/criminal activities.

Essentially, this is a complete and explicit subordination of every right of individuals to national stability and alleged security. The result is the partial or complete suppression of rights which are given for granted in the legal systems we previously examined.

2.1.3 Privacy in Computer Science

In this Section we discuss how privacy is framed within Computer Science and its applications such as digital devices. As we briefly mentioned in Chapter 1, in that context we cannot unlink the privacy of a user from the security of its data. In other words, a notable fraction of the privacy of a user coincides with the security of its data. In order to avoid a long periphrasis every time the subject is touched, we defined two new labels: *privacy of data*, and its abbreviated form *data privacy*. When either of them appears, their meaning is *privacy of a user that depends on the security of their data.*

From now on, in this thesis, unless differently specified, the term *privacy* will be synonymous with and refer to *data privacy*. In that regard, threats to privacy can be mostly partitioned into *insufficient security* and *misuse and abuse* of legitimately obtained data. We discuss both in Section 2.1.3.1 and Section 2.1.3.2. Likewise, the expression *private data* for our purposes indicates a set of data whose privacy is valuable by its owner (such as a user).

2.1.3.1 Privacy and Security

Whenever data are managed by digital devices, one of the biggest threats to their privacy is insufficient security. For a device storing or interacting with any kind of private data, security is a necessary condition for the privacy of those data. While perfect security is not achievable in the general case, the correct expression to use (and what to aim for in practice) is *secure enough*, denoting an acceptable level of security, given a certain threat model. The reason why security is paramount for privacy descends from the fact that security vulnerabilities are likely to open sinks for data to flow towards an attacker's possession. It is not necessary for the attacker to compromise the entire device for that to happen, even though, in that scenario, every single piece of data on the device (unless encrypted at rest) is compromised. The situation would be so severe that, if the threat model includes an attacker having full and unrestricted access to a device (e.g., root privileges on an Operating System), little can be done in any case. It is also worth noting that it is in principle possible to exploit a vulnerability without leaving any trace. Therefore, it is possible that a user never realises that their device and their private data have been compromised. That of course does not remove the fact that a privacy breach has happened. Furthermore, if a device exchanges data with the external world - think network communications on the Internet - the privacy of those data is dependent on the security of the communication channel. The main issue here is that no communication channel is secure by default. Therefore, a secure enough channel has to be actively established before a secure data transfer can begin, together with some form of identity verification of the other party. Unfortunately, the establishment of a secure enough communication channel is error prone, not trivial at all, and full of traps. It does not help that attacks only get better with time, so what is secure enough today almost certainly will become insecure some time in the future. Ultimately, we can reinforce the concept that the lack of appropriate security within devices and their components (such as Operating Systems), and the lack of secure enough communication channels arguably pose the biggest threat to the privacy of data.

2.1.3.2 Privacy and data treatment

The other face of data privacy is how the devices, the people, and the institution that are entrusted with the custody of private data actually manage them. A given software could be secure enough from external threats, but malicious in its behaviours. If leaking private data is part of the intended functionalities of an application, rather than the result of a vulnerability, that falls out of the threat model of an external attacker. It is still deadly for privacy, though. Even discounting the behaviour of a user-controlled piece of software, such as a classic computer program or a mobile application, there are legitimate scenarios in which remote devices (think servers) legitimately come into possession of user data, such as the user sending them to the end-point to fulfil the prerequisites for accessing a service. It is hopefully crystal clear that, from a technical point of view, whenever a piece of data is sent to any of those remote sinks, the user has lost forever the exclusive control over it. At least some of the possible actions that can be performed on remotely stored data (whether or not accessible by the original user) are in general influenced by forces that do not depend at all on the will of the original user. Examples include:

- The remote party could have security problems of its own, which expose vulnerabilities that are exploitable by attackers. Those vulnerabilities can result in data leaks, such as database compromise. This point mainly refers to threats to privacy that do not depend on the active will of the remote party.

- The remote party could be outright malicious or go rogue at any point. Sometimes, in the case of organisations, one single rogue employee can significantly harm the privacy of the collected data. This point mainly refers to threats to privacy that stem from illegal activities.

- The remote party could copy the data to an indefinite number of locations and/or infer more data via data mining techniques. It could also sell the data to third parties such as advertising companies. This point mainly refers to threats to privacy that stem from legal activities. It also allows us to point out that not all the actions that compromise privacy are necessarily illegal.

The main issue here is that not only it is difficult to determine in general whether the first point applies, but also that a perfect cover up is technically always possible in the other two for an indefinitely long time. Sometimes, no cover up is needed, as the remote party is known to perform privacy damaging actions (such as selling private data to advertising companies, which is the norm for businesses whose revenue mainly depends on showing ads to users). For these reasons, we can anticipate that we are not interested in threats to privacy originating from the three scenarios, unless they are preventable by some user actions (e.g., refusing to send data to a server when insufficient security parameters from the server are detected). More on that in Section 2.1.4. We are still interested in the issue of malicious software described at the beginning of the Section, though.

2.1.4 Working definition of Privacy

After describing the different perspectives on privacy in Section 2.1.1, Section 2.1.2, and Section 2.1.3, we are now in the position of stating and discussing the working definition of privacy that we will assume for the remainder of the thesis, together with the classes of threats we will consider. The starting point is how privacy is framed in Computer Science, i.e, *data privacy* as defined in Section 2.1.3 (which includes the relation between privacy of a user and security of its data). Anything else is out of scope. We live in a world where digital devices are pervasive. We can take for granted that data privacy constitutes at least a plurality (if not the majority) in any meaningful partition of the concept of privacy. Threat wise, we limit our scope to those threats to privacy that are relatable to software that the user uses and has the control on. Network communications are included because the user side has the power to close the connection at any time. Furthermore, among all the classes of software, we limit the scope even more to Android applications. After applying all these constraints, we can conclude that, for the remainder of this thesis, our definition of privacy consists of the privacy of data that are managed by Android applications. The threats we consider are those that stem from insufficient security and malicious applications. Essentially, what is in scope is:

- Malicious behaviours.
- Application vulnerabilities while storing data.
- Application vulnerabilities while exchanging data.

To clarify, the latter includes not reacting appropriately to apparent problems with the security of remote endpoints of network communications. On the other hand, the following is out of scope:

- Anything regarding hardware vulnerabilities.
- Anything regarding non-Android Operating Systems.
- Any vulnerability that is intrinsic to Android as an Operating System, rather than Android applications.
- Anything regarding remote endpoints that is not reactable to and influenceable by users.

While we mainly disregard philosophy and the law in our working definition of privacy, we need to interface with them for the purpose of this thesis. We have already mentioned in Section 2.1.1 that lawmakers are inspired to some extent by philosophical definitions of privacy when they legislate on it. As one of our aims is to bridge the worlds of technology and law with regard to privacy (see Section 1.2), we focus exclusively on rules and regulations on data protection (See Section 3.1). We assume that our working definition of privacy is compatible with the intended interpretation of the concept of privacy referenced in or implied by the relevant law on data protection. That is the case because, by choosing to focus on the subset of privacy that is affected by security, we define a rule or regulation *relevant* if and only if it relates to certain properties that are characteristics of security. Once again, we explain the process in greater detail in Section 3.1.

Our working definition of privacy is an important prerequisite to our threat model to privacy. We will describe and discuss it in detail in Section 3.3.2.2.

2.2 Security Background for Android

Android is by far the most popular mobile Operating System (OS), and it has been for years. It is built on the top of a modified Linux Kernel. From the user experience point of view, the user is presented with an app model: rather than allowing the direct execution of software like most desktop OSs, programs are encapsulated within applications that are run on the underlying OS. The structure of Android is shown in Figure 2.1.

Being an application oriented OS is a deliberate security choice of the system which relies on a particular threat model [35], and has two sides. The first is about the platform level security and relates to the OS itself. The second

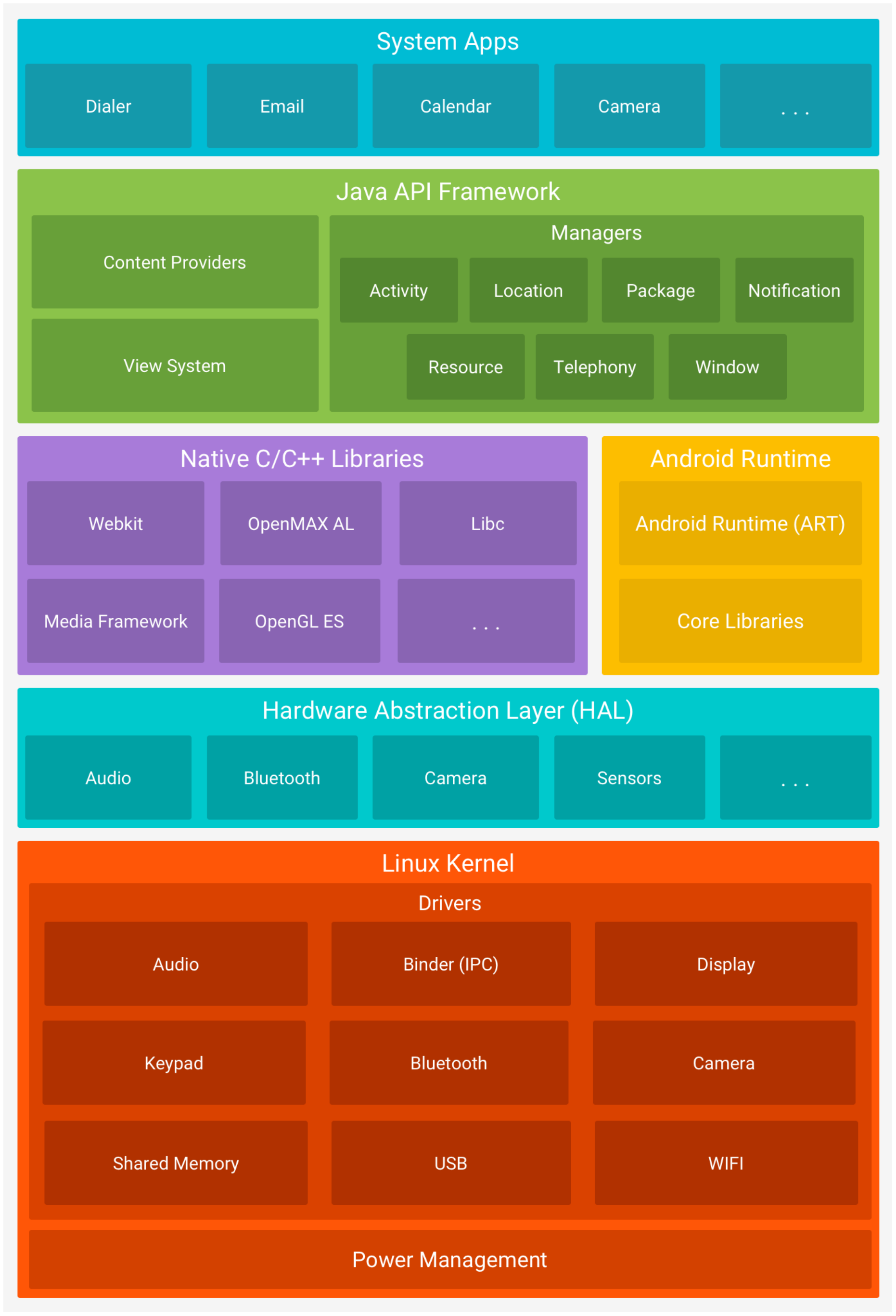

Figure 2.1: Android OS Structure: The layered structure of the Android Operating system. [34]

is about application level security and relates to the security of the single applications. Next, we discuss platform level security in Section 2.2.1, and application level security in Section 2.2.2.

2.2.1 Platform level security

At the OS level, for the most part, Android does not expose an interface to directly interact with low-level OS files, unlike many desktop OSs. While some interaction is possible, it is heavily regulated by the permission system described in Section 2.2.2. Additionally, every application runs in a dedicated sandbox, so to minimise the attack surface for an attacker that has compromised a single application. Inter-app communication is possible, but still heavily regulated under the aforementioned permission system. A detailed description of the Android OS security model can be found on the official website [36].

2.2.2 Application level security

At the application level, the main features of the security model focus on exposing a secure enough API to interact with the OS and other applications. The underlying idea is that every privilege that is not explicitly needed and requested is denied by default. That is in essence the rationale behind the permission model. The API for performing certain actions (e.g., interacting with the OS, performing network activity, communicating with other applications, etc.) is always available to developers. However, at runtime, if the appropriate permission (or set of permissions) has not been requested by the app and granted by the user, an error is triggered. Before Android 6.0, an application needed to ask for all the permissions it might have needed until the next update at install time. That was not the best practice, and led to many applications asking for too many permissions, some in good faith, some not. Since Android 6.0, the permission framework has become more dynamic. In particular, while most permissions can still be asked at install time, the permissions labelled as *DANGEROUS* are requested at runtime when the developer thinks the application needs them. Upon receiving a permission request, the user can either deny a permission, or grant it just for that particular time, or grant it just when the application is in use (which potentially impacts applications which perform some tasks in background), or grant it until it is explicitly revoked. While the latter is less secure than having to ask for a permission every time it is needed, it is a trade-off between security and usability. Android permissions are categorised as belonging to 4 different classes: *PROTECTION_NORMAL*, *PROTECTION_SIGNATURE*, *DANGEROUS* (as anticipated earlier in the Section), and *SPECIAL*. Some permissions belong to multiple classes. For a detailed description of every single permission, we encourage to read the official specification [37]. The permissions that a certain application holds at runtime feature in our ASO (see Section 3.3.2.2). A notable aspect of application security within Android is that, other than requiring a specific permission, there are little constraints by default on network communications. We explain why that is a problem in Section 2.2.3 and Section 2.2.3.5. The other notable face of Android application security is how they manage data at rest. We discuss that in Section 2.2.4 and Section 2.2.4.1.

2.2.3 Security of network communications

We define network communication as the set of all the interactions that an Android application performs on the top of the TCP/IP protocol stack. They include interactions with other devices within a LAN, and interactions with anything of the Internet. Network communications involving an Android application are not secure by default. In fact, they inherit all the security issues of network communications in general. The main issues are that no communication channel is secure by default (i.e., a Man In The Middle - MITM attack can always be performed), and communication endpoints can, in principle, lie about their identity. MITM refers to the ability of an attacker to sit in the middle of a communication and record data in transit (passive MITM), and/or alter data in transit (active MITM). In both scenarios, the attacker impersonates each endpoint to the other. The first issue (insecure channel) raises the problems of *confidentiality* and *integrity* of the data in transit, while both issues (insecure channel, and lying about the identity) raise the problem of *authenticity* of the data in transit, or, in other words, the problem of endpoint *authentication*. For these properties, it is important that either they are guaranteed (confidentiality), or violations are immediately discovered (integrity and authentication), so that counter measures can be adopted - usually interrupting the communication, or giving the user an informed choice. Confidentiality refers to the fact that data in transit should not be available to anyone other than the sender and the intended recipients. Integrity refers to the fact that data is transmitted unaltered (i.e., no additions or removal) from the sender to the recipients. Note that ensuring integrity of a given set of data is impossible due to network errors and the ability of active MITM attackers to drop every packet they see over a channel they control. Rather, the focus is on detection of integrity violation. Authentication refers to how to prove the identity of an endpoint to the other(s) while each one can in principle lie, and a MITM is always possible. As the three properties are not independent (e.g., failing to provide authentication undermines the other two), solutions, all of which widely employ modern cryptography, tend to address all of them as an indivisible single issue. Authentication, discussed in Section 2.2.3.1 is usually the most problematic of the three, because no amount of cryptography (or anything else, for what matters) can reasonably bind at scale the declared identity of two endpoints that had no prior contact to their real identities. Insecure protocols and insecure protocol parameters are discussed in Section 2.2.3.3 and Section 2.2.3.4 respectively.

2.2.3.1 The problem of authentication

In the field, the authentication problem is known as *how to safely transmit Alice's public key to Bob*, after the usual little story involving two parties (Alice and Bob) that want to communicate over an insecure channel where Eve (or Mallory, depending on the version) is always ready to perform a Man in The Middle attack. The public key, relating to public key cryptography, is what exposes the identity of a party to everyone else. There is no definitive solution to that. As such three competing approaches attempt to mitigate the problem at scale (meeting in person and exchanging keys is not a scalable solution for obvious reasons):

- Public Key Infrastructure (PKI)

- Web of Trust (WoT)
- Trust On First Use (TOFU)

The next paragraphs discuss all of them.

PKI The first approach involves a party that wishes to engage in future network communications to request a certification from a third party (called Certification Authority - CA). The CA, after verifying the identity of the party with some out-of-bound methods (which is usually context-dependent) releases a certificate that includes the party identity, the party public key, some additional data and constraints (such as what the certificate can be used for), the party signature, and the CA signature. The idea is that the party can prove its identity to anyone trusting the CA by simply sending the certificate. The main issue with the approach is that all it does is shifting the core point from trusting a party to trusting some CA that is out there. The issue is mitigated by creating a pyramidal structure of CAs whose purpose is to provide certificates for other CAs that conceptually are on a lower level in the pyramid. Several of this pyramidal logical structures exist, and each of them has a so-called Root CA at the top. A root CA has a certificate that is self signed. The major root CAs have their certificate pre-installed in all the major clients (which include Android devices), so that it is possible for a party to prove their identity by sending their certificate together with all the certificate of the intermediate CAs. The other party can now validate the chain with the help of the appropriate Root CA certificate. The whole pyramidal structure of CAs and certificates is referred to as Public Key Infrastructure (PKI). By design, the PKI has an obvious single point of failure which is the Root CA. If that is compromised, then nothing downstream can be trusted anymore. Additionally, even non-Root CAs can become points of failure if they are compromised or go rogue. For all those reasons, CAs are expected to meet excellent standards both in terms of security and transparency. As web communications are the main scenario where a PKI is used to mitigate the problem of authentication, the rules all CAs and web browsers are obliged to abide by are publicly discussed and voted on in the so-called *CA/Browser forum* [38]. However, the main unsolved issue with the PKI is trust revocation. As this is a major problem for privacy, we explain and discuss it in Section 2.2.3.2. For now, we can say that the PKI trust model is intrinsic to the Transport Layer Security (TLS) protocol which aims at creating a secure enough and (partially) authenticated communication channel between two endpoints on a network.

WoT The second approach relies of the principle that *I trust someone that someone else that I already trust trusts.* Far from being a tongue twister, the idea is, for a party, to explicitly mark as trusted all the public keys of the parties that it trusts (for whatever reason). Then all the public keys are published to special servers called keyservers, where anyone can get them together with the “trust marks”. Whenever a party encounters a new public key, it trusts it if and only if, after a search on a keyserver, it is able to find a “trust mark” of another party that is already trusted by the original party. The main issue with WoT for an endpoint is how to behave when presented with a public key that has not been signed with a “trust mark” by anyone that the endpoint already trusts. Trusting everyone of those public keys is equivalent to not having any trust system at all, while trusting none generates an unacceptable amount of false positives, given that there are far more good endpoints than attackers, and the probability of being subject to an active attack when presented with a not-yet-trusted key is rather low. For this reason, WoT has niche use cases, and was never considered as a scalable solution to the authentication problem in general. In particular, WoT has not been systematically integrated into the Android platform at any level.

TOFU The third approach relies on the assumption that it is unlikely that an active MITM attack is being performed when communicating for the first time with a remote endpoint. As such, an endpoint that wishes to verify the identity of another endpoint for the first time simply assumes that the received public key is genuine and binding. Therefore, as long as the remote endpoint continues to present the same public key in successive network communications together with a valid signature (the details are protocol-specific), the first endpoint believes that no MITM is being performed. The obvious point of failure of TOFU is that, if a MITM was happening during the first communication, then the endpoint is mistakenly believing that an attacker controlled keypair represents the identity of a certain endpoint, and will not question that as long as the attacker is able to perform a MITM. Another notable issue is that the user is the ultimate decider on what to do when and if the endpoint public key changes in the future. Unless a MITM or a breach of the remote endpoint are suspected, it is all but secure that the user will trust any new public key. On the other hand, the obvious advantage with respect to a PKI approach is the non-involvement of any third party.

SSH and Signal are examples of protocols that incorporate a TOFU approach to validate remote endpoints.

2.2.3.2 The problem of revocation

Establishing trust is only half of the problem of authentication. The other face of the medal is trust revocation. If a party is compromised by an attacker, they can now mount an impersonation attack on all the endpoints that engage in any kind of network activity with the party (think a compromised web server). When that happens, the legitimate owner of the device(s) that allow the original party to communicate must be able to instruct all the potential communication endpoints not to trust the (compromised) party anymore. It is noteworthy that regaining full and exclusive control of the compromised infrastructure is not enough to prevent impersonation attacks. In all likelihood the attacker has been able to copy the private key that can be used to sign whatever can be verified with the corresponding public key. Therefore, the attacker can still hand over a public key (or a certificate in the case of a PKI approach), and use the private key to "prove" that they are someone or something they are not. For this reason, any incident that results in a non-null probability of private key theft triggers the issues of revoking the previously established trust, and re-establishing a new trust.

WoT For WoT the process involves explicitly revoking the public key, and reuploading it on the keyserver(s). Obviously that only works if the "status" of a received public key is always checked and compared with data from a keyserver. Re-establishing trust is more difficult and time consuming, because all the "trust marks" on the now revoked key do not get passed to the new key.

TOFU For TOFU trust revocation and trust re-establishment happen by accepting a new public key from the previously breached endpoint, under the assumption that the legitimate owner has regained full and exclusive control of the infrastructure. That implicitly discards the old public key. The main issue is that there is no general way to distinguish a good-faith key change from an active MITM attack.

PKI Trust revocation in a PKI environment starts with the compromised party informing the relevant CA of a potential security breach. According to the CA/Browser forum rules, the CA is obliged to mark any certificate whose public key relates to a potentially compromised private key as revoked. The issue is that certificates are read-only. They cannot be updated, as that would cause a signature mismatch. And even if they could be updated, the real problem is how to convey the message that the original certificate must not be trusted anymore to all the possible network endpoints. Two main strategies are usually employed: Certificate Revocation Lists (CRL) and Online Certificate Status Protocol (OCSP). For various reasons, none of them work in practice [39]. In particular, CRLs do not scale [40], and OCSP is completely useless in case of an active MITM attack, and severely compromises availability and usability [41].
For the reasons explained in the previous paragraphs, revocation is broken in general. However, in very specific cases, some of the techniques described above can be used to effectively mitigate MITM attacks. As such, they are

part of our ASO, described in Section 3.3.2.2.

2.2.3.3 Insecure protocols

The mitigations to the problem of authentication need to be framed into the broader issue of creating a secure enough communication channel. That implies careful consideration on the properties of data confidentiality and data integrity. The field that studies how to build secure protocols for data in transit and how to analyse existing protocols from a security point of view is called Transport Security. Despite its name, the secure channel is always built at the application layer because it would be extremely difficult to update every single network device in existence that only implements TCP and UDP as transport protocols. Even Quic, a proposed standard for a transport protocol that encapsulates and ameliorates TCP and TLS runs over UDP [42]. If no third party is involved in the establishment of a secure channel, the expression end-to-end encryption is often used to describe it. The most notable protocol in that regard is Signal. Among non end-to-end protocols, the most notable and relevant in our context is Transport Layer Security. Regardless of the end-to-end nature of a protocol, it is a fact that security was an after-thought (if a thought at all) when the Internet infrastructure was built and its protocol stack was standardised in the '80s and '90s of the 20th century. Consequently, network communications today are littered with a plethora of grossly insecure protocols, and protocols that were once deemed secure enough, but have since aged poorly.

Cleartext protocols Any protocol that does not encrypt data in transit is trivially vulnerable to MITM. Any attacker can read and modify (even drop) the data in transit without any of the endpoints being or becoming aware of it. Additionally, an attacker can flawlessly impersonate each endpoint to each other. These kind of protocols are called cleartext protocols. Every protocol below the application layer in both the ISO/OSI and TCP/IP protocol stacks is a cleartext protocol. This reality makes rather difficult to preserve the confidentiality of at least some communication metadata (e.g., the IP addresses and the TCP/UDP ports often reveal which device is communicating with which device). Originally, every protocol in the application layer (in a network sense, rather than in an Android sense) was cleartext. However, in the mid-90s, when MITM was recognised as theoretically possible, non-cleartext protocols where standardised. Even discounting the level of security that they actually provide, the fact that all the protocols where cleartext at the beginning means that cleartext is still the default way to go when implementing one side of a network communication. For instance the developer of an Android application can easily open a cleartext connection with any other network-capable endpoint (unless the endpoint explicitly refuses). In some cases, cleartext data transfer is mandatory (think captive portals when connecting to certain WiFi networks). As cleartext protocols are trivially insecure, we include them in our ASO (see Section 3.3.2.2).

Non-cleartext protocols When the need for a secure communication channel became apparent, various entities began standardising protocols featuring cryptography as a mean to protect data confidentiality, detect violations of data integrity, and allow for endpoint(s) authentication. Unfortunately taking off-the-shelf cryptography primitives and combining them in a way that seems to work is a terrible way to secure a communication channel, because a secure protocol requires more than secure components. Additionally, if one or more of those primitives become insecure, the whole construction fails spectacularly. For these reasons, almost every protocol aimed at establishing a secure communication channel is considered obsolete when not completely insecure. That includes most past versions of TLS. As the Internet does not update atomically [43], a large number of software implementations of insecure protocols are still being used today. Their use is often sufficient to expose significant vulnerabilities to external attackers. For this reason, we include them in our ASO (see Section 3.3.2.2).

2.2.3.4 Insecure parameters

Even though the designers of modern transport security protocols, such as TLS 1.3, have undertaken a monumental effort to minimise cryptographic agility (i.e., remove all the unnecessary features that may turn insecure, and rather focus on a small number of strong primitives), there are still secure enough protocols that can be used in an insecure way. That is the case because it is often up to developers to choose some of the parameters of the protocol (e.g., the advertised ciphersuites in TLS). The lack of secure opinionated defaults, the incremental addition of new features, and the progression of attacks have made rather easy to choose a weak or insecure configuration by mistake or by ignorance. Choosing insecure parameters is often enough to transform a secure enough protocol into an insecure protocol. For this reason, insecure parameters are a part of our ASO (see Section 3.3.2.2).

2.2.3.5 Consequences of insufficient network security

After discussing how and why network communications can be insecure, in this Section we describe the consequences that they bear in terms of data privacy with respect to the properties of data confidentiality, data integrity, and authentication. The breach of data confidentiality is trivially a sufficient condition for a data leak, as the attacker is able to, among other things, read the transmitted data. The undetected breach of authentication implies that data intended to be transmitted to a certain endpoint is actually transmitted to a third party. That qualifies as a breach of data confidentiality. The undetected breach of data integrity is often a sufficient condition to mount an attack against data confidentiality and/or authentication. For these reasons, we can say that an insecure communication channel is a direct threat to the privacy of *data in transit*.

2.2.4 Security of data at rest

The dual scenario of *data in transit* is *data at rest*, or, in other words, data that is stored somewhere for future use. In the context of Android applications, data at rest is whatever is stored within an Android device. Concrete examples include the list of contacts, images, videos, etc. The access to any of those is regulated by the permission framework (see Section 2.2.2). As the ultimate decider on whether to grant or not a specific permission to an application is the user, it is possible for malicious applications to employ social engineering techniques in order to trick the user into granting certain permissions. Another scenario is that legitimate functionalities of a legitimate application are locked behind permissions that are too broad, and can therefore be abused. Note that we are not interested in the security of data stored by servers that Android applications may connect to, as that kind of security cannot be influenced by an application itself. With respect to remote servers, we are only interested in whether an application connects to known malicious servers, and how it reacts to live security decisions by servers that affect the security of either data in transit, or data stored on the Android device.

2.2.4.1 Consequences of insufficient storage security

It is reasonable to assume that Android devices store a significant amount of data that either need to remain private, or are to be shared with a selected and limited number of third parties. As Android application can access those data under certain conditions, it is possible for vulnerable and/or malicious applications to act as the source for a data leak. For this reason, we include the most important application vulnerabilities that can result in data at rest being leaked in our ASO (see Section 3.3.2.2). For the same reason, we include the status of an application (benign, malicious, suspicious, etc.) as part of our ASO.

2.2.5 Security and data leaks

In Section 2.2.3.5 and Section 2.2.4.1 we discussed how insufficient security correlates with data leaks. In this Section we discuss how the security community mitigates the causes of data leaks. First of all, we need to make clear that this Section does not focus on prevention (which we already implicitly discussed in the previous Sections). Once we accept that at the moment there are paths that lead to data leaks, part of the focus shifts to defense-in-depth and detection. We explain what they are in Section 2.2.6. In parallel, part of the focus is dedicated to detecting vulnerabilities and malicious applications, so that the first can be patched, and the second neutralised. We discuss security-based application analysis in Section 2.3.

2.2.6 Defense-in-depth techniques

In the context of security, a defense-in-depth technique is a second line of defense that is supposed to be triggered when unpatched vulnerabilities can lead to exploits. A notable example for web-based interactions (which are not alien to Android, thanks to WebViews and web browser applications) is Content-Security-Policy (CSP) [44]. Cross Site Scripting (XSS) vulnerabilities have plagued the web ecosystem since its beginning, due to the fact that the web platform is not secure by default, and easily allows for the unsafe mix of code and data. CSP is a defense-in-depth technique that, when used properly, instructs browsers to block the execution of scripts which do not fulfil at least one of the following conditions:

- Present a response-specific “nonce” attribute whose value matches a specific value delivered out-of-bounds via the Content-Security-Policy HTTP header (or perhaps via Origin-Policy in the future).
- Present a “hash” attribute whose value matches the hash of the script and a specific value delivered out-of-bounds via the Content-Security-

Policy HTTP header (or perhaps via Origin-Policy in the future).

- Have been dynamically added to the DOM in a safe way (i.e., non parser-inserted), provided that the keyword `strict-dynamic` is included in the Content-Security-Policy HTTP header.

Another notable example of defense-in-depth is the mechanism known as *TrustedTypes* [45]. Its purpose, when enforced, is to inform browsers not to load certain DOM elements that have not been wrapped with a specific Trusted Type. As such, a better distinction between code and data is enforced, which makes many underlying XSS vulnerabilities not exploitable.

2.3 Security analysis for privacy

In Section 2.2 we discussed the Security of Android and Android applications in the context of privacy. In this Section, we discuss how application analysis can be employed to detect vulnerable and malicious applications, together with the families of application analysis techniques.
When analysing a piece of software that can be directly executed or interpreted in some context, there are essentially two approaches (plus a hybrid strategy): *static analysis* and *dynamic analysis*. They complement each other both in terms of strengths and weaknesses. Before discussing them, we have to precise that they are *families* of more fine-grained approaches that depend on the context. Throughout the next paragraphs the term *application* will refer to any kind of application software, with an eye on Android applications.

Static Analysis When an application is statically analysed, the focus is on its code, whether source code or compiled binaries. In particular, the application is not run. The main advantages of a static approach are:

- Not having to build an environment in which the application is able to run.
- The ability to explore code paths that may not be triggered at runtime.
- The ability to bypass emulation detection techniques aimed at hiding certain behaviours when the application detects it is run in an emulated (and controlled) environment.

On the other end, the main disadvantages are:

- Having to deal with obfuscated code, if obfuscation techniques were used.
- Not being able to analyse code that is dynamically loaded from remote locations.
- Not being able to analyse code that is self-encrypted.

Dynamic Analysis When an application is dynamically analysed, the focus is on the behaviours that it exhibits while it is run (often in an emulated and controlled environment). The main advantages of a dynamic approach are:

- The ability to bypass obfuscated code (as it exhibits the same behaviours as the corresponding un-obfuscated counterpart).

- The ability to bypass encrypted code (as it has to get decrypted at some point, in order to be executed).
- The ability to ignore dead code (i.e., code that can never be executed).

On the other hand, the main disadvantages are:

- Having to build an environment in which the application is able to run.
- Missing on all the behaviours deriving from those code paths that are not triggered during the particular execution.
- Having to contend with emulation detection techniques which result in the application potentially changing its behaviours after detecting that it is being run in an emulated (and controlled) environment.

Hybrid Analysis A hybrid analysis of an application aims at combining the strengths of static and dynamic analysis, while minimising the weaknesses. Often static analysis is employed as the first step (e.g., reading the Android Manifest file in the case of an Android application), and its results are then used to better guide the dynamic analysis that follows. While the advantage is obvious (see just above), the downsides are the level of complication and the absence of a general guarantee regarding how many of the weaknesses of static and dynamic analysis are actually mitigated by a specific hybrid approach.

2.4 Limitations of the State-of-the-Art

In Section 2.2 we discussed the security implications of Android, and the efforts to provide a secure enough environment to its users, together with the consequences of potential failures. In Section 2.3 we discussed how application analysis attempts to mitigate some of the security flaws that put private data in jeopardy. We can regard what we discussed in those Sections as the state-of-the-art. In this Section we discuss the limits of the state-of-the-art with respect to privacy. In particular, in Section 2.4.1 we discuss the limits of application analysis, and in Section 2.4.2 we discuss the limits of the law, represented in our case by the limits of the 2018 UK DPA. Finally, in Section 2.4.3 we discuss the barriers that prevent application analysis and the law from efficiently complement each other with respect to the aim of protecting privacy in the context of Android application.

2.4.1 Limits of security analysis

The individual fine-grained approaches that fall under the umbrella of the families of strategies discussed in Section 2.3 are usually implemented into analysis tools. While they are often good at finding what they were built for under the specified preconditions, they all suffer from two limiting factors: the mostly standalone nature, and the arbitrary format (syntax and semantics of their reports). We discuss the former in Section 2.4.1.1 and the latter in Section 2.4.1.2. Furthermore, the analysis tools usually come with performance metrics that have been measured by running the tools on certain datasets of Android applications. In Section 2.4.1.3 we discuss the importance of the dataset in correctly assessing the performances of tools, while in Section 2.4.1.4 we discuss the issue of believability of the provided performance indicators.

2.4.1.1 Standalone nature of the tools

Most tools are not built by design to be part of a toolchain that takes the output of a tool to guide the analysis of another tool downstream. While there are exceptions (e.g., FlowDroid and VirusTotal), most tool only focus to find something very specific within Android applications, and, whether they find it or not, they output a report and call it a day. The inevitable result is little chance of integration with other tools, in order to deliver a broader picture of the security vulnerabilities and implications (including being malicious) of a certain application. In turn, this poses an additional obstacle to the integration of security analysis with other subjects, such as the law.

2.4.1.2 Arbitrary format of the reports

Each tool, more often than not, outputs its findings in a format that is specific to the tool itself. In other words, there is no underlying ontology that supports a better organisation of the results via standardised syntax and semantics. The inevitable result is, once again, little chance of integration with other tools, in order to deliver a broader picture of the security vulnerabilities and implications (including being malicious) of a certain application. Yet again, this poses an additional obstacle to the integration of security analysis with other subjects, such as the law.

2.4.1.3 Dataset and experimental evaluation of the tools

Each tool purports to be able to achieve certain scores with respect to certain metrics (such as accuracy, precision, recall, etc.). Such scores are normally inferred with a process of experimental evaluation on the tool itself. For such evaluation to be possible, a dataset of Android applications is needed. The crucial point is that the dataset can influence the results, if it exhibits biases. For instance, in an extreme situation, if a tool aims at detecting network traffic over insecure protocols, and the dataset fully consists of applications that never connect to the Internet by construction, then the experiment is not really evaluating whether the tool can detect insecure network communications. In other words, for the performance scores to be truthful (plus or minus an unavoidable degree of uncertainty) the dataset must be representative of the real world ecosystem, and free from biases as much as possible.

2.4.1.4 Believability of the tools

Each tool purports to be able to achieve certain scores with respect to certain metrics (such as accuracy, precision, recall, etc.). In Section 2.4.1.3 we discussed the importance of the dataset in that regard. In this Section we discuss how the believability of the results of a tool constitutes a limitation of the state-of-the-art when such results are used as input for further analysis. The subject has been researched in the past, e.g., by Pauck et al. for Android taint analysis tools [46]. The main limitation that stems from unreliable tools (i.e., tools whose performance scores are not believable) has to do with how biases, errors, and misclassifications propagate through toolchains. Essentially, when the output of a tool is used as input for another tool (which is usually not straightforward - see Section 2.4.1.1), the output of the second tool is influenced by the quality of the data in input, as well as the uncertainty introduced by the tool itself. As such, *un-believable* results (in a broad sense) produced by tools in a toolchain tend to multiply downstream. The issue of containing such propagation is still open.

2.4.2 Limits of the law

The law, represented in this thesis by the 2018 UK DPA, uses technical terms that it fails to further define or reference. For example, the terms *security* and *confidentiality* (just to cite some) that can be found within the DPA have a precise meaning in the context of security, but no effort is made within the DPA to link them with the world of security. This omission has very real consequences due to the inevitable interpretation of the law that is intrinsic to court cases. In particular, not grounding some of the most important terms, does little to limit the degree of interpretation, which in turn may lead to misguided rulings. Also, on the legal side, there has been no serious attempt to systematically link the relevant articles of the DPA on data protection to the findings of security analysis. Therefore, at the moment a strong and systematic integration between security analysis and law on data protection is virtually non-existent.

2.4.3 Obstacles to integration

We have established that security analysis and the law at the moment hardly cooperate to protect the privacy of Android applications users. The causes of this lack of cooperation can be traced to pitfalls that are intrinsic to both sides. We have discussed them in Section 2.4.1 and Section 2.4.2. Essentially, they can be resumed into three main issues:

- The standalone nature of most analysis tool.
- The lack of an underlying ontology specifying syntax and semantics for the reports of the analysis tools.
- The use of undefined and unreferenced terms from the domain of security by the law.

2.4.3.1 Requirements for integration

In this thesis, we aim at bridging security analysis and the law. As such, we necessarily have to deal with the above issues. We designed and implemented MagnetDroid in part to mitigate those issues. Chapter 3, Chapter 4, and Chapter 5 detail how we tackle them.

2.5 Summary

In this background Chapter we have laid the foundations for the reader to contextualise and understand our work on MagnetDroid. We began by discussing the concept of privacy from different points of view: philosophical/historical, legal, and technological. Then, we established privacy of data as our area of interest, and insufficient security as its biggest threat. In that regard, we discussed the security features intrinsic in Android at the platform level, and at the application level. In particular, we focused on network security, and security of data at rest as the most relevant components of the security of Android applications. After discussing the consequences of insufficient security, we spoke about defense-in-depth mitigations, and application analysis as a detection "tool", focusing on the different families of application analysis. Following the discussion on the technological background, we shifted our focus to the legal background. In particular, we selected the 2018 UK DPA as our candidate for the relevant law on data protection. Once we established the state-of-the-art, technological and legal, we discussed its limitations. In particular, we focused on the limits of application analysis, the limits of the law, and the obstacles to a cooperative approach between the two. In Chapter 3 we will discuss the design of MagnetDroid, the framework we built to assess and address the aforementioned limits with respect to our goals and aims (fully described in Section 3.2).

Chapter 3

MagnetDroid: Design and Architecture

In order to mitigate the gaps and the shortcomings in the state-of-the-art (identified in Section 2.4), we propose MagnetDroid, a multipurpose agent-based framework that we envision as foundation for the integration of the Android application analysis and legal domains, as well as a contribution to application analysis itself. In this Chapter, we discuss the approach we followed, and the conceptual design of MagnetDroid. We start by stating our platform assumptions in Section 3.1, and continue with Section 3.2 in which we discuss in detail the goals and aims of MagnetDroid in relation to the expected contributions stated in Section 1.3. Then, in Section 3.3 we present the design of MagnetDroid, with particular focus on its internal partitioning into sub-frameworks (whose design and implementation will be discussed in Chapter 4, and Chapter 5). In the same Section, we also introduce our first main contribution of the thesis: the Android Security Ontology (ASO). We conclude this Chapter with a summary in Section 3.4.

3.1 Platform assumptions

MagnetDroid as a project relies on some platform assumptions, some of which are derived from the state-of-the-art, while others are self-imposed constraints that define what is in scope, and what is not. While certain major points warrant dedicated Sections, e.g., the agent-oriented nature of the framework (Section 4.1), and the relation between privacy and security (Section 2.1.3.1), most of the assumptions are elaborated in the following paragraphs.

The need for a bridge between analysis and law The most essential assumption we make is that an actual bridge between the domains of application analysis and the law is desirable. Put another way, the behaviour of software applications needs to be compliant with the law. Additionally, non-compliance is bound to trigger legal consequences. As many of the compliance aspects focus on privacy (and, therefore, on security), and as application analysis is a prime technique to discover security issues (and, therefore, privacy issues), it makes sense to have something that takes the findings of the latter, and relates them to at least a possible interpretation of the letter of the former. Without any form of technological aid, such connection is usually made by legal professionals (e.g., judges and lawyers during litigations). However, the inevitable proxy that is interpreting the letter of the law, combined with the lack of technical (in the context of software applications) expertise from legal professionals, makes for a less than ideal uncertainty regarding the outcome of privacy-related litigations. Any structured and systematic option to reduce such uncertainty is likely to improve the current situation, where there is a long-standing gap between the technical and legal issues involved.

The focus on Android applications While a bridge between security oriented application analysis and the law would in principle benefit any kind of software application, we focus on Android applications. As mobile phones are both the main instance of Android devices, and containers for some of our most personal data (e.g., our location history), we believe they are a good point where to start. Extending the bridge to non-Android and non-mobile applications, would require a degree of added complexity that we believe would compromise our efforts towards the declared goals and aims. Still, we are open to the possibility of such extension in the future, as part of future lines of research, as we explicitly discuss it in Section 7 at the end of the thesis.

The 2018 UK Data Protection Act As outlined in Chapter 1, and Chapter 2, and elaborated in Chapter 5, MagnetDroid currently supports only a small subset of the 2018 UK Data Protection Act (DPA) for what concerns the relevant law on data protection. The design choice is deliberate, and finds justification in the following:

- Identifying and translating the relevant law on data protection for all possible jurisdictions is a task that transcends our goals and aims (see Section 3.2). Therefore, the legal framework is limited to one jurisdiction only. As this thesis and the underlying MagnetDroid framework are developed and produced within an English institution, it is sensible for us to focus on the broad UK legal system.

- Even within the broad UK legal framework, identifying and translating all the relevant rules and regulations on data protection would distract us from our goals and aim. Since the 2018 UK DPA implements the well known and relevant GDPR, it is a natural candidate for our work.
- The 2018 UK DPA itself contains many uninteresting parts and articles for this thesis. Thus, our focus is on a limited subset of it (Art 32, Art. 66(1), and Art. 66(2) of Part 3), which, in our opinion, contains some of the most relevant prescriptions that apply to our research subject (data privacy and security on Android applications).

Soundness vs. Completeness Recall from Section 1.3, one of our aims is to create a framework that relates the security issues identified by Android application analysis tools to legal violations. Two of the key questions that may arise, in that regard, are:

1. Can our framework identify all the legal violations (with respect to the supported law) that a non-compliant application may trigger?
2. If our framework finds a link between a reported security issue and a legal violation, how believable is it?

The first question relates to the concept of *completeness* of analysis, while the second question relates to the concept of *soundness* of analysis. However, we believe that these questions, no matter how legitimate, are misplaced. The main reason, as discussed in Section 2.4.2 is the issue of interpretation of the law. Depending on the interpretation, the same security issue in the same context can be deemed as probable cause for a legal violation or not, subject to the views of different judges. As such, it is impossible to answer the second question in general, because the concepts of *true positive* and *false positive* are always relative. What MagnetDroid provides instead, whenever a link is found, is a possible interpretation of the consequences of security pitfalls in Android applications under specific conditions. In fact, as we discuss in Section 3.3.2.2, we bifurcate the interpretation for every security issue, by introducing the concept of *mindset* (explained in Section 3.3.3.1), built on the top of the *indexes* features of our *Android Security Ontology* (ASO), described in Section 3.3.2.2. Likewise, the issue of interpretation of the law, makes it impossible to clearly define what a *false negative* is. Therefore, the first question is unanswerable as well, in general. A final consideration on the matter is that, even in the unrealistic hypothesis that only one possible interpretation of the law existed, our performances would be limited by precision and recall of the tools we employ to find security issues, some of which are not known or declared by the tools' developers. We accept that, if a tool produces a false positive (with respect to a security issue) which eludes the *Conflicting* safeguard (essentially a procedure to cross-check the consistency of reports on a specific instance of a specific security issue, fully explained in Section 3.3.2), then it is likely that, no matter the interpretation of the law, MagnetDroid finds a link that is always a false positive (with respect to violating the law). Likewise, we accept that, if all tools fail to report a security issue, it is likely that, no matter the interpretation of the law, MagnetDroid produces a false negative by not establishing a link to a security violation. This does not mean that our approach is not useful, however, but instead that we should be aware of the nuances that we will have to consider when analysing an Android application with our system.

3.2 Goals and aims

Given the open issues with the state-of-the-art discussed in Section 2.4, and the ground assumptions discussed in Section 3.1, we are now able to state and discuss the goals and aims of MagnetDroid. In the following Sections we discuss the logical flow that starts from our main goal, and brings us to new challenges, and novel approaches.

3.2.1 Cyberlegal privacy

Our top goal with MagnetDroid relates to what we call *cyberlegal privacy*. We regard cyberlegal privacy as the union of *data privacy* with legal elements derived from the relevant law. The aim is to provide an automated bridge between those links the security issues found (also with an automated strategy) by Android application analysis tools to legal violations and their consequences. We aim at supporting three main scenarios, as follows. We note that, in the rest of the thesis (due to the structure of MagnetDroid), we will usually split *cyberlegal privacy* into its components, namely *data privacy* and *relevant law on data protection*, and mainly focus on them, rather than on the concept of *cyberlegal privacy* which we just introduced.

The regular user A regular user of an Android application wishes to know whether such application has security issues that pose a threat to the privacy of their data (specifically), and whether the aforementioned application is compliant with the relevant law on data protection (more generally). The assumption on the user part is that an application that either puts private data in jeopardy, or fails to comply with the relevant law, is better avoided.

The vendor and/or publisher The vendor and/or the publisher of an Android application would like to avoid the damage to their reputation, as well as long and costly civil lawsuits that may derive from such application exhibiting security issues that pose a threat/risk to its users, and may also violate the relevant law. The assumption on the vendor/publisher part is that, both a tarnished public image, and a civil lawsuit are detrimental to their finances, and, therefore, any step to prevent the issues that can cause those is welcome.

The legal professional In the last scenario, there is an active lawsuit between a plaintiff who claims monetary reparations for damages caused by privacy leaks allegedly deriving from the insufficient security of an Android application that they have been using. Assuming that the lawsuit is able to survive a motion to dismiss by the defendant (in our case the vendor of the application), the discovery process starts. As every finding, and every related claim of violation of the law is inevitably followed by a process of interpretation of the law, MagnetDroid aims to help by providing a link between security issues found by specialised analysis tools (technological element) and legal violations, together with their consequences (legal element).

Enabling the three aforementioned scenarios, while maintaining an automated approach, opens some new issues whose resolution becomes a set of sub-goals. Section 3.2.2 explains those issues.

3.2.2 Prerequisites for cyberlegal privacy

Building an automated bridge that links security issues to legal violations requires a set of conditions to be true:

- A reliable and automated way to derive security issues from an Android application, and to express them in a standard format. We discuss this prerequisite in Section 3.2.2.1, where we introduce the concept of *integrative analysis*.

- A common language between the domains of Android application analysis and the law. In Section 3.2.2.2 we discuss the issue of bringing two completely different sets (the identified security issues, and the relevant articles of the law) together in terms of syntax and semantics.

- A suitable procedure to link security issues to legal violations, when expressed in a common language. In Section 3.2.2.3 we identify logical reasoning as a suitable procedure to derive legal violations from security issues, once both the security issues, and the relevant law have been translated to logical rules and predicates.

- An easy to understand way to present the results to different kinds of users (given the three different scenarios). In Section 3.2.2.4 we identify a web application as a suitable tool for explaining the results.

3.2.2.1 Integrative analysis

On the Android application analysis side, the tools ecosystem consists of mostly standalone tools that look for a very specific set of security issues, and ignore everything else. As discussed in Section 2.4.1, most of those tools produce reports that are written for humans to understand. As such, the report format (syntax and semantics) is often completely arbitrary, and differs from tool to tool. Not only this is an issue for building a toolchain (an automated a chain of tools where each tool receives as input the previous tool's output), but it also hinders the aggregation of the findings of different tools running on the same application. In our context, aggregating the security issues from different reports produced by different tools is an important task towards the identification of possible causes for legal violations. Therefore, we can formulate the issue as the need for an automated and reliable framework to support a peculiar analysis flow, as follows.

1. Run different tools in parallel on a specific Android application, and collect the raw reports.
2. Translate the reports, so that they exhibit a common format (which we also need to define).
3. Aggregate the translated reports into a final technological report that:
    - encompasses all the security issues that have been found;
    - keeps track of disagreements between reports, solving those that are solvable, and marking as *Conflicting* those that are not.

We call the process described by the above steps *integrative analysis*. With regard to the final technological report, it is also desirable to be able to quickly match each security issue with an explanation in layman terms (see Section 3.2.2.4).

3.2.2.2 Common representation for analysis results and law

Historically, one of the greatest obstacles to any automated bridge between application analysis and the law has been the different language of the two domains (see Section 2.4.3). Therefore, it is no surprise that one of our goals is to establish such common representation language. Additionally, we desire a quick and easy translation from the language of the final technological report (see Section 3.2.2.1) to this common representation. Furthermore, we also desire for the common representation language both to be able to express the core rules from the letter of the law, and to be able to maintain a link to the specific articles of the law for explainability purposes (see Section 3.2.2.4). All considered, what we need is a language that enables the creation of knowledge bases (a technological one from the final report, and a legal one from the letter of the law) that can be then aggregated, in order to extract useful information from such aggregation. In Section 3.2.2.3 we identify normal logic programs[47] as the desired representation language, because not only they can be used to express the content of technological reports, but also allow us to represent legal rules[48]. In addition to that, they have an innate computational counterpart for the implementation of these ideas in Prolog[49].

3.2.2.3 Reasoning as synthesis

Once the security issues and the letter of the law have been translated to knowledge bases that share the same language, a suitable procedure for extracting useful information from the aggregation of the knowledge bases is needed. In our case, we want to be able to derive legal violations and consequences from security issues. Logical reasoning is a suitable procedure. If we frame the legal knowledge base as a set of logical rules of the form *if* A *then* B, we can then use the technological knowledge base to provide A, whose value of truth will determine B. It is easy to see how A can represent a security issue, and B a legal violation. In that regard, Prolog provides an intuitive framework that enables our needs. Additionally, since conditions can contain negative statements, we can make use of *negation as failure* in logic programs, where to prove *not* A we have to prove that all possible ways of demonstrating A fail (and, therefore, we can conclude *not* A)[50]. Furthermore, although normal logic programs do not support explicit negation in conclusions[51], this issue can be addressed at the knowledge representation level by means of choosing suitable domain predicate names (see Section 5.3 for concrete examples). We regard reasoning and its output as the *synthesis* that follows the *integrative analysis* (See Section 3.2.2.1).

3.2.2.4 Results explanation

Logical reasoning is a powerful tool to extract information from knowledge bases via queries. However, the visual format of the query results can fail to be self-explanatory. In our case, a considerable effort would be required by the users in our three scenarios (see Section 3.2.1) in order to interpret the results (e.g., to understand which security issue triggered which legal violation). Having a powerful framework whose results are hardly explainable is deleterious to user experience. As anticipated in Section 3.2.2.1, and Section 3.2.2.2, the common format for security issues and the common language between security issues and the letter of the law need to allow for easily reverting to a natural language explanation. If we label each security issue and each legal finding with a meaningful ID, and link each ID to a natural language explanation, it is possible to create different tables of clickable IDs that expand to the appropriate explanations. We believe a web application exhibiting tables and hyperlinks is a suitable way to present the results of both the integrative analysis, and logical reasoning in a human-friendly fashion.

3.3 MagnetDroid design

In Section 3.2 we stated the goals and aims of MagnetDroid. In this Section we relate those goals and aims to the design of MagnetDroid, and to its partition into its sub-components. In Section 3.3.1 we discuss the high-level architecture of the system, while in Section 3.3.2, Section 3.3.3, and Section 3.3.4 we discuss the architecture of the sub-components.

3.3.1 High level architecture

As discussed in Section 3.2, the top goal of MagnetDroid is to provide an automated bridge that links the security issues found by Android application analysis tools to legal violations and their consequences. As also discussed

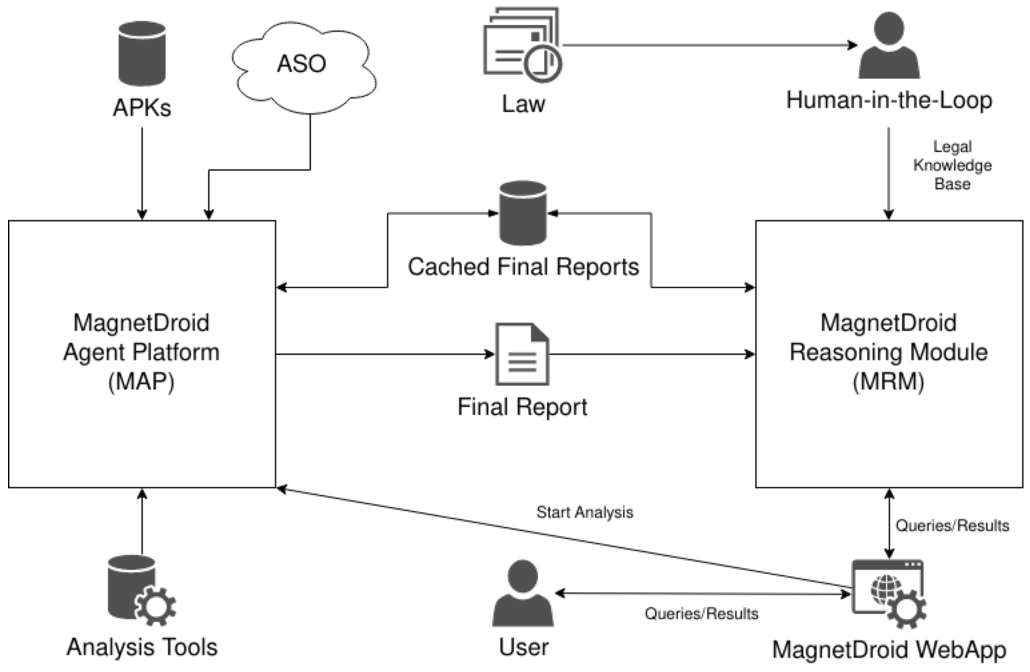

Figure 3.1: MagnetDroid top level architecture: The partition into MAP, MRM and a MWA is visible, as well as the inputs and outputs of each sub-component.

in the same Section, the top goal has four prerequisites. The high-level architecture of MagnetDroid (shown in Figure 3.1) reflects such prerequisites by means of partitioning the system into three main sub-components:

1. A *MagnetDroid Agent Platform* (MAP) whose purpose is to support *integrative analysis* (see Section 3.2.2.1) on Android applications. The goal is to run different tools in parallel on an APK, collect the raw reports, translate them according to an *Android Security Ontology* (ASO), and aggregate them into an ASO-compliant *final report*.

2. A *MagnetDroid Reasoning Module* (MRM) whose purposes are:

- Deriving a *Technological Knowledge Base* (TKB) from the final report.
- Loading a pre-built *Legal Knowledge Base* (LKB), derived from a subset of the 2018 UK DPA.
- Combining the two knowledge bases, and use Prolog reasoning to solve queries on the resulting knowledge base.

3. A *MagnetDroid Web Application* (MWA) whose purposes are:

    - Being the entry point of the system, where users can choose the Android applications to analyse, a subset of the available tools for the integrative analysis, and a subset of queries to run.
    - Displaying and explaining the results to the users in a human friendly fashion.

Section 3.3.2 describes the design of MAP. Section 3.3.3 describes the design of MRM. Section 3.3.4 describes the design of the web application.

3.3.2 A platform for integrative analysis and synthesis

The MagnetDroid Agent Platform (MAP) is an agent-based platform that enables integrative analysis. Its inputs are the APK file corresponding to an Android application, a set of Android application analysis tools, and the Android Security Ontology (ASO - discussed in detail in Section 3.3.2.2). Its output is an ASO-compatible final report that specifies in a standard format all the security issues that the analysis tools have been able to detect with respect to the analysed application. In order to derive the output from the inputs, we separated the flow of MAP into three main phases:

- **Phase 1: Parallel Analysis** which takes the APK and the tools, runs the latter in parallel on the former, and outputs the raw reports.
- **Phase 2: Translation** which, given ASO, for each raw report, produces an ASO-compatible translated reports.
- **Phase 3: Aggregation** which, given ASO and the translated reports, aggregates the latter, detects conflicting information, attempts to solve any such conflict, and produces an ASO-compatible final report.

Figure 3.2 illustrates the flow of the phases within MAP. While technical details, including how agents are employed to perform them, are explained in Chapter 4, the design is discussed in the following Sections.

3.3.2.1 Phase 1: Parallel Analysis

This phase represents the beginning of the integrative analysis of an Android application. Figure 3.3 illustrates the flow from the inputs to the outputs. MAP receives an APK, and a list of application analysis tools. The tools are run in parallel on the APK, and any input to the tools is provided by MAP. Section 4.4 describes the implementation details, the specifics of the protocol used by MAP to interact with each tool, and the currently supported tools. After a tool has completed its analysis (or after a suitable timeout if the tool has no predefined exit point - see Section 6 for details), it produces a report. Note that this is a simplification, as some tools output the report incrementally during the analysis. However, we can safely ignore such output until the

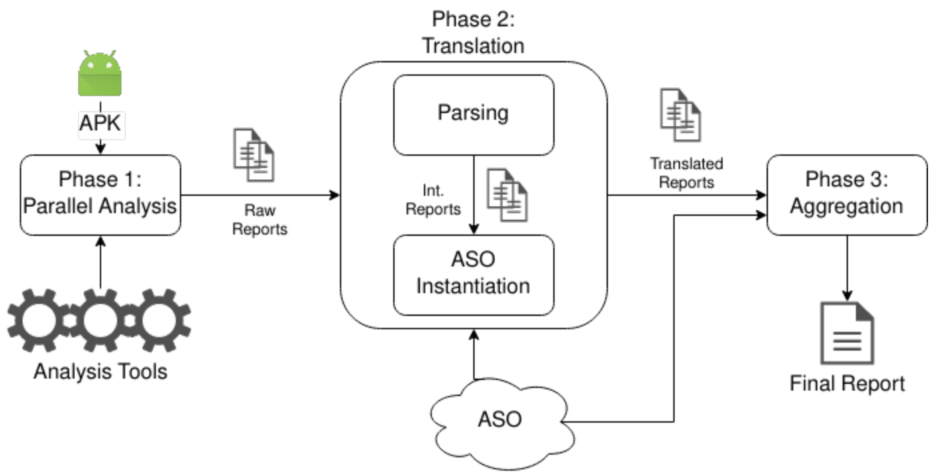

Figure 3.2: MAP architecture: The flow of parallel analysis, translation, and aggregation that produces a final report starting from an Android application, and a set of analysis tools.

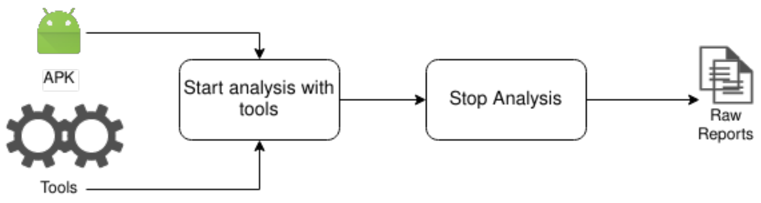

Figure 3.3: Phase 1: Parallel analysis: MAP receives an APK, and a set of tools. Each tool is run in parallel on the APK, with MAP providing inputs (when required), start signals, and stop signals. At the end of the analysis, each tool's raw report is collected.

end of the analysis, and, therefore, call *raw report* the output that is visible after the analysis has terminated. Once all the tools have completed their analysis (Section 4.4 describes what happens when a tool fails to produce a report), the available reports are individually collected. These raw reports in their current format are unusable for our top goal (see Section 3.2), because their syntax and semantics are arbitrary. Therefore, they flow to *Phase 2: Translation*, where they are morphed into something more useful.
As explained in Section 4.4, at the moment, MagnetDroid supports a limited set of tools. That is the case mainly because different tools have different inputs and commands. In Section 4.4 we also explain that, in terms of the modular multi-agent implementation of MAP, implementing support for newer tools (for what concerns *Phase 1: Parallel Analysis*) requires limited additional work. The extensibility of MAP is discussed in more detail in Section 4.7.

3.3.2.2 Android Security Ontology

Before describing any additional phase after *Phase 1: Parallel Analysis*, we need to present our Android Security Ontology (ASO). In this Section we state what ASO consists of, and we illustrate its structure. On the other hand, in Section 6.2 (and its subsections) we discuss our choices for its parameters (mainly the indexes), and the sources for the relevant security concepts, as part of the broader discussion and evaluation of MagnetDroid as a whole. The purpose of ASO is to provide a standard format for reports, in order to tackle the issue of arbitrary syntax and semantics of raw reports which we detailed in Section 2.4.1.2.

The structure of ASO We designed ASO as a tree where the nodes represent security concepts, and each parent-child relation describes either the passage from a more general concept to a more specific one, or a partition of a concept into its sub-components. The root node is the concept of *Android application*. The leaves represent the security issues that can be found by analysing Android applications, or, in some cases, intrinsic properties of Android applications (such as its metadata). One of the advantages of a tree structure is the possibility to group the leaves together by similarity (e.g., the leaf referring to the insecure protocol known as *plain HTTP* is closer to the leaf referring to the obsolete protocol of *SSL 3.0* than the leaf referring to the permission to write the external storage). Figure 3.4 shows the generic tree structure of ASO, while Figure 3.5 illustrates the format of all internal nodes, and leaves. Each leaf, among other internal fields, contains state variables. A state variable represents a particular instance of the security issue that the corresponding leaf encapsulates. For instance, the leaf relating to plain HTTP has a state variable that takes into account whether plain HTTP was observed or not during the analysis. Figure 3.5 illustrates the format of all state variables. A state variable, among other internal fields, has a special field called value. In its standard form, ASO is un-instantiated, i.e., all state variable have empty values.

State variables State variables are the most important components of ASO. They not only encapsulate the details of security issues by means of their *value* field, but also define and support a partial ordering of the security issues according to the following indexes that we propose as part of the model:

- *Absolute Relevance Index* (ARI), an integer 1-10 which represents how relevant the security issue is to data privacy.
- *Minimal Impact Index* (MIII), an integer 1-10 which represents how impactful the identified issue is to data privacy if the best (i.e., the least damaging) case scenario is assumed.
- *Maximal Impact Index* (MXII), an integer 1-10 which represents how impactful the identified issue is to data privacy if the worst (i.e., the most damaging) case scenario is assumed.

Additionally, the value of each index can be classified into one or more equivalence classes: *minor, mild, moderate, severe, critical.* The naming of the classes is derived by merging two existing conventions, and down-scaling them to 5 values. The first convention is defined by Sonar, a static analyser for Java, and comprises *info, minor, major, critical,* and *blocker.* The labels at the extremes are irrelevant in our context, because *info* hints at something that is not an issue, and *blocker* refers to something that prevents the correct functioning of a piece of code, something that does not apply to our analysis. The second convention is usually employed to classify the severity of the symptoms of a disease, and consists of *mild, moderate,* and *severe.* As *major* and *severe* have a similar meaning, we keep only the latter. For all indexes, values 1-2 are considered minor, values 3-4 are considered mild, values 5-6 are considered moderate, values 7-8 are considered severe, and values 9-10 are considered critical. *Absolute Relevance Class* (ARC) is implied from ARI, *Minimal Impact Class* (MIIC) is implied from MIII, and *Maximal Impact Class* (MXIC) is implied from MXII.

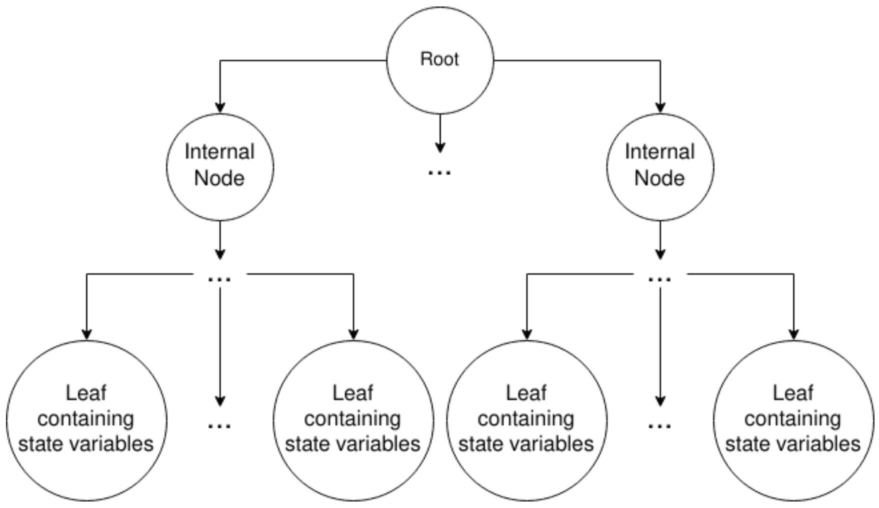
Root
Internal Node
Internal Node
...
Leaf containing state variables
Leaf containing state variables
Leaf containing state variables
Leaf containing state variables

Figure 3.4: ASO structure: A root, the internal nodes, and the leaves containing state variables.

A full visualisation of ASO is available on the DICELab website [1].

Using ASO within MAP By construction, ASO allows for a structured presentation of security issues in a standard format. The only variable elements are the values that are assigned to the state variables, representing the details of security issues. ASO is a crucial component for MAP, because it represents the underlying standard format for security reports. One of the goals of MAP is to translate raw reports into instantiated versions of ASO. Instantiation refers to assigning values to state variables according to the contents of a raw report. It is not mandatory to assign a value to each state variable. Conceptually, this relates to the fact that a report (and, therefore, a tool) does not need to cover all the possible security issues that an Android

[1] https://dicelab-rhul.org/?page_id=121

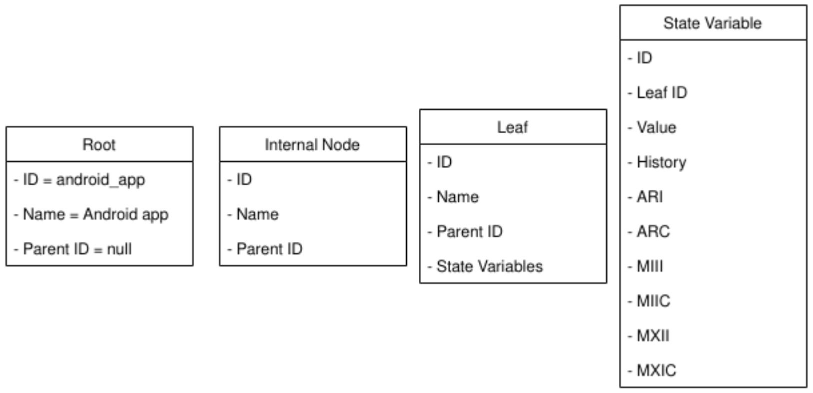

Figure 3.5: ASO building blocks: The structure of the root nodes, of the internal nodes and leaves, and of the state variables.

application might exhibit. Section 3.3.2.3 illustrates the approach that MAP uses to produce instantiated versions of ASO from the raw reports deriving from *Phase 1: Parallel analysis* (see Section 3.3.2.1).

3.3.2.3 Phase 2: Translation

This phase represents the transition from raw reports to translated reports under ASO. Figure 3.6 illustrates the flow of the phase. The inputs are the raw reports produced by *Phase 1: Parallel Analysis*, the relevant pseudo-grammars of the tools (see Appendix 7.2) and ASO (see Section 3.3.2.2). The output is a collection of ASO-compatible translated reports containing all and only the ASO-relevant information in a standardised format. We describe ASO in Section 3.3.2.2. While the translation happens in parallel for each report, each translation flow is fully independent. Therefore, it is sufficient to describe a single flow. In that regard, the translation of a raw report is split into two sub-phases: *report parsing* and *ASO instantiation.*

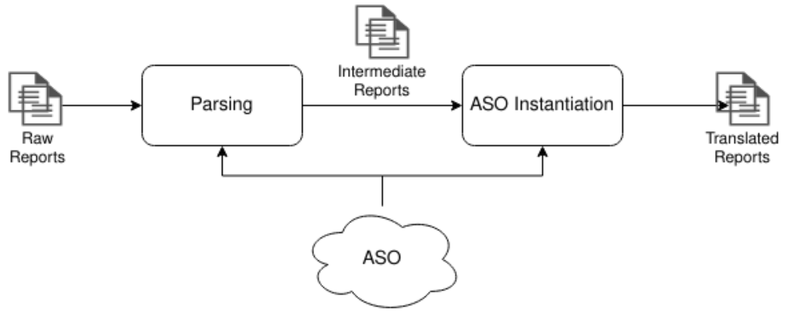

Figure 3.6: Phase 2: Translation: Each raw report from Phase 1 is parsed in order to extract the ASO-relevant content, and remove the ASO-irrelevant content, creating in the process an intermediate report (either narrative, or factual). Then, for each intermediate report, a new translated report is produced, by means of instantiating ASO with the content of the intermediate report itself.

They are both described in the following paragraphs.

Report parsing A raw report usually contains a mixture of ASO-relevant information, and irrelevant fillers. Notable example of the latter are welcome messages, debug messages, and any verbose printout that can be removed without depriving the report of its ASO-relevant content. Additionally, raw reports can be classified into *narrative* and *factual*. The main difference is that time plays a role in narrative reports (i.e., it is possible to define an ordering based on time), while factual reports are a collection of findings whose order is irrelevant. Both considerations (content and time) shape the goals of report parsing:

- Identify the type of the raw report (*narrative* vs *factual*).
- Remove the ASO-irrelevant content.

- Extract the ASO-relevant content to either a (still raw) narrative or collection of facts.

The main idea behind report parsing is to create an intermediate representation of a raw report, so that then an ASO-compatible report is easier to compile from such intermediate representation. Also, splitting the report translation into sub-steps helps creating a less arbitrary and more standardised procedure.

As we discussed above, raw reports are characterised by their arbitrary format. Therefore, identifying the ASO-relevant content from arbitrary reports coming from arbitrary tools is hardly feasible, so we settled for a compromise which, in our opinion, is still a notable improvement with respect to the state-of-the-art. The compromise is that a report from a tool can be parsed (and therefore translated) if a pseudo-grammar of the tool is available. In our context, pseudo-grammar means a collection of machine-parsable directives that either whitelist the format of the ASO-relevant information, or highlight the ASO-irrelevant information that can be find in the reports produced by the tool. An example of pseudo-grammar can be found in Section 4.5.1.2, together with implementation of the parsing protocol. As we detail in Section 4.5.1, MAP currently supports a limited set of tools whose pseudo-grammars were compiled by us during the implementation. For newer tools to be compatible with MAP (for what concerns *Phase 2: Translation*), their pseudo-grammar needs to be supplied, unless the reports are already ASO-compatible, which would allow for the complete skip of translation altogether. The extensibility of MAP is discussed in more detail in Section 4.7.

ASO instantiation The ultimate goal of *Phase 2: translation* is to produce an ASO-compliant report from the content of a raw report. Having parsed a raw report into an intermediate report, whether narrative or factual, the goal of ASO instantiation, as the name suggests, is to use the intermediate report to fill the leaves and state variables of ASO (see Section 3.3.2.2 for an explanation). While the implementation details are left to Section 4.5.2, in this paragraph we describe the approach that MAP uses to instantiate ASO from an intermediate report. We preempt the discussion by saying that such instantiation is only possible if the instantiator has some kind of inside knowledge about which leaves and state variables of ASO are the best match for either the narrative or the facts of the intermediate report. However, as with *Phase 1: Parallel Analysis* and *report parsing*, the structure of MAP allows for extensibility (i.e., the support of a new tool) by adding a small amount of custom code. See Section 4.7 for more details.

Instantiating ASO depends on whether the intermediate report is narrative or factual. In the former case, the narrative as a whole is used to identify the leaves and state variables that need to be updated by the narrative itself. On the contrary, in the latter case, each fact is considered in isolation in terms of the leaves and state variables it needs to update. As explained in Section 3.3.2.2, updating an ASO leaf means to create one or more state variables within that leaf. Likewise, creating a state variable means to get the standard format of a state variable from ASO (given the appropriate leaf), and to instantiate it. Section 4.5.2 offers concrete examples of how the ASO-relevant content from real intermediate reports can be used to create ASO instantiations. We call each instantiated ASO a *translated report*. The output of *ASO instantiation*, as well as the entire *Phase 2: Translation* is a collection of ASO-compliant translated reports. The main advantage such collection has with respect to the collection of raw reports is that the arbitrary syntax and semantics have been replaced by a common format, regardless of the nature and content of the original raw reports.

Translated reports Recall that the format of a translated report reflects ASO. The main consequence of having a collection of translated reports, is the ability to perform any kind of operation on such reports with procedures that are completely agnostic of the nature and content of the original raw reports. Therefore, we can say we are half way through the process of creating unity on the technological side, which is a sub-goal of MagnetDroid, as originally stated in Section 3.2.2.1. The missing step is to integrate the translated report into a final ASO-compliant report that can be considered the final word on the privacy-impacting security issues identified by Android application analysis tools. In other words, aggregating the translated reports is the last step of our *integrative analysis*. Section 3.3.2.4 explains the aggregating approach.

3.3.2.4 Phase 3: Aggregation

The name *integrative analysis* implies that something from different sources is integrated into a product. In our context, we want to integrate reports from Android application analysis tools. Having translated those reports into ASO-compliant versions of themselves, their integration passes through their aggregation under ASO.

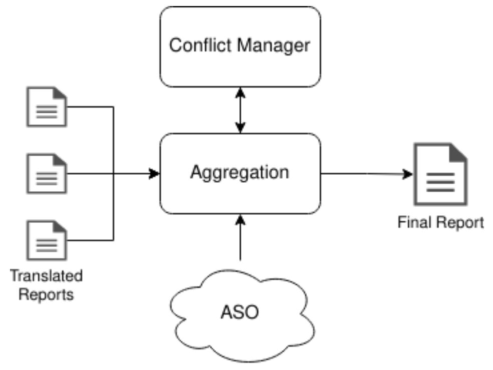

Figure 3.7: Phase 3: Aggregation: Each translated report from Phase 2 is aggregated into a final report by means of instantiating ASO from the content of the translated reports themselves. If one or more reports disagree on an ASO state variable, the conflicting management mechanism engages. Unsolvable conflicts are stored within the final report.

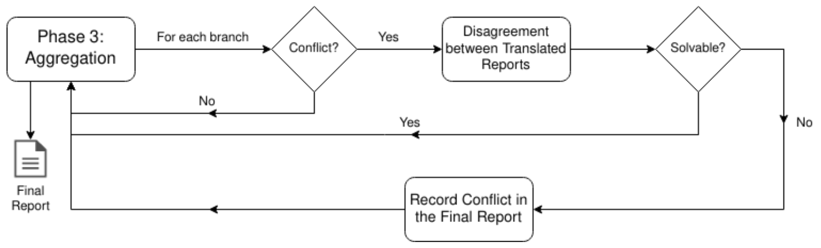

Figure 3.8: Conflict management: If during Phase 3 one or more reports disagree on the value of a state variable, the conflict manager attempts to derive a value that is acceptable nonetheless. If that is not possible, the conflict is recorded within the final report.

Figure 3.7 illustrates the flow of the phase, where the aggregation procedure loops through the branches of ASO. Each branch has exactly one leaf. Such leaf can be either instantiated or not. The latter case emerges when either no tool has found any issue that the leaf represents, or no tool has analysed the application with respect to that issue, or a combination of the two. Regardless of the reason, un-instantiated leaves are not interesting, and, therefore, are skipped. On the other hand, if the leaf is instantiated, one or more instantiated state variables exist within the leaf. As such, the procedure loops through those state variables, and, for each of them, looks whether the translated reports agree on its value. If a report does not exhibit a value for a particular state variable, we skip it for what concerns that state variable. If all the reports agree on the value of a state variable, then such values is copied into the final report. Otherwise, we say we have a conflict.

The implementation of the conflicting mechanism is better explained in Section 4.6.3, while the architecture of the conflict manager is illustrated in Figure 3.8. When a conflict arises between one or more reports on the value of a particular state variable, the procedure attempts to derive a value nonetheless by trying to solve the conflict. Sometimes, a solution can be found, e.g., when a report states that an application is malicious, while another states that it is simply suspicious. In that specific case, the procedure assumes that the application is indeed malicious, and that the second tool simply could not prove its suspicions. However, if the conflict is not solvable (i.e., incompatible values with no reason to choose one over the other), the procedure constructs a special *Conflicting* value, and copies that into the final report. The structure of the conflicting value includes the disagreement. A concrete example of a *Conflicting* value is discussed in Section 4.6.3.

It is worth noting that the *Conflicting* value can support an indefinite number of conflicting parties, some of which may agree among themselves. We decided against using majority voting (or, even worse, plurality voting) as a strategy to solve conflicts, in order to avoid the introduction of false positives, at the expense of potentially introducing false negatives.

The final report Once all the state variables that can be instantiated for every leaf/branch of the *final report* have been instantiated, the aggregation comes to an end, and the final report is its output. As *Phase 3: Aggregation* is the last phase of MAP, the final report is also the output of MAP itself. The final report and the *integrative analysis* that produces it are two contributions on their own right (plus the ASO which is another standalone contribution). In particular, the final report has many uses that transcend its role as mere input to our reasoning framework (as per top goal of MagnetDroid). We speak about some alternative routes with the final report in Chapter 7. For now, the final report represents the success of the integration of the application analysis side of the bridge that we are trying to build between technology and the law. Section 3.3.3 describes the approach of how the final report, together with a model of the law, can be used to build such bridge.

3.3.3 A reasoning platform

The second of the three main sub-components of MagnetDroid is the *MagnetDroid Reasoning Module* (MRM for short). Figure 3.9 illustrates its ar-

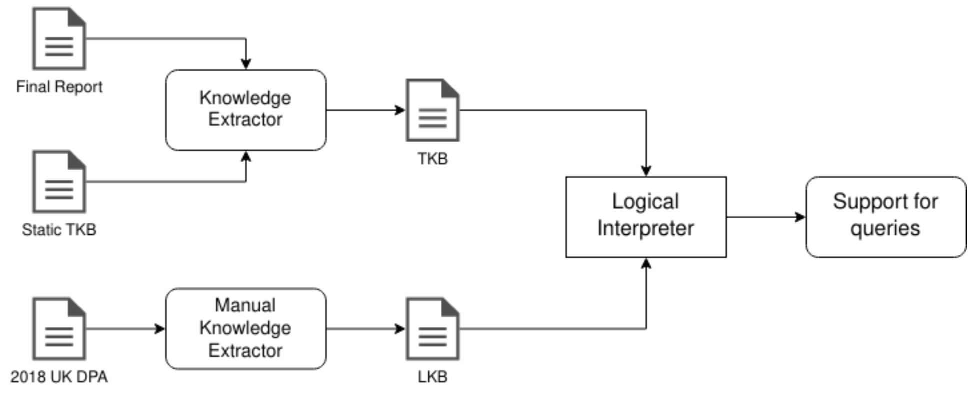

Figure 3.9: MRM architecture: A TKB is built from the final report (from MAP), and some static rules. A LKB is manually derived (once and for all) from the 2018 UK DPA. The union of the knowledge bases is fed into a logical reasoner, in order to support queries.

chitecture. MRM is a module that, given a knowledge base, employs logical reasoning in order to allow for queries. In our context, the queries are related to the legal violations and consequences that privacy-impacting security issues found by Android application analysis have (although a simple list the security issues can also be the subject of queries). The issue is to build a suitable knowledge base that encompasses both the results of Android application analysis, and the relevant law. As we already discussed in Section 3.1, our legal corpus consists of a small subset of the *2018 UK Data Protection Act*. Also, conveniently, MAP is able to produce a *final report* that specifies the identified security issues with regard to a specific Android application. Given these facts, we can characterise the flow of MRM as follows:

- **Phase 4a: Creation of a Technological Knowledge Base** which, given a *final report* from MAP, derives a Technological Knowledge Base

(TKB).

- **Phase 4b: Creation of a Legal Knowledge Base** which derives a Legal Knowledge Base (LKB - essentially an incomplete logic program) from a subset of the 2018 UK DPA.

- **Phase 5: Reasoning** which allows for queries on the union of TKB and LKB.

The naming convention of the three phases of MRM warrants an explanation. *Phase 4* is divided into *Phase 4a*, and *Phase 4b* because only the former consistently happens for each Android application. The reason is that the creation of a LKB can be performed asynchronously, and, more importantly, once and for all (i.e., unlike the TKB, the LKB is the same for every Android application). Its position in the flow simply reflects the need for the LKB to be ready before *Phase 5: Reasoning*. Finally, the reason why MRM starts with *Phase 4* is that the creation of a TKB directly depends of the availability of a final report from MAP. As the MAP phases are numbered 1-3, MRM starts with *Phase 4*, to avoid any possible confusion.

3.3.3.1 Creation of a Technological Knowledge Base

Recall the discussion about the common representation from Section 3.2.2.2. A prerequisite for linking security issues to legal violations is to have both expressed in a common format. At the end of the aforementioned Section, we established *normal logic programs* [47] as the route MagnetDroid pursues in that regard. The first step is to derive a *Technological Knowledge Base* (TKB), expressed as a logic program, that both encompasses the security issues detailed in the final report, and links them to some cardinal properties of security (*data confidentiality*, *data integrity*, and *data authenticity*), according to a mindset, which is discussed in the following paragraph.

Mindset: credulous vs. skeptical As anticipated in Section 3.3.2.2, different findings can be considered issues or not depending on the mindset. MRM supports two opposite mindsets: *credulous*, and *skeptical*. It is important to point out that a mindset is credulous or skeptical with respect to the security of an application, not with respect to the content of the final report. As such, we define *credulous* the mindset by which the gravity of an identified issue is downplayed whenever possible (i.e., the best - least damaging - case scenario is considered). By contrast, we define *skeptical* the mindset by which every single identified issue is always considered with the worst case scenario in mind. The rationale behind these opposite mindsets is to minimise false positives (in the case of *credulous*), and false negatives (in the case of *skeptical*). Note that the positive and negative classes here refer to issues, and are therefore immune from the discussion from Section 3.1.
It is well known that security issues only make sense in the context of a threat model. Bifurcating the interpretation of the identified issues allows for some flexibility while querying for legal violations and consequences related to the identified issues (see Section 3.3.3.3). Section 5.1 describes the implementation details of building a TKB from a final report.

3.3.3.2 Creation of a Legal Knowledge Base

The last step before the bridge between application analysis and the law can be built is to model the law into a *Legal Knowledge Base* (LKB), also expressed as a logic program, that can be unified with the *Technological Knowledge Base* (TKB) described in Section 3.3.3.1. In Section 3.1 we discussed why we limited our coverage of the law to a small subset of the 2018 UK DPA. In particular, our focus begins on Art. 66(1) and Art. 66(2) of Part 3. Art. 66 relates to *security of processing*, and is interesting to us because Android applications do process data, and security issues do arise when data processing lacks certain security properties (namely, confidentiality, integrity, and authenticity). Both Art. 66(1) and Art 66(2) refer to *controllers* and *processors*. They are defined in Art. 32 of Part 3. According to our interpretation of such article, in our context, an Android application plays the role of both. As such, we conceptually merge them into the more compact concept of *party*. We are able to do so, because, as discussed in Section 2.1.4 we are not interested in servers. Therefore, the only party that is subject to our analysis is the Android application itself.

Our LKB contains a translation of our interpretation of the aforementioned articles, linking their content to the same properties (*data confidentiality*, *data integrity*, and *data authenticity*) that can be found within the TKB. Consequently, TKB and LKB have a natural point of contact in those properties, which makes their union suitable for logical reasoning. The implementation details of our LKB, including the process of translation of the law are described in Section 5.2.

3.3.3.3 Reasoning

Once a TKB and a LKB have been produced, we can reason on their union. While the logical queries that the unified knowledge bases support can be arbitrary, we identified some notable classes:

Purely technological queries These queries attempt to derive information from the technological part of the knowledge base. Essentially, they aim at finding the security issues that have been identified by the analysis, without relating them to the law. They can be both comprehensive (i.e., query for all the security issues), or selective (i.e., check whether a specific security issue has been identified). While we did not envision MRM as a system to query a purely TKB (for that we believe analysing the technological final report directly would constitute a better path), these queries are supported nonetheless.

Comprehensive legal queries These queries attempt to find all the possible ways in which all identified security issues trigger any legal violation under a certain mindset (credulous vs. skeptical). In general, they are aimed at "connecting the dots". As the properties of data confidentiality, data integrity, and data authenticity can be violated in multiple ways, and can trigger different legal violations, these queries attempt to cover all the possible connections between security issues, and the aforementioned legal violations.

Selective legal queries These queries attempt to find whether a specific article of the law was violated or not. As such, whether it is violated in 10 different ways or in a single way is irrelevant. The advantage they offer over

comprehensive queries is the possibility to cut the search when a “positive” result (i.e., a violation) is found.
Section 5.3.1 describes the three classes of queries in more detail, including some concrete real-world examples.

3.3.4 A webapp to close the ring

The combination of MAP (see Section 3.3.2) and MRM (see Section 3.3.3) allows for the identification of security issues on Android applications by means of application analysis. It also represents a bridge that, given the aforementioned issues, links them to legal violations. Furthermore, it allows for queries resolved via logical reasoning that return those links, when they exist. However, what such combination lacks is an intuitive and human-friendly way to interact with the system. For users (see Section 3.2.1) to fully benefit from MagnetDroid, they need to interact with it at the following levels:

- Booting the system with a selected application, and a subset of the available analysis tools.
- Selecting the queries to perform without the need of specifying them in any particular logical language.
- Viewing the results, and understanding them.

We believe a web application is a suitable approach to all of them, because of the unique and seamless blending of text and multimedia content that web technologies support. The concrete implementation (including visual examples) is illustrated and discussed in Section 5.4. The *MagnetDroid Web Application* (MWA) is reachable from the DICELab website [2]

3.4 Summary

In this Chapter we introduced MagnetDroid, an agent-based framework that builds a bridge between the domains of Android application analysis and the law. We established the platform assumptions this thesis and MagnetDroid rely upon, and we discussed the goals and aims of MagnetDroid, starting from the top goal, and detailing the sub-goals that arise from it. We also presented MagnetDroid at the design level, linking each high level component to one or more of the aforementioned goals. In particular, we discussed the partitioning into three macro systems:

- the *MagnetDroid Agent Platform* (MAP) supporting the integrative analysis;
- the *MagnetDroid Reasoning Module* (MRM) that, given the output of MAP, and a subset of the 2018 UK Data Protection Act, derives two knowledge bases, and reasons on their union;
- a *MagnetDroid Web Application* (MWA) to explain the results.

With regard to MAP, we also presented the design and implementation of an *Android Security Ontology* (ASO). Among the three main sub-components of MagnetDroid, MAP and MRM deserve a detailed discussion of their im-

[2] `https://dicelab-rhul.org/?page_id=121`

plementation. In Chapter 4 we present the implementation the former, while in Chapter 5 we present the implementation of the latter.

Chapter 4

MagnetDroid Agent Platform

In Chapter 3 we stated the goals of MagnetDroid, and presented an architecture to support them. In particular, we discussed the partition of MagnetDroid into *MAP* (see Section 3.3.2), *MRM* (see Section 3.3.3), and *MWA* (see Section 3.3.4). In this Chapter we present and discuss the implementation details of *MAP*.

We implemented MAP as a multi-agent platform. The rationale behind such choice is discussed in Section 4.1. MAP makes use of an agent model and architecture which we explain in Section 4.2. We then present the agent-oriented high level view of MAP in Section 4.3. We continue our discussion with the agent-oriented implementation of the phases (first described in Section 3.3.2) in Section 4.4 (*Phase 1: parallel analysis*), Section 4.5 (*Phase 2: translation*), and Section 4.6 (*Phase 3: aggregation*). We conclude this Chapter with a discussion regarding the agility and extensibility of our agent-oriented implementation of MAP (see Section 4.7), and a summary (see Section 4.8).

4.1 The opportunity of an agent Platform

In this section we make the case for an agent platform as a convenient abstraction for our system. When looked at from an external perspective, it would seem that the tasks performed by the agents of the *MagnetDroid Agent Platform* (MAP - see Section 3.3.2, and Section 4.3), could also be performed by a lower-level set of abstractions like distributed-objects or scripts. Our case for an agent-based approach relies on the fact that certain properties of agents (see below and Section 4.2) can be used to support a number of features that we argue are key for MAP as explained in Section 4.1.1.

4.1.1 Key Agent Properties

A multi-agent implementation of MAP can benefit from certain innate agent properties, as follows.

Coordination A simple parallel execution of different tasks is not useful by itself, especially when there are dependencies between the agents' activities. Coordination is the way of managing inter-dependencies between activities performed to achieve a goal. This part is important, as we want the system to behave as a coherent unit. The link between agent coordination and cooperation has been researched in the past [52], both for cyberphysical systems [53], and pure software frameworks. [54] Our agent-based approach maps the tasks to different roles which agents can play while assessing an application. More specifically, a ManagerAgent is responsible for the coordination of WorkerAgents, ParsingAgents, TranslatingAgents, and AggregatingAgents. Coordination allows the various components of MAP to perform their task knowing that someone else (i.e., the `MAPCoordinatorAgent` - explained later in Section 4.3.1.2) will be responsible for effectively managing the top-level goal and the inter-dependencies between tasks, including the functioning of the platform. I.e., they can perform their work without the need of constantly checking whether they should keep going or abort, or go idle.

Cooperation A prominent feature of agents is the possibility to cooperate to reach a common goal. The possibility for a cooperative behaviour of software agents has been studied for years. [55] Within MAP we can find multiple points where agent cooperation brings a noticeable benefit to the system. Running the tools in parallel to produce reports, while being aware of failures and/or needs for retries is an example. Translating the raw reports in parallel, so that the AggregatingAgent can then aggregate them into the final report is another example. Cooperation provides a boost in performance with respect to a linear execution, while still allowing for non-trivial *Coordination* via *Communication*.

Communication A prominent feature of agents is the possibility of message passing. As we previously described, *Coordination* is a desirable feature for MAP. Agent communication provides a natural and fluid mean for that to be possible. The semantics of agent communication has been studied in the past, both as a mean to support cooperation [56], and from the pure semantic point of view. [57] Within MAP, agent communication benefits the aggregation of translated reports, among other instances. The `MAPAggregatingAgent` (explained later in Section 4.3.1.2) is always waiting for all the translated reports before starting the aggregation process. Should a fault happen in the chain that produces a translated report, agent communication can be employed to either inform the AggregatingAgent to proceed nonetheless, or to abort the process altogether, depending on the circumstances. In other words, agents communication can be employed to provide additional contextual semantics to uncertain situations. Drawing a parallel with the law, we could say that agent communication provides more information for a better interpretation of the current situation by the recipient(s) [58].

Extensibility The set of inputs that the MAP receives is not meant to be crystallised in time. For instance, new tools could be plugged in, or some of them could be removed at a certain point in the future. Likewise, the focus could shift from security analysis to, for instance, performance analysis. The kind of application we analyse could shift from Android applications to, for instance, non-mobile software (see Section 7.2). We want our system to be as robust as possible, i.e., we would like to minimise the amount of re-coding that is needed in those cases. The sensible design strategy in that regard, is to maximise the invariants and minimise agility, where invariants refers to everything that holds in general, and agility refers to the contextual details that are intrinsically tied with certain elements of MAP (such as the format of the reports coming from a specific tool). An agent platform allows us to create some high-level routines that never change (see Section 4.2 and Section 4.1.2) while concentrating the agility into a small number of places (see Section 4.1.2 for an example regarding the process of parsing raw reports).

Integration of different technologies The integration of different technologies (such as security-oriented application analysis, ontologies, and document translation) is intrinsic to MAP. We believe that a highly structured design is an optimal choice for a framework made of such heterogeneous components. The features of *Coordination*, *Cooperation*, *Communication*, and *Extensibility* make the integration much easier to perform, and easier to adapt to new circumstances (such as plugging in a new tool, or tweaking analysis and/or reasoning while they are already running).

4.1.2 Practical examples

In Section 3.2 we introduced a high level plan consisting of a sequence of goals. In Section 3.3 we described the architecture of the framework that supports them. In particular, in Section 3.3.2 we introduced an agent platform whose ultimate purpose is to produce a final report from an APK, and a set of analysis tools. Furthermore, in Section 3.3.3 we introduced an extension to the agent platform. In this section, we explain why we think that an agent platform is a more appropriate design choice (both in terms of approach and implementation) with respect to other possibilities, such as simpler programs, by referencing some of the concrete tasks that MAP supports.
First of all, looking at the steps required to produce a final report from an Android application, we can identify several tasks:

- Coordinating a parallel analysis.

- Performing an analysis on an APK with a tool.
- Collecting a raw report from the tool.
- Removing the unnecessary information from the raw report.
- Extracting the useful information from the report.
- Instantiating ASO given the extracted content.
- Aggregating several instances of instantiated ASO.

The above tasks are characterised by cyclical actions which need to be performed in a certain order depending on the result of the previous one. Having s/w components that play different roles in the form of agents, each specialised in one or more tasks, is helpful for structuring the flow, and allows for a more granular and natural subdivision of the work. Also, some of the tasks, most notably all of them up to aggregation, can be defined as contextual. For instance, as detailed in Section 4.5.1, parsing a raw report, and extracting the relevant content from it, is highly dependent on the syntax and semantics of the report itself. Recalling that every report has its own arbitrary syntax and semantics, we devised an approach that is reusable no matter which report is given, to an extent, if certain conditions are met. In particular, all the agility is concentrated in a pseudo-grammar that is given in input to the relevant ParsingAgent. Everything else, including the parsing procedure, does not change. What that entails is the possibility to integrate a new tool simply by providing the tool itself, and the corresponding pseudo-grammar. Additionally, one of the known capabilities of agents is message passing. MAP benefits from that in many different ways. For

instance, a user (recall the use cases described in Section 3.2.1) may, on second thought, decide that they are not interested in certain kinds of security problems (together with the corresponding privacy problems) once the flow has already started. Via message passing, the agents could turn off or interrupt those parts of the flow that have become irrelevant or no more needed, given the latest user input (e.g., stopping or not performing the analysis with a tool that was already scheduled or running, or removing content from the instantiated ASO).
Overall, designing MAP as an agent platform allows for modularity, extensibility, and minimisation of agility.

4.2 Agent Model and Architecture

Before explaining our agent implementation of MAP, we need to explain the model and architecture that characterises our agents. We took inspiration from the GOLEM agent environment [59] with regard to both. In particular, each MAP agent is characterised by four main components:

- **Mind**: the core of the reasoning capabilities of the agent.
- **Body**: the outer shell of the agent which contains the mind, and the other components.
- **Sensors**: a set of components which receive perceptions from whatever environment the agent resides in.
- **Actuators**: a set of components which attempt actions in whatever environment the agent resides in.

Each MAP agent, irrespective of its role, shares the same structure for what concerns body, sensor, and actuators. What sets different agents apart is their mind. We further describe each component of a MAP agent in the following paragraphs.

Mind The agent mind encapsulates the *working memory* of an agent, its internal *goals* and *beliefs*, and its reasoning capabilities expressed as a cyclical procedure named *cycle step*. The cycle step can be further divided into its four primitives:

- *perceive* which stores any perception (e.g., messages) that the agent sensors received from the environment since the last mind cycle began.
- *revise* which updates the agent's beliefs according to the latest perception.
- *decide* which selects the next action according to the agent's internal beliefs and, possibly, is current goal.
- *execute* which passes the selected action to the actuators for its attempt.

For each MAP agent, irrespective of its role, *perceive* and *execute* are standard procedures (i.e., they are equal for any two different agents). Consequently, the role of an agent is always determined by its *revise* and *decide* primitives, together with some fixed internal beliefs that are peculiar to a certain role.

Body The agent body serves as a container and medium of communication for the other components. It is important to precise that *communication* in this scenario does not refer to *agent communication*. Instead, it describes how action attempts are transferred from the mind to the appropriate actuators, and how perceptions are transferred from sensors to the mind.

Sensors Each agent has a set of 2 sensors. First, the *physical sensor* which constantly waits to perceive feedback from the environment in response to physical actions (according to some definition of physics which is intrinsic to the environment). Second, the *listening sensor* which constantly listens for communicative actions coming from other agents. In other words, in GOLEM, the perception of the result of physical actions and communicative actions are separated, and we have kept this separation in MAP.

Actuators Every agent has a set of 2 actuators. First, the *physical actuator* which is responsible for attempting physical action within the environment (for some definition of physics which is intrinsic to the environment). Second, the *speaking actuator* which is responsible for communicative actions.

4.2.1 Agent capabilities

The agent model and architecture described in Section 4.2 result in MAP agents possessing a number of general capabilities, described as follows.

- *Time awareness*: thanks to the cycle step, an agent is able to measure the passing time in terms of number of cycles.

- *Physical actions*: thanks to the physical actuator and the physical sensor, an agent is able to perform physical actions within the environment, and receive feedback. As mentioned earlier, the label *physical* only makes sense in the context of the physics of the environment.
- *Agent communication*: thanks to the speaking actuator and the listening sensor, MAP agents support agent communication via message passing.

4.3 High level structure

In Section 3.3.2 we proposed an architecture for the *MagnetDroid Agent Platform* (MAP) that supports *integrative analysis*. In this Section we present and discuss our multi-agent implementation of MAP. We start by describing the entities of MAP (agents and environment) in Section 4.3.1. Then, we present the agent protocol that permeates MAP as a whole in Section 4.3.2.

4.3.1 MAP entities

MAP as a multi-agent platform is characterised by different entities. Essentially, it can be described as an *environment* containing *semi-autonomous agents* and *tool wrappers*, the latter belonging to the active bodies category. Figure 4.1 illustrates the relation between them.

4.3.1.1 MAP Environment

The environment serves both as home for the agents, and as the medium of communication between them. The environment has a state which evolves

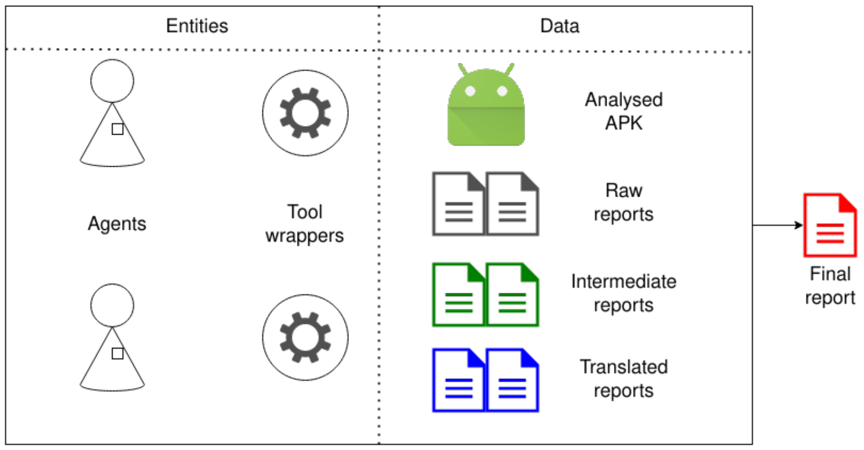

Figure 4.1: MAP high-level structure: A `MAPEnvironment` containing a set of agents, a set of tool wrappers, the analysed APK, and the different kinds of reports. While agents and tools wrappers are considered entities, the APK and the reports are treated as pure data.

as agents attempt actions in it, while interacting with each other, and semi-autonomous bodies. Each and every non-communicative (i.e., physical) action that is performed by an agent on the environment results in the environment sending a feedback to the actor agent. A feedback, among other contextual, optional, and miscellaneous data and metadata, always contain information regarding the outcome of an action attempt. In particular, a feedback is structured as follows (here presented in pseudo-JSON for illustrative purposes):

```
{
    "data": {
        ...
    },
    "metadata": {
        "actor_id": <actor_id>,
        "attempted_action_type: <action_type>,
        "result": "IMPOSSIBLE/SUCCESS/FAILURE",
        ...
    }
}
```

While the `actor_id` value is left parametric, its meaning is self-evident, as it represents the ID of the agent that attempted the particular action (whose ID is also part of the feedback). The ... represent the contextual data and metadata that depend on the specific feedback. The `result` field warrants an explanation. Whenever an agent attempts an action, the environment checks whether the pre-conditions (which may or may not exist) for such action are met. For instance, for an agent to stop a tool's analysis, such analysis must be running at the time the stop action is attempted. If at least one of the pre-conditions is not met, then the environment sends to the actor a feedback where `result` is set to `IMPOSSIBLE`. In that scenario, the environment does not even begin the execution of the action. As such, we can say that an impossible action is guaranteed not to have side effects on the state of the environment. If all the pre-conditions for an action are met, or, if there is no pre-condition to satisfy, then the environment executes the

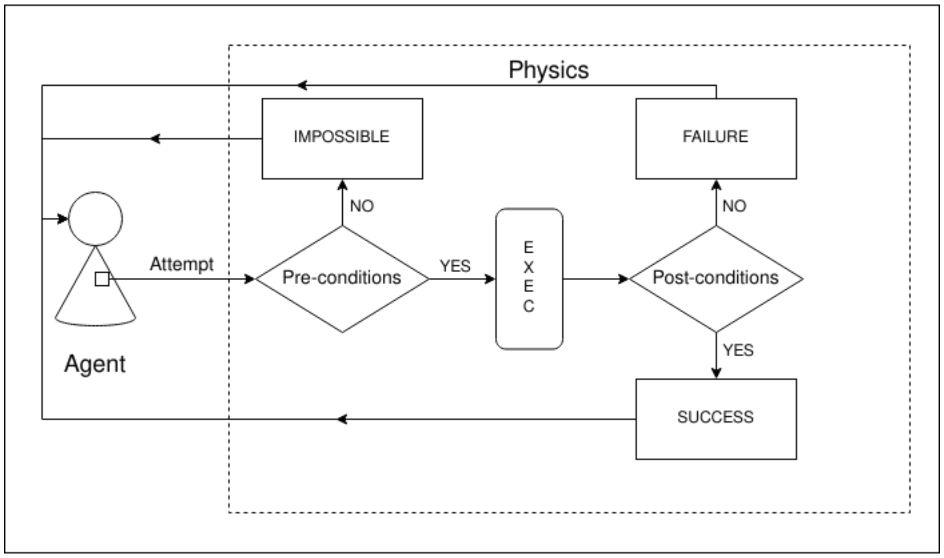

Figure 4.2: Action attempt: If the pre-conditions are not met, the execution is impossible. Otherwise, the execution is performed, and the post-conditions are checked. If they are met, the result is a success, otherwise, it is a failure.

action according to its *physics*. Each action has a (possibly empty) set of post-conditions. After the execution has ended, the environment checks all the post-conditions. If at least one is not met, then a feedback with `result` set to `FAILURE` is sent to the actor. A failed action may or may not alter the state of the environment in a way that the environment itself has no control over. Finally, if no post-condition check fails, a more complex feedback is constructed, depending of the specific action, and sent to the actor. In such feedback, `result` is set to `SUCCESS`. We regard that as the only positive result for an action. A positive result implies that the state of the environment has been successfully altered, assuming the action had side effects. Figure 4.2 showcases the flow of an action attempt within the environment.

On a related note, we often use the terms *agent*, and *actor* interchangeably. In our context, only agents can be actors, and the term *actor* refers to the agent that attempted a particular action. Finally, we define an environment with the properties and behaviours described in this Section as a `MAPEnvironment`, and its internal physics as `MAPPhysics`. Furthermore, a `MAPEnvironment` contains `MAPAgents`, and `MAPTool` objects which interact each other and with the environment itself. Both are described in the following Sections. We discuss `MAPAgents`, and `MAPTool` objects in more detail in Section 4.3.1.2, and Section 4.3.1.3 respectively.

4.3.1.2 MAP Agents

MAP autonomous agents, referred to as `MAPAgents`, are the main actors within MAP. They reside in a `MAPEnvironment`, and are characterised by the model and architecture described in Section 4.2. Within MAP, each `MAPAgent` has a role, specific knowledge and capabilities, in terms of internal knowledge representation and potential actions. The role is the prime factor in determining the knowledge and capabilities of `MAPAgents`. According to their roles, they can be classified as follows.

MAPCoordinatorAgent The first kind of agent we can find in MAP is the `MAPCoordinatorAgent`. Its primary feature is the ability to coordinate other kinds of `MAPAgents`. As such, it does not possess detailed knowledge of or interaction capabilities with the `MAPBody` objects of the `MAPEnvironment`. The actions it can attempt are described in detail in Appendix A. Essentially, the `MAPCoordinatorAgent` is the supervisor that links together the other agents of MAP, and propagates the intermediate products of each phase to the successive phase.

MAPWorkerAgent The second kind of agent we can find in MAP is the `MAPWorkerAgent`. Its primary feature is the ability to interact with a specific `MAPBody` wrapping a certain Android application analysis tool. Consequently, a specific `MAPWorkerAgent` is tightly coupled with a specific tool. For this reason, MAP includes a `MAPWorkerAgent` for every supported tool, each one with slightly different capabilities, in order to accommodate different tools. The actions a `MAPWorkerAgent` can attempt are described in detail in Appendix A.

MAPParsingAgent The third kind of agent we can find in MAP is the `MAPParsingAgent`. Its primary features are the ability to parse a raw report produced by a specific Android application analysis tool, and to produce from it either a *narrative report* or a *factual report*. Consequently, a specific `MAPParsingAgent` is tightly coupled with a specific tool, as it needs to understand the format of its raw reports. For this reason, MAP includes a `MAPParsingAgent` for every supported tool, each one with slightly different capabilities, in order to accommodate different tools. The actions a `MAPParsingAgent` can attempt are described in detail in Appendix A.

MAPTranslatingAgent The fourth kind of agent we can find in MAP is the `MAPTranslatingAgent`. Its primary feature is the ability to instantiate ASO from the content of either a *narrative report* or a *factual report*, producing a *translated report* in the process. As both such kinds of intermediate reports partially preserve the format of the original raw report, a specific `MAPTranslatingAgent` is still tightly coupled with a specific tool. For this reason, MAP includes a `MAPTranslatingAgent` for every supported tool, each one with slightly different capabilities, in order to accommodate different tools. The actions a `MAPTranslatingAgent` can attempt are described in detail in Appendix A.

MAPAggregatingAgent The final kind of agent we can find in MAP is the `MAPAggregatingAgent`. Its primary feature is the ability to aggregate a set of ASO-compliant *translated reports* into a *final report*. Also, it can attempt to solve conflicts (i.e., inconsistent security findings) when they happen while aggregating the translated reports. As all the translated reports are ASO-compliant by design, the `MAPAggregatingAgent` is no more bound to any particular Android application analysis tool. The actions a `MAPAggregatingAgent` can attempt are described in detail in Appendix A.
The `MAPAgents` described above are the primary enablers of the MAP phases described in Section 3.3.2, and in even more detail in Section 4.4 (*Phase 1: parallel analysis*), Section 4.5 (*Phase 2: translation*), and Section 4.6 (*Phase 3: aggregation*).

4.3.1.3 MAP Tool Wrappers

MAP tool wrappers, referred to as `MAPTool` objects, are the final components of MAP. They reside within a `MAPEnvironment` like `MAPAgents`. However, unlike `MAPAgents`, their internal architecture is opaque, and does not match the `MAPAgent` internals. Additionally, they do not comply with the agent model proper of `MAPAgents`. Within MAP, a `MAPTool` is essentially a wrapper for an Android application analysis tool. It serves as the tool interface for the `MAPEnvironment` (and, therefore `MAPAgents`). Its primary purposes are:

- Translating actions to commands that are compatible with the analysis tool proper interface.
- Forwarding the output of the tool to the `MAPEnvironment`.

From its characterisation, it is apparent that a `MAPTool` is tightly coupled with the corresponding tool. Currently, MAP supports `MAPBody` objects that interact with the Android application analysis tools described as follows.

4.3.1.4 Bettercap

Bettercap [60] is an analysis tool that supports MITM attacks on network connections. By convincing an Android application to route its outgoing traffic through a machine where Bettercap is running, it is possible to intercept and modify network connections. In our context, we use Bettercap to analyse the traffic looking for cleartext protocols, insecure protocols in general, and insecure protocol parameters. We also use it to alter some of the server responses, in order to observe how the Android application react (e.g., whether it responds properly to clearly dangerous situations, such as downgrading of secure connections). As Bettercap requires live network traffic to work in any meaningful way, we categorise it as a dynamic analysis tool. Its report provides a narrative of the intercepted network traffic and the MITM "events". Among the ASO-relevant concepts that Bettercap reports contain, we can cite insecure protocols (e.g., plain HTTP, SSL 3.0, FTP, etc.), and insecure protocol parameters (e.g., insecure bulk ciphers such as RC4, vulnerable constructions such as MAC-then-encrypt, etc.).

Why Bettercap We included Bettercap in our pool of available tools because it is a highly-customisable dynamic analysis tool whose report contains, marked by what we call *interesting patterns* (see Section 4.5.1.2), information that can be used to populate the state variables of those leaves of ASO (see Section 3.3.2.2, and Section 4.5.2.2) that refer to network activity security issues.

4.3.1.5 MalloDroid

MalloDroid [1] [61] is a static analysis tool that looks for (dangerously) misconfigured TLS communications within an application's code. Built on the top of AndroGuard [62], it uses the latter to disassemble an Android application, in order to get access to the code. Its report consists of a collection of facts regarding the identified misconfigurations. Among the ASO-relevant concepts that MalloDroid reports contain, we can cite insecure versions of otherwise secure protocols (e.g., SSL 3.0 with respect to TLS, etc.), insecure protocol parameters (e.g., insecure bulk ciphers such as RC4, vulnerable constructions such as MAC-then-encrypt, etc.), and improper validation of X509 certificates.

Why MalloDroid We included MalloDroid in our pool of available tools because it is a static analyser whose report contains, marked by what we call *interesting patterns* (see Section 4.5.1.2), information that can be used

[1]hosted at `https://github.com/sfahl/mallodroid`

to populate the state variables of those leaves of ASO (see Section 3.3.2.2, and Section 4.5.2.2) that refer to network activity security issues.

4.3.1.6 VirusTotal

VirusTotal [63] is a service that checks a file (e.g., an APK) against multiple malware analysis tools, looking for its reputation (e.g., known malware), and known malicious behaviours. Originally a web service [2], it also offers an API which is how we integrated it into MAP. Its report consists of a collection of facts detailing each malware analysis tool's opinion on the APK. For the sake of unifying the results, we consider an Android application malicious if and only if at least one of the results from VirusTotal reports so. Among the ASO-relevant concepts that VirusTotal reports contain, we can cite the reputation that an application has (e.g., malware).

Why VirusTotal We included VirusTotal in our pool of available tools because it is a service that consults a set of anti-malware software in order to determine the reputation of the analysed APK. As such, it is a good choice for producing information that can be used to populate the state variables of those leaves of ASO (see Section 3.3.2.2, and Section 4.5.2.2) that refer to the reputation of an Android application.

4.3.1.7 CSP Checker

CSP Checker is a dynamic analysis tool we built on the top of Bettercap that checks how an application reacts to an injected Content-Security-Policy

[2]hosted at `https://www.virustotal.com`

(CSP) in web communications. In particular, by looking at the resulting network traffic after the injection, it is possible to infer (at least partially) how the application reacted to the CSP, and whether it followed its prescriptions (e.g., loading vs. not loading certain scripts which, if loaded, cause additional network traffic). Its report consists of a narrative highlighting improper reactions from the analysed Android application. Among the ASO-relevant concepts that Bettercap reports contain, we can cite the incorrect loading of restricted resources.

Why CSP Checker We included CSP Checker in our pool of available tools because, in conjunction with Bettercap, it provides information that can be used to populate the state variables of those leaves of ASO (see Section 3.3.2.2, and Section 4.5.2.2) that refer to network activity security issues, in particular behaviours that are inconsistent with what is prescribed by a received Content-Security-Policy.

4.3.1.8 OCSP Checker

OCSP Checker is a dynamic analysis tool we built on the top of Bettercap that checks how an application reacts to various responses from a simulated OCSP server which is supposed to check the status of an X509 certificate. The available tests vary from an unresponsive OCSP server (in which case it is acceptable for an Application to soft-fail - i.e., ignore the server and proceed as if no issue was encountered), to an improperly signed response. Its report consists of a narrative highlighting improper reactions from the analysed Android application. Among the ASO-relevant concepts that Bettercap reports contain, we can cite the soft-fail approach to an OCSP server unavailability when the certificate to validate exhibit the OCSP-Must-Staple extension.

Why OCSP Checker We included OCSP Checker in our pool of available tools because, in conjunction with Bettercap, it provides information that can be used to populate the state variables of those leaves of ASO (see Section 3.3.2.2, and Section 4.5.2.2) that refer to network activity security issues, in particular behaviours that are inconsistent with the correct actions that have to be performed when an *OCSP-Must-Staple* [64] extension is received.

4.3.2 MAP agent protocol

While MAP is conceptually divided into three phases (see Section 3.3.2), it is possible to establish an uninterrupted flow of activities that spans all the phases. In our multi-agent implementation of MAP, some of those activities require *cooperation* and *coordination* between different agents, which in turn leads to different *roles*, and *communication* as a mean to cooperate and coordinate with each other. Figure 4.3 illustrates how different agents (whose roles are explained in Section 4.3.1.2) use communication.

The high-level protocol starts with a `MAPCoordinatorAgent` (CoA) sending analysis jobs to a pool of `MAPWorkerAgents` (WoA). Once a WoA has finished the analysis, which happens without any need for further agent communication, it sends either a raw report, or an error message to the CoA. As Figure 4.3 shows, that is the extent of *Phase 1: parallel analysis* for what

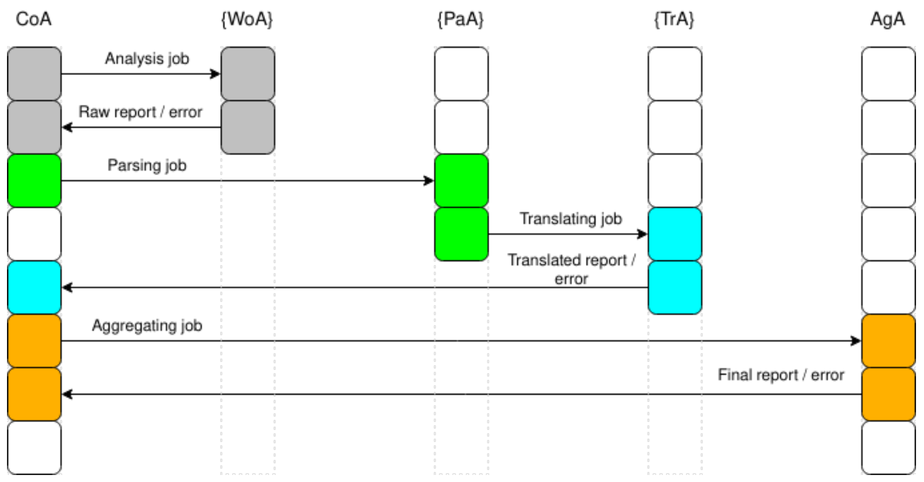

Figure 4.3: MAP agent protocol: Different agents pass messages to each others in order to cooperate and coordinate with each other. The graduated columns represent agents with different roles as time passes. Within each graduation, agents may or may not perform additional actions that do not require message passing. Curly brackets indicate a pool of agents, as opposed to a single agent. Labelled arrows represent messages, and the "/" symbol indicates and separates two possible alternative messages. Different colours highlights the phases of MAP.

concerns the communication protocol.

Upon receiving the raw reports from all WoAs, the CoA sends them to a pool of `MAPParsingAgents` (PaA). Once a PaA has parsed a report, which happens without any need for further communication, it sends an intermediate report to a `MAPTranslatingAgent` (TrA) which, behind the scenes, uses it to instantiate ASO, producing a translated report in the process. Such report is then sent back to the CoA. As Figure 4.3 shows, that is the extent of *Phase 2: translation* for what concerns the communication protocol. Two colours are used to highlight the internal division of the phase into parsing and ASO instantiation.

Upon receiving the translated reports from all TrAs, the CoA send them to a `MAPAggregatingAgent` (AgA) which, behind the scenes, aggregates them into an ASO-compliant final report. Such report is then sent back to the CoA which outputs it, causing the MAP flow to end. As Figure 4.3 shows, that is the extent of *Phase 3: aggregation* for what concerns the communication protocol.

While the agent protocol highlights how different agents communicate with each other in order to coordinate and cooperate, it treats some of the agent actions as black boxes, namely all the processes which produce the reports that are then exchanged via communication. Section 4.4, Section 4.5, and Section 4.6 describe those black boxes in detail, in terms of the algorithms that the relevant agents follow in other to produce the reports that are exchanged via agent communication according to the agent protocol. In particular, we highlight how the agent decision process can be deconstructed into *condition-action rules*, where the conditions check the internal beliefs of the agent which, as discussed in Section 4.2, are updated by the *revise* primitive.

4.4 Phase 1: parallel analysis

In Section 3.3.2.1 we described at the design and architectural level the process of running Android application analysis tools on an APK, and collecting the reports produced by the tools themselves. In this Section we explain how we employ some of the agents described in Section 4.3.1.2 in order to support the flow of *Phase 1: parallel analysis* (see Section 4.4.1) thanks to an algorithmic behaviour (explained in Section 4.4.2), and give concrete examples of the raw reports that are collected at the end of the phase (see Section 4.4.3).

4.4.1 The flow

According to the agent protocol described in Section 4.3.2, the actors of *Phase 1: parallel analysis* are the `MAPCoordinatorAgent` (CoA), and a pool of `MAPWorkerAgents` (WoA), one for each analysis tool. The CoA sends an analysis job to the WoAs, together with information on where to find the APK to analyse and the tool to analyse it with. Each WoA is responsible for a particular analysis tool. The agent-tool environmental interaction happens via an interface exposed by a `MAPTool` (ToW) acting as a wrapper for the tool. According to the agent protocol, after sending the jobs to the WoAs, the CoA waits for either a raw analysis report or an error message from each WoA. Such flow of *Phase 1: parallel analysis* is illustrated in Figure 4.4. From the point of view of a single WoA, analysing an APK with a tool can

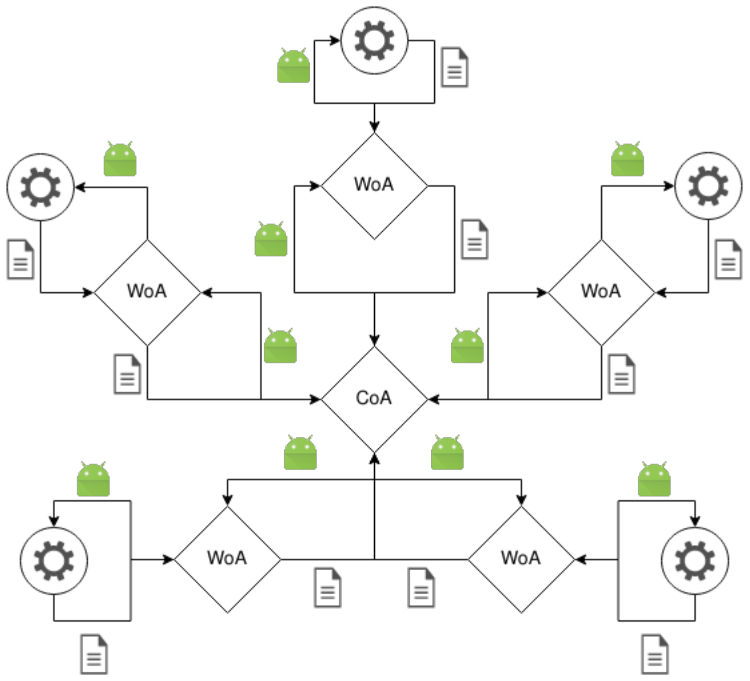

Figure 4.4: Phase 1: Parallel analysis: The `MAPCoordinatorAgent` (CoA) sends the APK to the pool of `MAPWorkerAgents` (WoA) which start the parallel analysis by interacting with the `MAPBody` objects wrapping the tools. After the analysis has finished, each `MAPWorkerAgent` collects the raw report, and sends it to the `MAPCoordinatorAgent`.

be described with the algorithm illustrated in Figure 4.5, and explained in Section 4.4.2.

4.4.2 The algorithm

Whenever a WoA receives a job message, it starts the analysis by running the tool on the APK via its ToW. After the analysis has started, the WoA wait for feedback. Whenever the tool provides a feedback requesting for inputs, the WoA provides them. If the tool provides a feedback informing the WoA

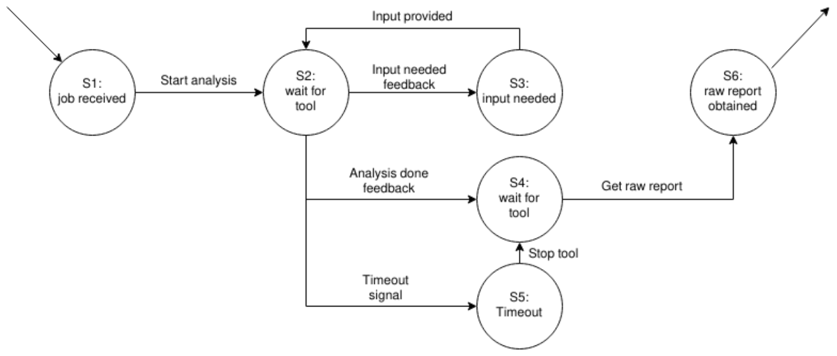

Figure 4.5: Phase 1 algorithm: A state diagram illustrating the conceptual states from a WoA point of view, together with the actions and/or feedbacks that cause state transitions.

that the analysis has finished, then the WoA retrieves the raw report from the tool. If the tool provides a feedback informing the WoA that the analysis ended with an error, the WoA does not attempt to collect any report. Finally, if the tool does not report either the end of the analysis, or an error within 10 minutes, the WoA internal timeout is triggered, and the analysis if forcefully stopped. In that case, if a report is available, it is retrieved.

Condition-action rules As mentioned in Section 4.3.2, the *revise* and *decide* primitives of the agent's mind can be used to implement the algorithm that the WoA follows. Furthermore, the decision process can be modelled with *condition-action rules*, according to a *teleoreactive* model [65]. For a WoA, the behaviour looks as follows.

```
#   Condition          -> Action

Analyse: {
    new_job_received  -> start_analysis
    input_needed      -> provide_input
    analysis_timeout  -> stop_analysis
    report_available  -> retrieve_report
    report_received   -> send_report
    default           -> stay_idle
}
```

The meaning of the listing above is that the conditions are checked in order, and the first that is met triggers the corresponding action attempt. If no condition is met, the *default* condition at the end will always be true. There are also sub-goals, so this model allows for structure, and can be parameterised.

4.4.3 Raw reports

The products of *Phase 1: parallel analysis* are the so called *raw reports*. As previously discussed, their format is generally arbitrary, which is one of the reasons why we built more phases into MAP. Below is an extract from a raw report produced by the *Bettercap* tool. The much longer complete report is visible in Appendix A.

```
  192.168.1.117/24 > 192.168.1.3  >> [14:10:21] [net.sniff.leak.http]
      http local POST example.com Mozilla/5.0 (X11; Android arm; rv:78.0)
          Gecko/20100101 Firefox/78.0
    Method: POST
    URL: /
    Headers:
      Host: example.com
      User-Agent: Mozilla/5.0 (X11; Android arm; rv:78.0) Gecko/20100101 Firefox/78.0
      Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
      Accept-Language: en-US,en;q=0.5
      Accept-Encoding: gzip, deflate
      Connection: keep-alive
      Upgrade-Insecure-Requests: 1
      Cache-Control: max-age=0
    Form:
      mgtxt => This message will be intercepted.
      nmtxt => U. N. Owen
      action => send
```

4.5 Phase 2: translation

In Section 4.4 we described *Phase 1: parallel analysis* from a multi-agent point of view. In this Section we present and discuss the multi-agent implementation of *Phase 2: translation*. The goal of this phase is to transition from reports whose format is arbitrary and depends on the specific tool to reports whose format is standard and dictated by *ASO*. We call such reports *translated reports*. The flow of the phase can be logically partitioned into *report parsing*, and *ASO instantiation*. The former is discussed in Section 4.5.1, while the latter is discussed in Section 4.5.2. The flow of the phase as a whole is illustrated in Figure 4.6.

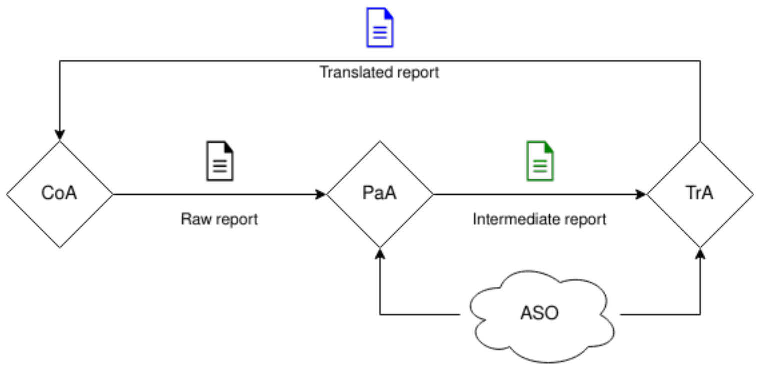

Figure 4.6: Phase 2: Translation: Visualisation of a branch: the `MAPCoordinatorAgent` (CoA) sends a raw report to a `MAPParsingAgent` (PaA) which parses it, creating an intermediate report. Then the `MAPParsingAgent` (PaA) sends the intermediate report to a `MAPTranslatingAgent` (TrA) which instantiates ASO from the content of such report, creating a translated report. Then, the `MAPTranslatingAgent` sends the newly produced translated report to the `MAPCoordinatorAgent`.

4.5.1 Parsing

The objective of the first half of *Phase 2: translation* is, for each raw report, to produce an intermediate report which contains only the ASO-relevant information that can be derived from the corresponding raw report. The flow of this sub-phase is described in Section 4.5.1.1, while the algorithmic agent behaviour is explained in Section 4.5.1.2.

4.5.1.1 The flow

According to the agent protocol described in Section 4.3.2, the *report parsing* sub-part of *Phase 2: translation* has only two groups of actors: the `MAPCoordinatorAgent` (CoA), and the set of `MAPParsingAgents` (PaA). For the sake of explainability, it is convenient to focus on a single branch. Under

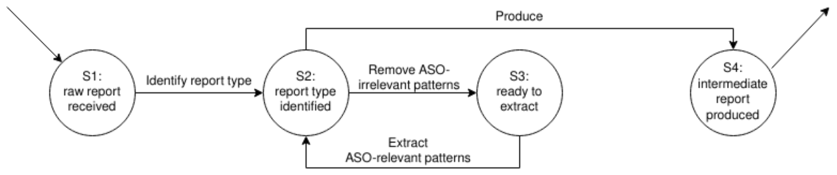

Figure 4.7: Phase 2a algorithm: A state diagram illustrating the conceptual states from a WoA point of view, together with the actions and/or feedbacks that cause state transitions.

that assumption, the actors are reduced to the CoA, and a single PaA. The CoA sends a parsing job to the AgA, together with a raw report from *Phase 1: parallel analysis*. The PaA then parses the report, producing an intermediate report. According to the agent protocol, after sending the job to the PaA, the CoA does not wait for any response from the PaA. Instead, it waits for a message downstream that will be sent at the end of the full phase. Such flow of *Phase 2a: report parsing* is visible on the left-hand side of Figure 4.6. From the point of view of the PaA, producing the intermediate report can be described with the algorithm illustrated in Figure 4.7, and explained in Section 4.5.1.2.

4.5.1.2 The algorithm

Whenever a PaA receives a job message and a raw report, its first action is to identify the report type. In particular, if the raw report exhibits a narrative (i.e., time is featured), then it knows it has to produce a *narrative* intermediate report. On the other hand, if the raw report exhibits a collection of facts (i.e., time is not featured), it knows it has to produce a *factual* intermediate report. Then, guided by the *pseudo-grammar* (which is part of its beliefs), and ASO, the PaA starts a loop of removing all identifiable ASO-irrelevant information, and extracting and storing all the identifiable ASO-relevant information from the raw report. The loop ends when there is nothing more to remove or extract. Once that happens, the PaA produces the intermediate report according to the extracted content.

Pseudo-grammars Whenever a TrA needs to parse a raw report, it needs a so-called *pseudo-grammar* that describes the reports of the tool that produced such raw report. A pseudo-grammar consists of different collections of *known uninteresting patterns*, *potentially interesting patterns*, and *interesting patterns*, plus some metadata (such as the expected kind of report - narrative or factual). Patterns are expressed either via regular expression, or in full (hence the *pseudo* prefix). A pattern is

- *Uninteresting* if all instances of matching content can be removed without any loss of meaningful information (with respect to ASO).
- *Interesting* if all instances of matching content are expected to contain meaningful information (with respect to ASO).
- *Potentially Interesting* if its value depends on the presence or absence of other patterns.

An example of a pseudo-grammar is shown in Appendix C.

Condition-action rules As mentioned in Section 4.3.2, the *revise* and *decide* primitives of the agent's mind can be used to implement the algorithm that the PaA follows. Furthermore, the decision process can be modelled with *condition-action rules*, as with *Phase 1: parallel analysis*. For a PaA, the behaviour looks as follows.

```
# Condition               -> Action

Parse: {
    new_job_received          -> identify_report_type
    report_type_identified    -> start_loop
    can_remove                -> remove
    can_extract               -> extract
    empty_raw_report          -> produce_intermediate_report
    default                   -> stay_idle
}
```

The same considerations regarding the meaning of condition-action rules described in Section 4.4.2 still apply.

Narrative vs. Factual reports The product of *report parsing* is an intermediate report. Its only purpose is to guide the second sub-phase of *Phase 2: translation*, known as *ASO instantiation*. The reason behind the split representation is that the procedure that instantiates ASO from a narrative is slightly different from the procedure that instantiates ASO from a collection of facts. Format-wise, a narrative report features an ordered list of raw events, each with its own relative timestamp.

```
happens_at(<raw_event_1>, 1).
...
happens_at(<raw_event_N>, N).
```

On the contrary, a factual report is an unordered collection of facts, where a fact is a piece of raw relevant content.

```
holds(<fact_1 >).
...
holds(<fact_N >).
```

Although the format of both narrative and factual reports may be reminiscent of Prolog (especially Event Calculus [66]), that is just a coincidence. Intermediate reports are not Prolog programs, and any assumption that their syntax is valid Prolog syntax is incorrect. Intermediate reports are never fed into any Prolog interpreter. Instead, they are the main input to the *ASO instantiation* sub-phase of *Phase 2: translation*.

4.5.2 ASO instantiation

The objective of the second half of *Phase 2: translation* is, for each intermediate report, to instantiate ASO with the content of the aforementioned intermediate report. The flow of this sub-phase is described in Section 4.5.2.1, while the algorithmic agent behaviour is explained in Section 4.5.2.2.

4.5.2.1 The flow

According to the agent protocol described in Section 4.3.2, the *ASO instantiation* sub-part of *Phase 2: translation* has only three groups of actors: the set of `MAPParsingAgents` (PaA), the set of `MAPTranslatingAgents` (TrA), and the `MAPCoordinatorAgent` (CoA). For the sake of explainability, it is convenient to focus on a single branch. Under that assumption, the actors are reduced to a single PaA, a single TrA, and the CoA. The PaA sends a translating job to the TrA, together with the intermediate report it just pro-

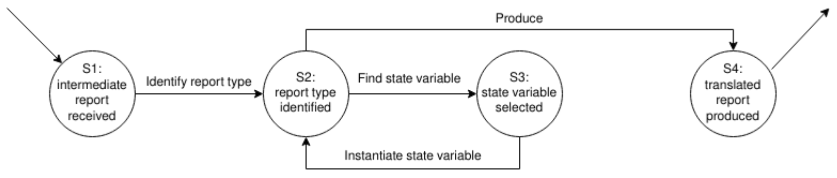
Produce
S1: intermediate report received
Identify report type
S2: report type identified
Find state variable
S3: state variable selected
Instantiate state variable
S4: translated report produced

Figure 4.8: Phase 2b algorithm: A state diagram illustrating the conceptual states from a WoA point of view, together with the actions and/or feedbacks that cause state transitions.

duced in *Phase 2a: report parsing*. The TrA then instantiates ASO from the content of the translated report, producing a translated report. According to the agent protocol, the CoA is still waiting for either a translated report or an error message from the TrA. Such flow of *Phase 2b: ASO instantiation* is visible on the right-hand side of Figure 4.6. From the point of view of the TrA, producing the translated report can be described with the algorithm illustrated in Figure 4.8, and explained in Section 4.5.2.2.

4.5.2.2 The algorithm

Whenever a TrA receives a job message and an intermediate report, its first action is to identify the report type. If the raw report exhibits a narrative (i.e., time is featured), then it knows that, potentially, the entire narrative may need to be used in order to identify the relevant ASO leaves and state variables. On the other hand, if the raw report exhibits a collection of facts (i.e., time is not featured), it knows that each fact potentially refers to a distinct leaf and/or state variable. Then, guided by the *pseudo-grammar* (which is part of its beliefs), and ASO, the TrA starts a loop of identifying a state variable whose value can be derived from either the narrative, or a fact.

Once the value has been assigned to the state variable, the loop continues until there is nothing more in the intermediate report to instantiate any state variable with. Once that happens, the PaA has successfully produced a translated report.

Condition-action rules As mentioned in Section 4.3.2, the *revise* and *decide* primitives of the agent's mind can be used to implement the algorithm that the TrA follows. Furthermore, the decision process can be modelled with *condition-action rules*, as with *Phase 1: parallel analysis*. For a PaA, the behaviour looks as follows.

```
# Condition                        -> Action

Translate : {
    new_job_received               -> identify_report_type
    report_type_identified         -> start_loop
    can_instantiate_sv             -> instantiate_sv
    empty_intermediate_report      -> produce_translated_report
    default                        -> stay_idle
}
```

The same considerations regarding the meaning of condition-action rules described in Section 4.4.2 still apply.

Translated reports As discussed multiple times, the format of a translated report reflects ASO. The following listings show an extract from a translated report. In particular, a leaf and one of its state variables are shown.

```
{
    "id": "ocsp_behaviour",
    "name": "OSCP behaviour",
    "parent_id": "x509_status_checks",
    "state_variables": [
        {
            "id": "unreachable_server_with_ocsp_must_staple",
            "leaf_id": "ocsp_behaviour",
            "value": "soft-fail",
            "history": [],
            "ari": 10,
            "arc": critical,
            "miii": 1,
            "miic": "minor",
            "mxii": 10,
            "mxic": "critical"
        }
    ]
}
```

The main consequence of having a collection of translated reports, is the ability to perform any kind of operation on such reports with procedures that are completely agnostic of the nature and content of the original raw reports. Therefore, we can say we are half way through the process of creating unity on the technological side, which is a sub-goal of MagnetDroid, as originally stated in Section 3.2.2.1. The missing step is to integrate the translated report into a final ASO-compliant report that can be considered the final word on the privacy-impacting security issues identified by Android application analysis tools. In other words, aggregating the translated reports is the last step of our *integrative analysis*. Section 4.6 describes how the aggregating works from an multi-agent implementation point of view.

4.6 Phase 3: aggregation

In Section 4.5 we described *Phase 2: translation* from a multi-agent point of view. Its output - a set of *translated reports* - feature as input to *Phase*

3: aggregation, which is the subject of this Section. The goal of this phase is to aggregate all the translated report into an ASO-compliant *final report*, which is also the only product of MAP as a whole. The flow of the phase is described in Section 4.6.1, while the algorithmic agent behaviour is explained in Section 4.6.2. We discuss how conflicts between reports are managed in Section 4.6.3, and we also discuss the final report from the conceptual and practical points of view in Section 4.6.4.

4.6.1 The flow

According to the agent protocol described in Section 4.3.2, *Phase 3: aggregation* has only two actors: the `MAPCoordinatorAgent` (CoA), and the `MAPAggregatingAgent` (AgA). The CoA sends an aggregating job to the AgA, together with the translated reports from *Phase 2: translations*. The AgA creates a new instantiation of ASO (called *final report*), and derives the values of its state variables from the corresponding values in the translated reports. According to the agent protocol, after sending the job to the AgA, the CoA waits for either the final report or an error message from the AgA. Such flow of *Phase 3: aggregation* is illustrated in Figure 4.9. From the point of view of the AgA, producing the final report can be described with the algorithm illustrated in Figure 4.10, and explained in Section 4.6.2.

4.6.2 The algorithm

Whenever an AgA receives a job message and the set of translated reports, it creates a new ASO instantiation called *final report*, and loops through its state variables. For each state variable, it checks whether all the translated

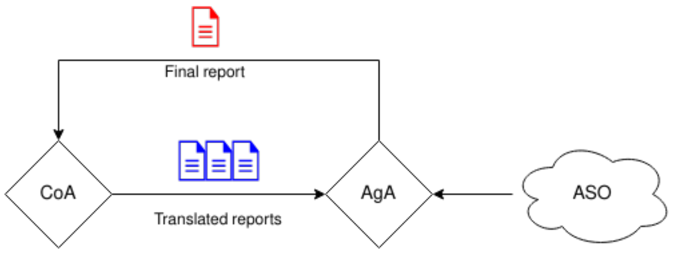

Figure 4.9: Phase 3: Parallel analysis: The `MAPCoordinatorAgent` (CoA) sends the translated reports to the `MAPAggregatingAgents` (AgA) which aggregates them into a final report. After such final report has been produced, the `MAPAggregatingAgent` sends it to the `MAPCoordinatorAgent`.

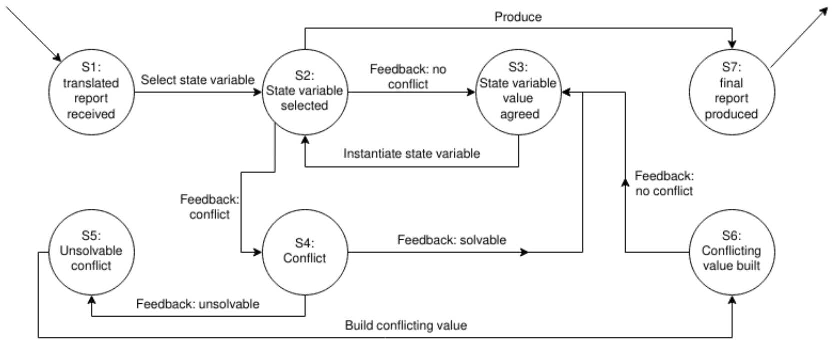

Figure 4.10: Phase 3 algorithm: A state diagram illustrating the conceptual states from the AgA point of view, together with the actions and/or feedbacks that cause state transitions.

reports agree on its value. For our purposes, translated report that do not assign a value to that particular state variable are simply ignored. If an agreed value can be found, then it is assigned to that particular state variable in the final report. Otherwise, a conflict arises, and the AgA attempts to solve it, by deriving a value that is "acceptable", given the actual values for all the translated reports. If the AgA is able to derive such value, then the conflict is solved. Otherwise, the AgA creates a custom *Conflicting* value (see Section 4.6.3), and assigns it to the appropriate state variable in the final report. Once all state variables in the final report have been looped through, and a value has been assigned to each of them (if possible), then the AgA has successfully produced the final report.

Condition-action rules As mentioned in Section 4.3.2, the *revise* and *decide* primitives of the agent's mind can be used to implement the algorithm that the AgA follows. Furthermore, the decision process can be modelled with *condition-action rules*, as with *Phase 1: parallel analysis*. For an AgA, the behaviour looks as follows.

```
# Condition                    -> Action

Aggregate : {
     new_job_received          -> start_aggregation
     state_variable_available  -> select_state_variable
     state_variable_selected   -> look_for_value
     no_conflict               -> assign_value
     conflict                  -> manage_conflict
     conflict_solved           -> assign_value
     conflict_unsolvable       -> assign_conflicting
     default                   -> stay_idle
}
```

The same considerations regarding the meaning of condition-action rules described in Section 4.4.2 still apply.

4.6.3 Conflict solving mechanism

Whenever a state variable in the final report needs a value, that value is derived from the available translated reports. However, it is possible that at least two translated reports exhibit a different value for the same state variable. In Section 3.3.2.4 we discussed how, in that scenario, a conflict manager attempts to solve the conflict, and derive a value. In our implementation, the conflict manager is the AgA. A conflict is solvable if a value *compatible* with all the values from the translated reports can be derived or inferred. A notable case of solvable conflict is when a state variable from a translated report labels the application as *malicious*, while the same state variable from another translated report labels the application as *suspicious*. In that scenario, the AgA assumes that the application is malicious, and that the second tool simply failed to prove it, despite having suspicions. However, many conflicts are not solvable, as there is no apparent reason to select the value from a specific translated report rather than the value from another translated report. However, if the conflict is not solvable, the procedure constructs a special *Conflicting* value, and copies that into the final report. The structure of the conflicting value includes the disagreement. For instance, assuming that *report A* and *report B* disagree on the value of the state variable `S1`, the *Conflicting* value has the following structure:

```
{
    "type": "unsolvable_conflict",
    "disagreement": [
        <report_A>: <report_A_value>,
        <report_B>: <report_B_value>
    ]
}
```

It is worth noting that the *Conflicting* value can support an indefinite number of conflicting parties, some of which may agree among themselves. In our implementation, we decided against majority voting (or, even worse, plurality voting), in order to avoid the introduction of false positives, at the expense of potentially introducing false negatives.

4.6.4 The final report

As previously discussed in Section 3.3.2.4, once the TrA has instantiated all the state variables that can be instantiated (according to the translated reports) for every leaf/branch of the *final report*, the aggregation comes to an end, and the final report is its output. As *Phase 3: Aggregation* is the last phase of MAP, the final report is also the output of MAP itself. The following listing shows an extract from a final report containing a conflicting value. In the particular example, and one of its state variables are shown.

```
{
    "id": "ocsp_behaviour",
    "name": "OSCP behaviour",
    "parent_id": "x509_status_checks",
    "state_variables": [
        {
            "id": "unreachable_server_with_ocsp_must_staple",
            "leaf_id": "ocsp_behaviour",
            "value": {
                "type": "unsolvable_conflict",
                "disagreement": [
                    "bettercap": "hard-fail",
                    "oscp-checker": "soft-fail"
                ]
            },
            "history": [],
            "ari": 10,
            "arc": critical,
            "miii": 1,
            "miic": "minor",
            "mxii": 10,
            "mxic": "critical"
        }
    ]
}
```

4.7 MAP extensibility and agility

One of the intrinsic challenges MAP is how to manage the arbitrary format of the reports produced by a certain tool. Conversely, one of the goals of MAP is to standardise as much as possible the flow that ultimately produces a *final report* from an APK and a set of analysis tools. Given that the arbitrary format is an issue that cannot disappear as long as analysis tools do not settle on a common report format (which is unlikely, in our opinion), the only way to maximise the amount of standardised (i.e., reusable) architectural components and code is to minimise the number of points where *agility* is concentrated. In our context, a segment of MAP is agile, if it can be replaced by a slightly different version of it without compromising the usability of the entire platform. Minimising the number of agile points within MAP is the key for an easy *extensibility* of the platform. Extensibility is a key property for MAP, because of the issue of how to support a new tool. Assuming that the reports produced by such tool are not ASO-compliant, having a low number of agile points within the platform, minimises the number of changes to the platform itself that are needed in order to support the new tool.
In our case, for MAP to support a new tool, the following needs to be provided (either by the platform itself, or by the tool's developers):

- A `MAPWorkerAgent` that knows how to interact with the tool, and the corresponding `MAPTool` wrapper.
- A dedicated pseudo-grammar to guide the `MAPParsingAgent` in *Phase 2: translation* and the `MAPTranslatingAgent` *Phase 3: aggregation*.

It is noteworthy that the global structure of MAP, including the implementation of most of its agents would not need to change for a new tool to be supported.

4.8 Summary

In this Chapter we presented and discussed our multi-agent implementation of MAP. We detailed the rationale behind such choice, together with the agent model and architecture we made use of. We presented the agent-oriented high level view of MAP, followed by the agent-oriented implementation and discussion of its three sub-phases (*Phase 1: parallel analysis*, *Phase 2: translation*, and *Phase 3: aggregation*), including the agent protocols, and examples of each phase's products. We also showcased the implementation details of ASO. Finally, we discussed the agility and extensibility of our agent-oriented implementation of MAP.

Chapter 5

MagnetDroid Reasoning Module

In the previous Chapter we presented and discussed the implementation details of the *MagnetDroid Agent Platform* (MAP). In this Chapter we present and discuss the implementation details of the *MagnetDroid Reasoning Module* (MRM), and of the *MagnetDroid Web Application* (MWA) used to visualise its output, and to interact with MagnetDroid as a whole.

The goal of MRM is to support logical queries over the union of two knowledge bases. We start by explaining the process of creation of a *Technological Knowledge Base* (TKB) from a *final report* from MAP in Section 5.1, and we explain the process of creating of a *Legal Knowledge Base* (LKB) from a small subset of the 2018 UK DPA in Section 5.2. Then, we present our implementation of a Prolog framework that reasons on the union of TKB and LKB in Section 5.3. We also discuss our implementation of the *MagnetDroid Web Application* (MWA) as an entry point to MagnetDroid, and as a visualisation tool for the results of the queries that are supported by the Prolog framework in Section 5.4. We conclude the Chapter with a summary in Section 5.5.

Prolog knowledge bases Before we discuss how we derived and used the TKB, and the LKB, we need to establish a concrete language for both. Recall Section 3.2.2.2: there we stated that *normal logic programs* were our choice both for the representation of knowledge bases, and as a tool to enable logic reasoning. In our implementation of MRM we decided to use Prolog as the language in which both the TKB and the LKB are expressed. Likewise, we decided to use SWI-Prolog as the interpreter that supports queries on knowledge bases. We assume familiarity with the syntax and semantics of normal logic programs, as supported by Prolog. For details on Prolog, the reader is referred to [67].

5.1 Phase 4a: creation of a TKB

The goal of *Phase 4a: creation of a TKB* is to derive a logic program from a *final report* and a set of static logic rules regarding the security properties of *data confidentiality*, *data integrity*, and *data authenticity*. Despite being a full phase, it is numbered as 4a, because its dual phase (*Phase 4b: creation of a LKB*) is an asynchronous phase which needs to be completed before *Phase 6: reasoning*.

Recall the format specified by ASO in Section 3.3.2.2. In particular, ASO is a tree where the nodes represent security concepts, and the leaves represent the specific issues that are supposed to be subject of analysis by tools. The core components of each leaf are its state variables. The main component of a state variable is its value. In this section, we discuss how an ASO-compliant final report is translated into a set of Prolog predicates.

5.1.1 From leaves and state variables to Prolog rules

The issue of deriving part of a TKB from a final report can be rephrased as the issue of deriving a set of Prolog facts and rules from leaves and state variables. Any final report can be thought as the combination of fixed elements (i.e., the ASO structure from the root to the leaves), and mutable elements (i.e., the state variables). The translation into Prolog of the fixed elements preserves the ASO structure as a tree. For instance:

```
root(android_app, final_report).

child(<child_ID>, <parent_ID>, final_report).

leaf(<leaf_ID>, <parent_ID>, final_report).
```

The listing shows how to represent the tree structure in Prolog. The labels with angular brackets are just placeholders for constants. The root elements is represented by `root/2`, while each intermediate node is represented by a collection of `child/3` predicates. Finally, leaves are represented by a collection of `leaf/3` predicates. On the other hand, the mutable elements of a final report are represented by `state_variable/11` predicates, as illustrated in the following listing:

```
state_variable(<sv_ID>, <leaf_ID>, <value>, <history>,
               <ari>, <arc>, <miii>, <miic>, <mxii>, <mxic>, final_report).
```

Once again, the angular brackets represent generic constants, as opposed to variables in the above listing. Each state variable is represented by its ID, its leaf ID, its value, its history (if relevant), and a series of indexes and classes whose meaning is explained in Section 3.3.2.2. The following listing shows a simple example of the Prolog representation of a (doctored for space reasons) subset of a final report.

```
root(android_app, final_report).
...
child(network_activity, android_app, final_report).
...
child(insecure_protocols, network_activity, final_report).
...
leaf(plain_http, insecure_protocols, final_report).
...
state_variable(observed, plain_http, yes, [yes], 10, critical,
                6, average, 10, critical, final_report).
```

Essentially, the Prolog representation of this particular final report both preserves the tree structure, and conveys the relevant information that the state variable whose ID is `observed`, and whose leaf parent is `plain_http` contains. However, a Prolog representation of a final report is one of two building blocks of a TKB. The other one is a collection of Prolog rules that link the issues encapsulated by the state variables to the properties of *data confidentiality*, *data integrity*, and *data authenticity*, according to a mindset. As discussed in Section 3.3.3.1, MRM supports the *credulous* and *skeptical* mindsets. The following listing shows some of the rules that complement the Prolog representation of the final report:

```
data_confidentiality_issue(App, Evidence, plain_http, skeptical) :-
    app(App, Evidence),
    state_variable(observed, plain_http, yes, _, ARI, _, MIII, _, MXII, _, Evidence),
    MXII > 0

data_confidentiality_issue(App, Evidence, plain_http, credulous) :-
    app(App, Evidence),
    state_variable(observed, plain_http, yes, _, ARI, _, MIII, _, MXII, _, Evidence),
    MIII > 6

...

app(App, Evidence) :-
    state_variable(app_id, app_metadata, App, _, _, _, _, _, _, _, Evidence).
```

The listing shows how the same issue (an observed plain HTTP connection) is treated differently according to different mindsets. In *skeptical*, a data confidentiality issue is triggered as long as the *Maximal Impact Index* is greater than 0. In *credulous*, a data confidentiality issue is triggered only if the *Minimal Impact Index* is greater than 6. The Prolog representation of the final report, and the rules linking security issues to security properties and the mindset constitute our *Technological Knowledge Base* (TKB). Section 3.3.3.3 shows how a TKB in conjunction with the LKB (see Section 3.3.3.2) can be used to support Prolog-based queries, in order to determine legal violations and consequences.

5.1.2 The algorithm

Much like the sub-phases of MAP, *Phase 4a: generation of a TKB* can be described with the state diagram of Figure 5.1. As MRM is not part of an agent platform, we say that the algorithm is implemented by a *knowledge extractor*. The core of the algorithm is the loop that translates each node and state variable within the final report into a Prolog predicate. Afterwards, the fixed Prolog rules are appended, and the TKB is stored for future uses.

5.2 Phase 4b: creation of a LKB

The goal of *Phase 4b: creation of a LKB* is to derive a LKB from a subset of the 2018 UK DPA. As discussed in Section 3.3.3, this phase has three peculiar (with respect to every other phase) properties:

- It is asynchronous, i.e., it can be completed at any time before *Phase*

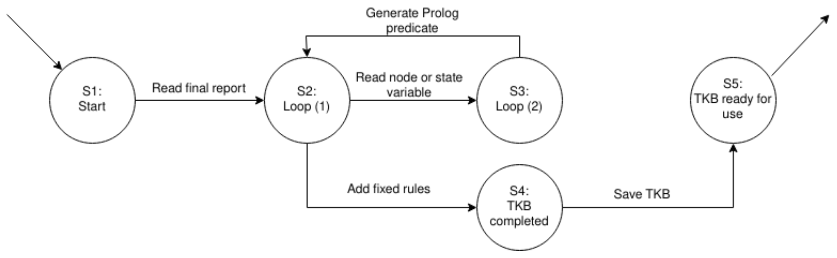

Figure 5.1: Phase 2a algorithm: A state diagram illustrating the conceptual states from the point of view of the knowledge extractor, together with the procedures that cause state transitions.

6: reasoning, and does not depend on the output of any other phase.

- It needs to be completed only once and for all, as, unlike the TKB, the LKB does not depend on any mutable entity (assuming the 2018 UK DPA is not amended).

- It is the only phase that is not automated. In fact, we manually translated the relevant articles (see Section 5.2.1) into Prolog rules.

5.2.1 The 2018 UK Data Protection Act

In this Section we examine the subset of the 2018 UK DPA that we translated into Prolog. Within the DPA our focus begins on Art. 66(1) and Art. 66(2) of Part 3. Recall from Section 3.3.3.2 that Art. 66 as a whole relates to *security of processing*, and that an Android application can be characterised as a *controller* and a *processor* (both of which are defined in Art. 32 of Part 3), or, more compactly as a *party*. In the next paragraphs we include and examine each of Art. 66(1) and Art. 66(2) (plus Art. 32 for the meaning of controller and processor), discussing our interpretation of them.

Relevant Articles of the 2018 UK DPA The relevant articles of the 2018 UK DPA [68] are as follows.

Art. 66(1) of Part 3

Each controller and each processor must implement appropriate technical and organisational measures to ensure a level of security appropriate to the risks arising from the processing of personal data.

Art. 66(2) of Part 3

In the case of automated processing, each controller and each processor must, following an evaluation of the risks, implement measures designed to –
(a) prevent unauthorised processing or unauthorised interference with the systems used in connection with it,
(b) ensure that it is possible to establish the precise details of any processing that takes place,
(c) ensure that any systems used in connection with the processing function properly and may, in the case of interruption, be restored, and
(d) ensure that stored personal data cannot be corrupted if a system used in connection with the processing malfunctions.

Art. 32 of Part 3

(1) In this Part, "controller" means the competent authority which, alone or jointly with others –

(a) determines the purposes and means of the processing of personal data, or

(b) is the controller by virtue of subsection (2).

(2) Where personal data is processed only –

(a) for purposes for which it is required by an enactment to be processed, and

(b) by means by which it is required by an enactment to be processed,

the competent authority on which the obligation to process the data is imposed by the enactment (or, if different, one of the enactments) is the controller.

(3) In this Part, "processor" means any person who processes personal data on behalf of the controller (other than a person who is an employee of the controller).

Appropriate security measures Art. 66(1) states that a party *must implement appropriate technical and organisational measures to ensure a level of security appropriate to the risks arising from the processing of personal data*. As discussed in Section 2.4.2, no further explanation of what constitutes *appropriate security measures* is given. In our interpretation, an Android application does not implement appropriate security measures if it exhibits *unmitigated risks* and/or *unmitigated threats*. In turn, if any of data confidentiality, data integrity, and data authenticity are not preserved, we imply the existence of unmitigated risks and threats. The aforementioned *data-X* properties are assumed preserved unless there is evidence that they are not. The evidence needs to be provided by a TKB.

Automated processing Art. 66(2) relates to the automated processing of data. Four separated sub-commas specify the obligations that a party engaged in automated processing must fulfil. However, we are not interested in some of them. Transparency (Art. 66(2b)), proper functioning (Art 66(2c)), and availability (Art. 66(2d)) are not interesting to us. We still model them, but the TKB will never provide any evidence that either of those properties is violated. As such, they will be assumed preserved. On the other hand, Art 66(2a) relates to *unauthorised processing and interference.* In our interpretation, for either to happen, at least one of the properties of data confidentiality, data integrity, and data authenticity need to be violated. As with Art. 66(1), we assume any property to be preserved, unless there is evidence of the contrary. Also, like with Art. 66(1), the evidence needs to be provided by a TKB.

5.2.2 From the articles to a Legal Knowledge Base

A Prolog model of the selected subset of the 2018 UK DPA needs to take into consideration the following elements:

- The numbered articles that may or may not be violated.
- The identifiable classes of problems that trigger legal violations, according to our interpretation of those articles:
    - Unmitigated risks.

  - Unmitigated threats.
  - Unauthorised processing.
  - Unauthorised interference.

- The security properties that need to be supplied by a TKB:
  - Data confidentiality.
  - Data integrity.
  - Data authenticity.

- The characterisation of an Android application as a controller and processor, or, in our interpretation, as a party.

The translation of those concepts needs to explicitly model the different ways in which different articles can be violated, at least in terms of our interpretation of the letter of the law. The resulting Prolog rules are discussed in the following paragraph.

Prolog Rules The first set of rules defines which parts of the law can be violated (*Article*), by what (*App*), how (*Reason*), according to which evidence (*Evidence*), and mindset (*Mindset*). In particular, the (*Evidence*) needs to be supplied by a TKB.

```
violates_law(App, Article, Evidence, FullReason, Mindset) :-
    violates_uk_dpa(App, Article, Evidence, Reason, Mindset),
    reverse(Reason, FullReason).

violates_uk_dpa(App, "Art._66(1)", Evidence, Reason, Mindset) :-
    violates_3_66_1(App, Evidence, Reason, Mindset).

violates_uk_dpa(App, Article, Evidence, Reason, Mindset) :-
    violates_3_66_2(App, Article, Evidence, Reason, Mindset).
```

The second set of rules specifies how the identified articles of the law can be violated, according to concepts that can still be found within the letter of the law. The *relevant_party/2* predicate is used to check whether it makes sense to investigate whether a specific party violates the law, according to the *Evidence* supplied by a TKB.

```
violates_3_66_1(App, Evidence, Reason, Mindset) :-
    relevant_party(App, Evidence),
    unmitigated_risk_found(App, Evidence, Reason, Mindset).

violates_3_66_1(App, Evidence, Reason, Mindset) :-
    relevant_party(App, Evidence),
    unmitigated_threat_found(App, Evidence, Reason, Mindset).

violates_3_66_2(App, "Art._66(2a)", Evidence, Reason, Mindset) :-
    in_scope("3_66_2a"),
    violates_3_66_2a(App, Evidence, Reason, Mindset).

...

violates_3_66_2a(App, Evidence, FullReason, Mindset) :-
    relevant_party(App, Evidence),
    unauthorised_processing(App, Evidence, Reason, Mindset),
    append(Reason, ["unauthorised_processing"], FullReason).

violates_3_66_2a(App, Evidence, FullReason, Mindset) :-
    relevant_party(App, Evidence),
    unauthorised_interference(App, Evidence, Reason, Mindset),
    append(Reason, ["unauthorised_interference"], FullReason).
```

The third set of rules further specifies how classes of problems that trigger legal violations can be triggered by means of not satisfying the properties of *data confidentiality*, *data integrity*, and *data authenticity*.

```
unauthorised_processing(App, Evidence, FullReason, Mindset) :-
    data_confidentiality_issue(App, Evidence, Reason, Mindset),
    append([Reason], ["data_confidentiality_issue"], FullReason).

...

unauthorised_interference(App, Evidence, FullReason, Mindset) :-
    data_authenticity_issue(App, Evidence, Reason, Mindset),
    append([Reason], ["data_authenticity_issue"], FullReason).
```

According to our interpretation of the law, the predicates relating to *data* *confidentiality*, *data integrity*, and *data authenticity* are not defined within the law itself. Therefore, they need to be supplied by a TKB. Likewise, all the *relevant_party/2* predicates need to be supplied by a TKB, as they refer to the name and relevance of a particular Android application.

5.3 Phase 5: reasoning

Once we have a TKB, and a LKB, we can reason on their union. Recall Section 3.3.3.2: the LKB makes use of certain security properties (*data confidentiality*, *data integrity*, and *data authenticity*) without providing evidence that shows their possible violation. Such evidence is always provided (if available) by the TKB. An example of a TKB complementing the LKB is available in Appendix D. If no evidence that a property has been violated is provided, it is assumed to be preserved. The act of integrating the evidence from the TKB completes the LKB, and enables logical reasoning via Prolog. We explain such queries in Section 5.3.1.

5.3.1 Queries

In Section 3.3.3.3 we discussed three main classes of queries: *purely technological*, *comprehensive legal*, and *selective legal*. In this Section we illustrate them in the context of a unified Prolog knowledge base deriving from a TKB and the LKB.

5.3.1.1 Purely Technological Queries

A MagnetDroid user may be interested in deriving the security issues that MAP has identified without having to parse the *final report*. For instance, they may be interested in every possible way in which a certain application does not preserve data confidentiality. The user is also rather concerned with the possibility that their private data could be seen by unauthorised individuals. Therefore, his mindset is *skeptical* (see Section 3.3.3.1). Given these premises, the user would formulate the following query [1]:

```
?- data_confidentiality_issue("app173", Evidence, Reason, "skeptical").
```

Essentially, what the user is interested in is whether the Android application whose ID is *app173* exhibits data confidentiality issues for some *Reason*, and according to some *Evidence*. A possible result of the query is the following:

```
Evidence = "final_report",
Reason = "plain_http_network_activity".
```

What the query result tells the user is that, according to the *final report*, *app173* exhibits a data confidentiality issue because it has engaged into network activity by means of a cleartext protocol (plain HTTP).

5.3.1.2 Comprehensive Legal Queries

Another MagnetDroid user may be interested in discovering all the possible ways in which a certain application violates the law. However, the user wants to give the benefit of the doubt to the application. Therefore, they assume a *credulous* mindset (see Section 3.3.3.1). Given these premises, the user would

[1] Actually, the user would interact with MagnetDroid via the MWA which formulates the queries automatically, depending on the user's choices. That is always true for all queries of all types.

formulate the following query:

```
?- violates_law("app173", Article, Evidence, Reason, "credulous").
```

Essentially, what the user is interested in is the set of all the *Article*s that the Android application whose ID is *app173* violates, together with all the identifiable *Reason*s, and according to some *Evidence*. A possible result of the query is the following:

```
Article = "Art. 66(1) of part 3 of the 2018 UK DPA",
Evidence = "final_report",
Reason = ["unmitigated_threat", "data_confidentiality_issue", "malicious_app"] ;

Article = "Art. 66(1) of part 3 of the 2018 UK DPA",
Evidence = "final_report",
Reason = ["unmitigated_threat", "data_confidentiality_issue", "malicious_app"] ;

Article = "Art. 66(2a) of part 3 of the 2018 UK DPA",
Evidence = "final_report",
Reason = ["unauthorised_processing", "data_confidentiality_issue", "malicious_app"] ;

Article = "Art. 66(2a) of part 3 of the 2018 UK DPA",
Evidence = "final_report",
Reason = ["unauthorised_processing", "data_confidentiality_issue", "malicious_app"] ;

Article = "Art. 66(2a) of part 3 of the 2018 UK DPA",
Evidence = "final_report",
Reason = ["unauthorised_interference", "data_confidentiality_issue", "malicious_app"] ;

Article = "Art. 66(2a) of part 3 of the 2018 UK DPA",
Evidence = "final_report",
Reason = ["unauthorised_interference", "data_confidentiality_issue", "malicious_app"] ;

false.
```

What the query result tells the user is that, according to the *final report*, *app173* violates both Art. 66(1), and Art. 66(2) for many different *Reason*s. In particular, limiting the explanation to the first result, Art. 66(1) was violated because the application exhibits an unmitigated threat due to a data confidentiality issue deriving from the fact that the application is a malware.

5.3.1.3 Selective Legal Queries

A third MagnetDroid user may be interested in whether or not a certain application violates the law. In particular, they do not care how, or which specific articles. However, the user is interested in whether a *credulous* approach would produce different results from a *skeptical* approach. Therefore, the user does not commit to any particular mindset, in order to bifurcate the results. Given these premises, the user would formulate the following query:

```
?- violates_law("app173", _, Evidence, _, Mindset).
```

Essentially, what the user is interested in is whether the Android application whose ID is *app173* violates any article of the law of any reason according to some *Evidence* and a certain *Mindset*. Note the use of _: in Prolog it represents a *don't-care variable*. A possible result of the query is the following:

```
Evidence = "final_report",
Mindset = "skeptical" ;

Evidence = "final_report",
Mindset = "credulous".
```

What the query result tells the user is that, according to the *final report*, *app173* violates the law according to both mindsets.

5.4 MWA: Interacting with MagnetDroid

In Section 5.3 we discussed how MRM enables and supports Prolog queries on the union of TKB and LKB. In Section 5.3.1 we showcased some of the possible queries, together with their results. The main issue with such system is that it requires specialised knowledge to formulate queries, and understand their results. The amount of specialised knowledge that is so far required in order to interact with MagnetDroid as a whole could be too much for most users. Therefore, we designed and built a *MagnetDroid Web Application* that allows average users to interact with the system in a human-friendly fashion.

5.4.1 Parameterised Analysis and Queries

The MWA is the entry point for MagnetDroid. In that regard, it allows for three main use cases:

- Upload an APK to analyse with a subset of the available tools (which are selectable), in order to produce a final report.
- Run a query on an APK for which a final report already exists. The query is automatically built by selecting its properties (e.g., the type, the legal articles to consider, etc.).
- A full run consisting of a sequential run of the two above use cases.

Additionally, a user can toggle their own type (default: regular user), in order to have the results presented in different fashions (see Section 5.4.2).

5.4.2 Visualisation of the Results

The other main purpose of the MWA is to visualise the results of the queries performed within MRM in a human-friendly fashion, and with an explanation that is tailored to each kind of user. Each run of MagnetDroid produces a result that is visualised as a row in a table. The table has three columns: *App Info*, *Security Issues*, and *Legal Violations*. Each cell contains a clickable link to the relevant information: the Android application info (i.e., metadata such as name, package name, etc.), the identified security issues (consisting of the final report, and the query result, if applicable), and the identified legal violations (consisting of the query result, if applicable). However, rather than presenting the raw result of a query, a textual interpretation of the *reason* behind the result is given. By toggling the user type, it is possible to switch between multiple interpretations which are tailored to the selected user type. Visualising a detailed screenshot of the aforementioned interpretation in a PDF like this thesis would not explain much. Therefore, in the following listing we present the three different interpretations of a violation of Art. 66(1) of the 2018 UK DPA (see Section 5.2.1) triggered by the analysed Android application using plan HTTP for its network activity.

```
# Developer
Art. 66(1) of Part 3 of the 2018 UK DPA has been violated:
- The application uses plain HTTP, which, being vulnerable
  to a Man-In-The-Middle (MITM) attack, is enough to
  compromise data confidentiality, data integrity, and data
  authenticity, according to the selected skeptical mindset.
  The probable cause is the use within the code of a
  HttpURLConnection with an "http://" URL.

# Legal Scholar
Art. 66(1) of Part 3 of the 2018 UK DPA has been violated:
- An unmitigated risk caused by the use of the insecure
  plain HTTP protocol has been identified. Any data sent
  via plain HTTP can be intercepted and modified by
  an attacker having access to the connection.

# Regular User
Art. 66(1) of Part 3 of the 2018 UK DPA has been violated:
- The use of plain HTTP makes so that any data sent
  through the Internet can be intercepted and modified by
  an attacker having access to the connection.
```

Thanks to the visualisation of different explanations we can present the output of queries in a way that is friendly to different kinds of users.

5.5 Summary

We have presented the most important aspects of our implementation of MRM, and of the visualisation of its results (via MWA). We explained a knowledge extractor that derives a Prolog TKB from a MAP final report, and a set of fixed rules. We then discussed how we manually derived a LKB from a subset of the 2018 UK DPA, and explained how Prolog can be employed to support different kinds of queries on the union of TKB and LKB. Finally, we showed how we use the MWA to present the results of the aforementioned queries to different kinds of users.

Chapter 6

Discussion

In the last three Chapters we presented MagnetDroid. In this Chapter we investigate the goodness of our approach and implementation of the system. We structure this Chapter as follows. In Section 6.1 we investigate whether the high level approach described in Chapter 3 is suitable towards the goals we stated in Section 3.2. Section 6.2 investigates whether the ASO is a reasonable model for the security issues that can impact Android applications. Section 6.3 explores whether MAP provides a workable environment for integrative analysis, and Section 6.4 investigates whether the relation between the security issues identified by MAP, and the legal violations identified by MRM.

We can anticipate that some of the above questions required an experimental evaluation for us to determine their goodness. However, for some of the others, we were able to provide justifications based on widely accepted standards and best practices. We want to stress that we envision MagnetDroid as a foundation for future work on the subject. Therefore, we do not claim that it is able to solve all the issues that the stat-of-the-art has not addressed yet.

6.1 Justification of the High Level Approach

In this Section we investigate whether the high level approach described in Chapter 3 is suitable towards the goals stated in Section 3.2. In Section 2.4 we identified a disconnect between the lacking legal response to privacy and the technological analysis that aims at identifying security issues that impact privacy. Lacking refers to the use by the law of technical terms that are not further defined, and left vague. We take the mandatory interpretation process of the relevant law in court cases as hard evidence of such vagueness. The core issue is clear: the languages of security analysis and law prevent any standardisable cooperation between the two domains. While interpretation is not disappearing anytime soon (nor we claim it should), it is undeniable in our opinion that it would significantly benefit from factual evidence coming from a standardised technological analysis. Additionally, such evidence, being the fruit of analysis, can be known in advance before a case is even brought to a court. That is advantageous, because it offers protections to regular users (e.g., “should I use this application”), and because it offers developers and publishers an early detection technique to prevent legal issues stemming from negligence (e.g., inherent vulnerabilities in published applications). Therefore, we claim that our approach provides an analysis based interpretation to the legal domain, which can only be useful, given the limitations of the current state-of-the-art. That is, of course, if the result of our analysis is not grossly misleading. Debunking that possibility is the purpose of our experimental evaluation of MagnetDroid (see Section 6.3 and Section 6.4).

6.2 The significance of ASO

Validating our Android Security Ontology is a necessary step towards validating MagnetDroid as a whole. Recall that ASO is an input to MAP. It is widely recognised that inadequate inputs provide a straightforward path towards meaningless outputs, according to the *garbage in - garbage out* (GIGO) [69] principle. Even worse, no amount of excellent internal processing can subvert GIGO. The main declared purposes of ASO (see Section 3.3.2.2) are to encompass the relevant security issues that an Android application can face at the application level (i.e., everything the application has the control of), and to provide a standard format (syntax and semantics) to express them. However, there is more to it: as detailed in Section 3.3.2.2, each state variable representing a single issue has 3 indexes (plus some corresponding classes for syntactic sugar). We discuss these indexes in turn.

6.2.1 Absolute Relevance Index

The Absolute Relevance Index (ARI) aims to assign a higher value to the more serious security problems, and lower values to less serious security problems. More specifically, an ARI value only refers to the seriousness of the issue per se, irrespective of the analysis. As such, the ARI for a specific state variable is immutable, and not dependent on the analysis. For instance, the state variable representing the application reputation (e.g., benign, malicious, suspicious, ...) is fixed to 10, because a malicious application represents on of the greatest security problems, should it happen. Essentially, the purpose of ARI is to provide a (partial) ordering of the security issues. It should surprise nobody that we regard an untruthful or completely arbitrary ARI as *garbage input*. Therefore, we need to justify how we assigned a value to each state variable ARI. As we discussed in Section 2.2 and Section 3.3.2.2, we sourced the Android threat model [70], and other [71] [72] Transport Security and WebSec related standards, in order to build ASO. Many of the sources already contain an implicit characterisation of the issues they describe in terms of severity. Our job was to infer concrete numbers from them. We decided against including the outcome of real world court cases in our process to determine the ARIs for two main reasons. First, the law body that we incorporated (2018 UK DPA) is relatively recent, and, therefore, not many data are available. Second, and, more important, the outcome court cases are not always proportionate to the offenses. We believe that interpretation (always occurring in any court case) and factors such as punitive damages (in civil cases) introduce a bias leading to more importance being assigned to the security issues that are *perceived* to be more important by judges and common people, rather than to those issues that *are* more important, as recognised by security professionals.

6.2.2 The Impact Indexes

The other indexes (Minimal Impact Index - MIII and Maximal Impact Index - MXII) represent how serious a certain security issue that has been identified by the analysis is. Their existence support the twofold mindset (*skeptical* vs *credulous*) described in Section 3.3.3.1. In this section, we provide a justification for our choice of a twofold mindset, and a justification of the values we assigned to the MIII and MXII of each state variable (much like ARI).

We start by recognising that one can make a reasonable case that the same vulnerability may have a different impact, depending on the context. For instance, transmitting anonymised diagnostics data from an application to a server in cleartext, as deplorable as it is, has arguably a lower impact than sending a password in cleartext to a server. The keyword is *arguably*. One can also make the case that cleartext communications *must not* be performed at any time (except perhaps when it is unavoidable due to terrible constructs such as WIFI captive portals which can't be helped by the developer), because they are inherently dangerous. In the interest of providing results that reflect both points of view to the users of MagnetDroid, we included a MIII and a MXII for each state variable. That way, it is possible to bifurcate the reasoning (see Section 5.3).

6.2.2.1 Values for MIII and MXII

Having made the case for the existence of MIII and MXII within each ASO state variable, we now need to justify how we assigned their values. We based our selection on the same sources that we used to build ASO in the first place. The difference with ARI is that MXII and MIII are context dependent, as explained in the previous section. Therefore, their value is tailored to the actual issues found by the analysis and contained in the translated reports. As such, the values of MIII and MXII are determined by `MAPTranslatingAgents` for translated reports, and by the `MAPAggregatingAgent` for the final report. Finally, as the names suggest, MXII is axiomatically greater than or equal to MIII under all circumstances.

To summarise our take on ASO, we believe that it accurately represents the most significant security issues that an Android application can face, together with some metadata that are useful and used while reasoning (see Chapter 5). We support our claim by referencing established threat models and international standards. Therefore, in the next sections of our evaluation of MagnetDroid, we will assume that ASO does not constitute an instance of *Garbage Input*.

6.2.3 The Nature of DNS Issues

In our ASO we regard plain DNS queries that are used to resolve domain names needed by an Android application in order to initiate some kind of network activity as application security issues. It is known that, by default, DNS resolution is managed by the underlying OS. Therefore, it appears that plain DNS queries should be classified as a platform issue, and thus ignored by MagnetDroid as per Section 2.1.4. However, there is a catch. While it is true that DNS resolution is an OS matter *by default*, it is possible for Android applications to resolve DNS names without involving the OS, by means of DNS-over-HTTPS (DoH) [73]. DoH (not to be confused with DNS-over-TLS [74] which simply wraps OS-initiated DNS activity with the TLS protocol) is a protocol that allows any kind of application software (not necessarily Android or even mobile applications) to perform DNS queries to a resolver (that also supports DoH) wrapping the request within HTTP which, in turn, is wrapped with TLS (as per textbook HTTPS). The important points are that every message to and from the DNS resolver is seen as regular HTTPS traffic by an external observer (e.g., a MITM), and the OS is not involved in crafting any DNS query and/or interpreting its results.

For Android applications the flow is as follows. Normally, whenever a domain name is given in input to a (SSL)Socket or an HTTP(S)URLConnection, the application delegates the resolution of such name to the OS. Therefore, it has no control over how that kind of DNS requests are made. However, it is not mandatory for the hostname to be a domain name when opening a connection to a remote endpoint. In fact, both (SSL)Socket and HTTP(S)URLConnection work well with IP addresses as hostnames. However, an HTTPSURLConnection still needs a proper domain name, but that can be specified in the *Server Name Indication* (SNI) [75] TLS extension in the *Client Hello* or *Encrypted Client Hello* (the latter still in its infancy). Notably, SNI does not trigger any DNS request.

The core point is that an application can rely on DoH to resolve a DNS name, and then use the IP address as the hostname, therefore avoiding OS-initiated DNS queries. As such, since there is an effective mitigation at the application level, the plain DNS issue can be considered an application issue.

To prove the point, we built a library that allows developers to initiate HTTPS connections (or connections with custom protocols over an SSLSocket) that automatically use DNS-over-HTTPS to resolve the domain name with one line of code. The library is available at `https://git.io/JDOTq`.

6.3 Experimental Evaluation: MAP

The goal of this section is to investigate whether MAP is a suitable platform for integrative analysis. We believe such goal requires an experimental evaluation of the flow of the MAP phases. As MAP is only a sub-component of MagnetDroid, we limited the scale of our experiments. The full-scale experiments on the full flow of $MAP + MRM$ described in Section 6.4 complement the small-scale experiments described in this Section.

Setup Our setup for the small-scale experimental evaluation consists of two Virtual Machines (VMs) with 8 GB of RAM each in which an independent version of MAP is booted. We enabled all the tools described in Section 4.3.1.3.

Samples Our sample set consists of 100 APKs mainly obtained from the AndroZoo [76] dataset. We only selected applications whose year of publication of their analysed version is at least 2018. We included a known malware (whose package name is *com.qrcodescanner.barcodescanner*).

Results Of the 100 analysed applications, 7 caused a crash in MalloDroid, while the other tools were able to produce a raw report for all of them. 80 out of 100 of the analysed applications did not exhibit any ASO-relevant issue in the final report. Among those that did exhibit ASO-relevant security issues, 19 of them performed at least a plain DNS connection (without DNSSEC), and initiated at least a plain HTTP connection to a server at least once. 10 out of 100 requested at least one permission that was not needed. 1 out

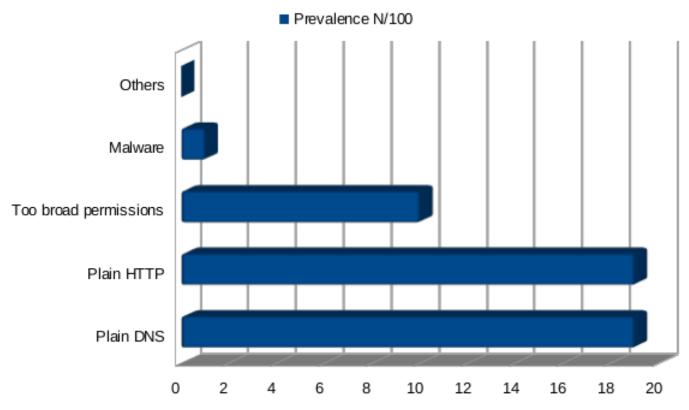

Figure 6.1: Prevalence of the security issues: The barplot shows the security issues identified by the integrative analysis. The X axis indicates the number of applications for which a certain issue (Y axis) was reported.

of 100 was classified as a malware (the one application where we had prior knowledge of its maliciousness). All applications that exhibited encrypted network communications by means of TLS 1.2+ correctly aborted the TLS handshake when presented with an injected *SERVER_HELLO* message selecting TLS ciphersuites marked as insecure by the Qualys SSL Labs threat model [71]. No conflicts were recorded during any *Phase 3: aggregation*. Figure 6.1 illustrates the prevalence of the identified issues.

Comment on the results Despite the strong incentive to encrypt all network communications with sound protocols (such as TLS 1.3, or the Signal protocol), there are still some bubbles of resistance in terms of the use of trivially insecure protocols, such as plain HTTP for network communications. Our opinion is that the issue derives by the failure of the Android API to present HTTPS connections as an opinionated default, as opposed to the (unfortunately) much easier to use, but insecure `HTTPURLConnection` with an *http://...* URL. Additionally, despite the current push from companies such as Google, Cloudflare, and Mozilla towards a more secure DNS model (via DNS-over-TLS and/or DNS-over-HTTPS), the experiments show that old plain DNS queries (even without DNSSEC) are still broadly used. Moreover, this first batch of experiments seems to confirm that a non-negligible number of Android applications ask for too broad permissions. Finally, MAP successfully included in the final report of the analysis of the only known malware its status as a malicious application.

6.4 Experimental Evaluation: MRM

In this section we investigate whether MAP and MRM constitute an appropriate bridge between security-oriented Android application analysis, and the law. In that regard, we believe a large-scale experimental evaluation of the combined flow of MAP and MRM is required.

Setup Once again, our setup for the small-scale experimental evaluation consists of two Virtual Machines (VMs) with 8 GB of RAM each in which an independent version of MAP is booted. We enabled all the tools described in Section 4.3.1.3. In terms of queries, we opted for the comprehensive legal queries, as they maximise the testing coverage of MRM.

Samples Our sample set consists of 10,000 APKs mainly obtained from the AndroZoo [76] dataset. We only selected applications whose year of publication of their analysed version is at least 2018. We included 10 known malicious applications.

Results Of the 10,000 analysed applications, 217 could not be run within an Android Virtual Device (because of crashes). Therefore, among the tools, Bettercap, CSP-checker, and OCSP-checker could not produce any raw report. Of the 10,000 analysed applications, 1,205 cased MalloDroid to crash, and, as such, the tool could not produce a raw report for them. The 9793 Android applications for which at least a raw report was produced collectively triggered 6,741 security issues. Out of 9793, 7034 applications (roughly 72%) did not trigger any detectable ASO-relevant issue. As such, we conclude that either they do not violate the law, or we were not able to demonstrate that they do. Given the number of identified security issues, the remaining 2795 applications triggered on average 2.44 issues each. Most of the issues (roughly 60%) regard either the use of plain DNS (34%), or the use of plain HTTP (26%). 10% of the identified issues relates to Android applications requesting at least one permission that was not needed. Roughly 12% of the remaining issues relates to insecure TLS (either obsolete versions of the protocol, or insecure parameters such as ciphersuites). The remaining issues refer to other ASO-relevant concepts such as maliciousness (21 instances), etc. No conflicts were recorded during any *Phase 3: aggregation*. Figure 6.2 illustrates the partition of the identified issues.

Legal Findings According to the comprehensive legal queries that look for all the possible links between ASO-relevant security issues and legal violations, under the *skeptical* mindset, all 6,741 security issues triggered the violation of both Art. 66(1) and Art. 66(2), for a total of 28312 different

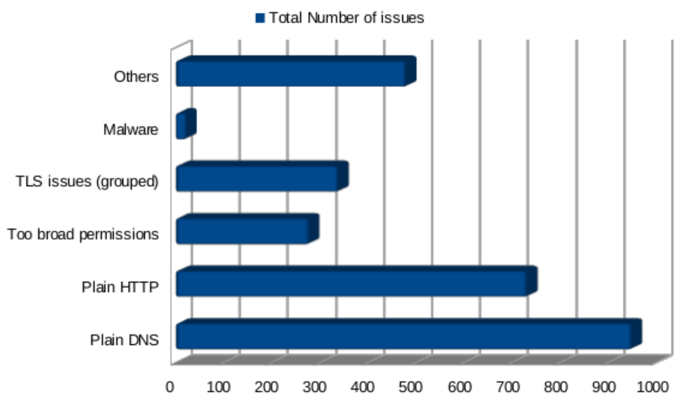

Figure 6.2: Prevalence of the security issues: The barplot shows for the top categories of issues, how many times they were reported by the integrative analysis. The X axis indicates the number of applications for which a certain issue (Y axis) was reported.

violations (roughly 4.2 per issue). In that scenario, 100% of the applications that exhibited at least a security issue triggered at least a legal violation. Conversely, under the *credulous* mindset, the 6,741 security issues triggered only 2649 legal violations, roughly 0.39 per issue. In that scenario, only 63% of the applications that exhibited at least a security issue triggered at least a legal violation. Figure 6.3 illustrates the relation between identified issues, and legal violations under the *skeptical* and *credulous* mindsets respectively.

Comment on the results With regard to the identified security issues, the comments we made in Section 6.3 still apply. With regard to the relation between the identified security issues, and the identified legal violations, it is not surprising that the *skeptical* mindset revealed a strong correlation (i.e., every single security issue triggered at least a legal violation), while the *credulous* mindset revealed a much weaker correlation (i.e., each security

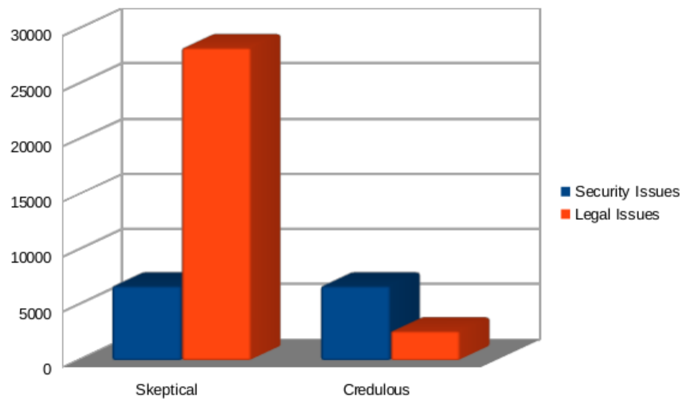

Figure 6.3: Relation between security issues and legal violations: The barplot shows for both the skeptical and the credulous mindsets, how many legal violations where triggered by the identified security issues.

issue triggered on average 0.39 legal violations).

6.5 Summary

We evaluated our thesis' contributions with respect to the goals of the work. We discussed how reasonable the high level approach, ASO, MAP, the model of the law, and MRM are. When possible, we relied on existing standards, and industry best practices to provide answers to our enquiries. When not possible, we relied on two separate experimental settings that informed our evaluation. The obtained results suggest that MagnetDroid is indeed a step-up from the state-of-the-art for developers, users, and legal experts, and solid ground for future work on the subject. Indeed, the advantages of an integrative approach open up new directions for application analysis. These advantages and the limitations of our work are discussed in the next Chapter that completes the work of this thesis.

Chapter 7

Conclusions and Future Work

7.1 Summary of the Thesis

In this thesis we have presented MagnetDroid, a bridge between security, privacy, and the law for Android applications.
We started our work by analysing the concept of privacy from different points of view: philosophical, legal, and technological. We established privacy of data as our subject of interest, and regarded security as a necessary condition for it. We focused, in particular, on the problem of data privacy in the context of Android applications, explicitly disregarding servers and non-Android software. In that regard, we discussed how the legal and technological states-of-the-art do not manage privacy-impacting security issues effectively. In particular, we identified the lack of a cooperative approach between technology and the law, due to a semantic disconnect. Additionally, we identified the lack of systematic cooperative approach between different security-oriented Android application analysis tools. As a result, we made

our primary goal to bridge the gap between security-oriented Android application analysis and the law, so that it becomes possible to identify legal issues related to data privacy from security issues related to Android applications. To experiment with these ideas, we selected a subset of the 2018 UK DPA as the representative for the law on data protection.

We then proposed a novel approach to bridge the gap between Android application analysis and the law by designing MagnetDroid. First, this system uses a novel Android Security Ontology (ASO) to express the common security issues in the context of Android applications in a standard format. Second, we designed the MagnetDroid Agent Platform (MAP) that, given an APK and a set of analysis tools, runs the tools in parallel on the APK, and collects, translates, and aggregates the reports under ASO, resulting in a technological final report. Third, we designed a MagnetDroid Reasoning Module (MRM) which derives a Technological Knowledge Base (TKB) from each final report and some fixed rules regarding security. We unified each TKB with a hand-compiled Legal Knowledge Base (LKB), which we derived from a subset of the 2018 UK DPA. We also identified normal logic programs as a way to support logical reasoning and queries on the union of the knowledge bases, so that it is possible to determine, under our interpretation, which identified security issues trigger which legal consequences. Finally, we designed a MagnetDroid Web Application as an interface to the system, and to present the query results to different kinds of users: developers, legal experts, and regular users.

We implemented MAP as a multi-agent platform with the goal of minimising agility in order to maximise extensibility (i.e., future support for new analysis tools). In doing so, we took advantage of the innate properties of semi-autonomous agents (such as communication, coordination, and cooperation), as well as the algorithmic nature of the sub-requirements of our main goal. We also implemented MRM by selecting Prolog as the language for TKB, LKB, and queries. We implemented MWA as a simple web application that presents to different kinds of users a customised explanations of the query results.

7.2 Future work

We envision MagnetDroid as the foundation layer for a new cooperative approach to data privacy between technology and the law. The current proof-of-concept implementation has limitations that open the possibility of future research directions.

- At present, only 5 tools are supported by MAP. More can be added by virtue of the easy extensibility of the platform, as explained in Section 4.7. Additionally, if a tool produces a report that is already ASO-compatible, it would be possible to completely bypass *Phase 2: translation* for that particular report.

- Currently, MagnetDroid is limited by the believability of the analysis tools upstream. The conflict management mechanism is able to detect (and remove from further use) some incorrect results at the expense of removing a likely correct result as well. Additionally, if an unsolvable conflict is found is *Phase 3: aggregation* of MAP, it is not possible to determine whether the security issue represented by the state variable whose value cannot be determined (due to the conflict) is real or not. Therefore, such security issue is ignored by MRM. A possible extension to MAP could assign a value of trustworthiness to each tool, and use it, together with a voting system, to solve at least some of the currently unsolvable conflicts.

- Right now, only a small subset of the 2018 UK DPA is translated into a LKB. Additionally, the compilation of the LKB is not automated. One interesting direction in the future would be to employ natural language processing in order to automate the process, as well as incorporating more legal rules and regulations.

- As of now, human users have not been involved in testing the visualisation of the MRM results via MWA. A feedback from real users could be employed to further tailor the experience to the needs of developers, legal experts, and regular Android application users.

Additionally, while MagnetDroid was designed with a limited technological perspective in mind (i.e., the focus on Android applications), many of its ideas can be recycled to build a bridge between any kind of software application and the relevant law on data protection. A future research direction could be to design security ontologies for different kinds of software applications, and to implement an agent environment in which arbitrary software applications can be analysed by the appropriate tools.

While all of the above was out-of-scope for this thesis, we believe it could stimulate researchers to raise to the challenges, and propose novel and interesting extensions to MagnetDroid.

Appendix A: MAP agents actions

This appendix contains the concrete action that the agents described in Section 4.3.1.2 can attempt within the `MAPEnvironment`.

MAPCoordinatorAgent A `MAPCoordinatorAgent` can attempt the following actions:

- `MAPStartPhase1Action` which, given the metadata of a set of available analysis tools, and an APK, forwards them to an appropriate set of `MAPWorkerAgent`s, officially starting *Phase 1: parallel analysis.*
- `MAPStopPhase1Action` which, upon receiving all the raw reports (or error signals) from the `MAPWorkerAgent`s, stores them for the next phase, officially ending *Phase 1: parallel analysis.*
- `MAPStartPhase2Action` which, given the raw reports, sends them to the appropriate `MAPParsingAgent`s, officially starting *Phase 2: translation.*

- `MAPStopPhase2Action` which, upon receiving all the translated reports from the `MAPTranslatingAgent`s, stores them for the next phase, officially ending *Phase 2: translation*.

- `MAPStartPhase3Action` which, given the translated reports (ASO instantiations), sends them to a `MAPAggregatingAgent`, officially starting *Phase 3: aggregation*.

- `MAPStopPhase3Action` which, upon receiving the final report from the `MAPAggregatingAgent`, stores it for future use, officially starting *Phase 3: aggregation*, and the MAP flow with it.

As the actions suggest, the `MAPCoordinatorAgent` is essentially the supervisor that links together the other agents of MAP, and propagates the intermediate products of each phase to the successive phase.

MAPWorkerAgent A `MAPWorkerAgent` can attempt the following actions:

- `MAPStartAnalysisAction` which, given the paths of an APK and of an Android application analysis tool, starts the analysis of such tool on the APK, by interacting with the appropriate `MAPBody`.

- `MAPInputAction` which, provides an input to a running tool (e.g., an injectable event into an Android emulator UI) by interacting with the appropriate `MAPBody`.

- `MAPStopAnalysisAction` which stops the analysis of a running tool (unless it is already over) by interacting with the appropriate `MAPBody`.

- `MAPRetrieveRawReportAction` which retrieves the raw report generated by the tool (if available) by interacting with the appropriate `MAPBody`.

- `MAPDataTransferAction` which can be used to send data to another `MAPAgent`, enabling, therefore, an effective form of agent communication. A `MAPWorkerAgent` uses it to send the raw report (or an error signal) to the `MAPCoordinatorAgent`.

MAPParsingAgent A `MAPParsingAgent` can attempt the following actions:

- `MAPIdentifyReportAction` which, given a raw report, identifies its nature as either a narrative, or a collection of facts.

- `MAPRemoveUselessDataAction` which, given a raw report (or its remnant), edits it by removing all the ASO-irrelevant information that can be identified according to the pseudo-grammar of the report that the `MAPParsingAgent` possesses.

- `MAPExtractUsefulDataAction` which, given a raw report (or its remnant), edits it by extracting all the ASO-relevant information that can be identified according to the pseudo-grammar of the report that the `MAPParsingAgent` possesses. The extracted information is stored in the working memory of the `MAPParsingAgent`.

- `MAPProduceIntermediateReportAction` which, given the stored ASO-relevant information, and the raw report nature, produces either a *narrative report*, or a *factual report*.

- `MAPDataTransferAction` which can be used to send data to another `MAPAgent`, enabling, therefore, an effective form of agent communication. A `MAPParsingAgent` uses it to send the produced intermediate report (either narrative, or factual) to the right `MAPTranslatingAgent`.

MAPTranslatingAgent A `MAPTranslatingAgent` can attempt the following actions:

- `MAPIdentifyStateVariable` which, given an intermediate report (either narrative, or factual), and ASO, identifies a state variable that can be instantiated from the content of the intermediate report.
- `MAPInstantiateStateVariable` which given an intermediate report and a state variable, instantiates the state variable from the content of the report. The instantiated state variable is stored in the working memory of the `MAPTranslatingAgent`.
- `MAPProduceTranslatedReportAction` which, given the stored instantiated state variables and ASO, produces a *translated report*.
- `MAPDataTransferAction` which can be used to send data to another `MAPAgent`, enabling, therefore, an effective form of agent communication. A `MAPTranslatingAgent` uses it to send the produced translated report to the `MAPCoordinatorAgent`.

MAPAggregatingAgent A `MAPAggregatingAgent` can attempt the following actions:

- `MAPCheckForConflict` which, given a set of *translated reports*, checks whether, for a specific state variable, a conflict arises among at least two translated reports with respect to the value of such state variable.

- `MAPCopyStateVariable` which copies the value of a state variable (or the derived value from a `MAPConflictSolverAction`, or a *Conflicting* value) to the work-in-progress final report in the internal memory of the `MAPAggregatingAgent`.

- `MAPConflictSolverAction` which attempts to derive an acceptable value for a state variable of the final report when a conflict arises between at least two translated reports with respect to the value of such state variable. If no acceptable value can be derived, a *Conflicting* value is produced instead.

- `MAPProduceFinalReport` which, given the in-memory stored information regarding the ASO state variables and ASO itself, produces the *final report*.

- `MAPDataTransferAction` which can be used to send data to another `MAPAgent`, enabling, therefore, an effective form of agent communication. A `MAPAggregatingAgent` uses it to send the produced final report to the `MAPCoordinatorAgent`.

Appendix B: Raw Report Example

The following is a raw report generated by the Bettercap tool (see Section 4.3.1.4).

```
[14:10:14] [sys.log] [inf]
    Loading proxy script caplets/http-req-dump.js ...
[14:10:14] [sys.log] [inf]
    Loading proxy certification authority TLS key from
        /root/.bettercap-ca.key.pem
[14:10:14] [sys.log] [inf]
    Loading proxy certification authority TLS certificate from
        /root/.bettercap-ca.cert.pem
[14:10:14] [sys.log] [inf]
    Loading proxy script caplets/http-req-dump.js ...
[14:10:14] [sys.log] [inf]
    http.proxy started on 192.168.1.3:8080 (sslstrip disabled)
[14:10:14] [sys.log] [inf]
    https.proxy started on 192.168.1.3:8083 (sslstrip disabled)
```

```
192.168.1.117/24 > 192.168.1.3  >> [14:10:14] [sys.log] [inf]
    You are running 2.7 which is the latest stable version.
192.168.1.117/24 > 192.168.1.3  >> [14:10:21] [net.sniff.leak.http]
    http local POST example.com Mozilla/5.0 (X11; Android arm; rv:78.0)
        Gecko/20100101 Firefox/78.0

  Method: POST
  URL: /
  Headers:
    Host: example.com
    User-Agent: Mozilla/5.0 (X11; Android arm; rv:78.0)
        Gecko/20100101 Firefox/78.0
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
    Accept-Language: en-US,en;q=0.5
    Accept-Encoding: gzip, deflate
    DNT: 1
    Connection: keep-alive
    Upgrade-Insecure-Requests: 1
    Cache-Control: max-age=0

  Form:
    mgtxt => This message will be intercepted.
    sendbtn =>   Send
    nmtxt => U. N. Owen
    action => send
```

```
192.168.1.117/24 > 192.168.1.3  >> [14:11:32] [net.sniff.leak.http]
    http local POST example.com Mozilla/5.0 (X11; Android arm; rv:78.0)
        Gecko/20100101 Firefox/78.0

 Method: POST
 URL: /login
 Headers:
   Host: example.com
    User-Agent: Mozilla/5.0 (X11; Android arm; rv:78.0)
        Gecko/20100101 Firefox/78.0
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
    Accept-Language: en-US,en;q=0.5
    Accept-Encoding: gzip, deflate
    DNT: 1
    Connection: keep-alive
    Upgrade-Insecure-Requests: 1
    Cache-Control: max-age=0

 Form:
   password => ifeellikeimbeingwatched
   submit =>
   username => cloudstrife9999
```

Appendix C: Pseudo-grammar Example

The following is an example of pseudo-grammar for the tool Bettercap.

```
{
    "expected_report_type": "narrative",
    "narrative_watchdogs": [
        "^\[[0-9]{2}\:[0-9]{2}\:[0-9]{2}\]"
    ],
    "known_uninteresting_patterns": [
        "You are running .+",
        "^\[[0-9]{2}\:[0-9]{2}\:[0-9]{2}\]",
        "\[sys\.log\]",
        "[stable|beta|alpha|development|dev] version",
        "^\s*[W|w]elcome to bettercap \!?",
        "Wifi access point",
        "arp spoofer started",
        "http.proxy started",
        "https.proxy started",
        "dns.spoof .+ \-> .+",
        "targets: .+",
        "Loading proxy .+",

    ],
    "potentially_interesting_patterns": [
        "sending spoofed DNS",
        "sending spoofed ARP",
        "Found strippted HTTPS link",
        "Stripping \d+ HTTPS lines",
        "net.sniff.leak.http"
    ],
    "interesting_patterns": [
        "http local POST",
        "http local GET",
        "http local HEAD",
        "http local PUT",
        "http local TRACE",
        "http local OPTIONS",
        "http local DELETE",
        "https local POST",
        "https local GET",
        "https local HEAD",
        "https local PUT",
        "https local TRACE",
        "https local OPTIONS",
        "https local DELETE",
        "Received request for .+"
    ]
}
```

Appendix D: Example of a TKB complementing a LKB

The following is an example of how a TKB complements a LKB by providing the definition of some fluents that are undefined in the LKB. Note that *data_confidentiality_issue*, *data_integrity_issue*, and *data_authenticity_issue* are undefined in the LKB, and provided by the TKB.

```
%%%%%%%%%%%%%%%%%%%%%%%%
% Legal Knowledge Base %
%%%%%%%%%%%%%%%%%%%%%%%%

:- [tkb]. % Importing the TKB.

violates_law(App, Article, Evidence, FullReason, Mindset) :-
    violates_uk_dpa(App, Article, Evidence, Reason, Mindset),
    reverse(Reason, FullReason).

violates_uk_dpa(App, "Art._66(1)_of_part_3_of_the_2018_UK_DPA", Evidence, Reason, Mindset) :-
    in_scope("3_66_1"),
    violates_3_66_1(App, Evidence, Reason, Mindset).

violates_uk_dpa(App, Article, Evidence, Reason, Mindset) :-
    violates_3_66_2(App, Article, Evidence, Reason, Mindset).

violates_3_66_1(App, Evidence, Reason, Mindset) :-
    relevant_party(App, Evidence),
    unmitigated_risk_found(App, Evidence, Reason, Mindset).

violates_3_66_1(App, Evidence, Reason, Mindset) :-
    relevant_party(App, Evidence),
    unmitigated_threat_found(App, Evidence, Reason, Mindset).
```

```
relevant_party(Party, Evidence) :-
    controller(Party, Evidence).

relevant_party(Party, Evidence) :-
    processor(Party, Evidence).

controller(Party, Evidence) :-
    app(Party, Evidence).

processor(Party, Evidence) :-
    app(Party, Evidence).

unmitigated_risk_found(App, Evidence, FullReason, Mindset) :-
    risk(Risk, Evidence),
    unmitigated(App, Risk, Evidence, Mindset, Reason),
    append(Reason, ["unmitigated_risk"], FullReason).

unmitigated_threat_found(App, Evidence, FullReason, Mindset) :-
    threat(Threat, Evidence),
    unmitigated(App, Threat, Evidence, Mindset, Reason),
    append(Reason, ["unmitigated_threat"], FullReason).

unmitigated(App, RiskOrThreat, Evidence, Mindset, Reason) :-
    inappropriate_security(App, RiskOrThreat, Evidence, Mindset, Reason).

inappropriate_security(App, RiskOrThreat, Evidence, Mindset, Reason) :-
    data_confidentiality_issue(App, Evidence, RiskOrThreat, Mindset),
    append([RiskOrThreat], ["data_confidentiality_issue"], Reason).

inappropriate_security(App, RiskOrThreat, Evidence, Mindset, Reason) :-
    data_integrity_issue(App, Evidence, RiskOrThreat, Mindset),
    append([RiskOrThreat], ["data_integrity_issue"], Reason).

inappropriate_security(App, RiskOrThreat, Evidence, Mindset, Reason) :-
    data_authenticity_issue(App, Evidence, RiskOrThreat, Mindset),
    append([RiskOrThreat], ["data_authenticity_issue"], Reason).

violates_3_66_2(App, "Art._66(2a)_of_part_3_of_the_2018_UK_DPA", Evidence, Reason, Mindset) :-
    in_scope("3_66_2a"),
    violates_3_66_2a(App, Evidence, Reason, Mindset).

violates_3_66_2(App, "Art._66(2b)_of_part_3_of_the_2018_UK_DPA", Evidence, Reason, Mindset) :-
    in_scope("3_66_2b"),
    violates_3_66_2b(App, Evidence, Reason, Mindset).

violates_3_66_2(App, "Art._66(2c)_of_part_3_of_the_2018_UK_DPA", Evidence, Reason, Mindset) :-
    in_scope("3_66_2c"),
    violates_3_66_2c(App, Evidence, Reason, Mindset).

violates_3_66_2(App, "Art._66(2d)_of_part_3_of_the_2018_UK_DPA", Evidence, Reason, Mindset) :-
    in_scope("3_66_2d"),
    violates_3_66_2d(App, Evidence, Reason, Mindset).
```

```
violates_3_66_2a(App, Evidence, FullReason, Mindset) :-
    relevant_party(App, Evidence),
    unauthorised_processing(App, Evidence, Reason, Mindset),
    append(Reason, ["unauthorised_processing"], FullReason).

violates_3_66_2a(App, Evidence, FullReason, Mindset) :-
    relevant_party(App, Evidence),
    unauthorised_interference(App, Evidence, Reason, Mindset),
    append(Reason, ["unauthorised_interference"], FullReason).

% This will never be reportedby the TKB.
violates_3_66_2b(App, Evidence, Reason, Mindset) :-
    relevant_party(App, Evidence),
    lack_of_transparency(App, Evidence, Reason, Mindset).

% This will never be reportedby the TKB.
violates_3_66_2c(App, Evidence, Reason, Mindset) :-
    relevant_party(App, Evidence),
    lack_of_proper_functioning(App, Evidence, Reason, Mindset).

% This will never be reportedby the TKB.
violates_3_66_2d(App, Evidence, Reason, Mindset) :-
    relevant_party(App, Evidence),
    availability_issue(App, Evidence, Reason, Mindset).

% If data confidentiality is broken, data can be processed by an unauthorised party.
unauthorised_processing(App, Evidence, FullReason, Mindset) :-
    data_confidentiality_issue(App, Evidence, Reason, Mindset),
    append([Reason], ["data_confidentiality_issue"], FullReason).

% If data integrity is broken, data can be processed by an unauthorised party.
unauthorised_processing(App, Evidence, FullReason, Mindset) :-
    data_integrity_issue(App, Evidence, Reason, Mindset),
    append([Reason], ["data_integrity_issue"], FullReason).

% If data authenticity is not satisfied, data can be processed by an unauthorised party.
unauthorised_processing(App, Evidence, FullReason, Mindset) :-
    data_authenticity_issue(App, Evidence, Reason, Mindset),
    append([Reason], ["data_authenticity_issue"], FullReason).

% If data confidentiality is broken, an unauthorised interference becomes possible.
unauthorised_interference(App, Evidence, FullReason, Mindset) :-
    data_confidentiality_issue(App, Evidence, Reason, Mindset),
    append([Reason], ["data_confidentiality_issue"], FullReason).

% If data integrity is broken, an unauthorised interference becomes possible.
unauthorised_interference(App, Evidence, FullReason, Mindset) :-
    data_integrity_issue(App, Evidence, Reason, Mindset),
    append([Reason], ["data_integrity_issue"], FullReason).

% If data authenticity is not satisfied, an unauthorised interference becomes possible.
unauthorised_interference(App, Evidence, FullReason, Mindset) :-
    data_authenticity_issue(App, Evidence, Reason, Mindset),
    append([Reason], ["data_authenticity_issue"], FullReason).
```

```
% What is in scope and out of scope in terms of the relevant articles of the law.
in_scope("3_66_1") :- true.
in_scope("3_66_2a") :- true.
in_scope("3_66_2b") :- false.
in_scope("3_66_2c") :- false.
in_scope("3_66_2d") :- false.

% We are not interested in those.
lack_of_transparency(_, _, _, _) :- false.
lack_of_proper_functioning(_, _, _, _) :- false.
availability_issue(_, _, _, _) :- false.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Technological Knowledge Base %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

data_confidentiality_issue("app173", "final_report", "plain_http_network_activity", "skeptical").
data_confidentiality_issue("app173", "final_report", "malicious_app", "credulous").
data_integrity_issue("app173", "final_report", "plain_http_network_activity", "skeptical").
data_authenticity_issue("app173", "final_report", "plain_http_network_activity", "skeptical").
app("app173", "final_report").
risk("plain_http_network_activity", "final_report").
threat("malicious_app", "final_report").
```

Bibliography

[1] A. Narayanan and V. Shmatikov, “Myths and fallacies of” personally identifiable information”,” *Communications of the ACM*, vol. 53, no. 6, pp. 24–26, 2010.

[2] N. Tabata, H. Sato, and K. Ninomiya, “Comparison of privacy consciousness between younger and older adults,” *Japanese Psychological Research*, vol. 63, no. 2, pp. 104–110, 2021.

[3] A. F. Westin, “Privacy and freedom,” *Washington and Lee Law Review*, vol. 25, no. 1, p. 166, 1968.

[4] J. A. Blumenthal, M. Adya, and J. Mogle, “The multiple dimensions of privacy: Testing lay expectations of privacy,” *U. Pa. J. Const. L.*, vol. 11, p. 331, 2008.

[5] P. M. Schwartz and D. J. Solove, “The pii problem: Privacy and a new concept of personally identifiable information,” *NYUL rev.*, vol. 86, p. 1814, 2011.

[6] J. Feinberg, *The moral limits of the criminal law.* Oxford University Press, 1984.

[7] D. J. Hand and N. M. Adams, "Data mining," *Wiley StatsRef: Statistics Reference Online*, pp. 1–7, 2014.

[8] K. Boda, Á. M. Földes, G. G. Gulyás, and S. Imre, "User tracking on the web via cross-browser fingerprinting," in *Nordic conference on secure it systems*, pp. 31–46, Springer, 2011.

[9] P. Eckersley, "How unique is your web browser?," in *International Symposium on Privacy Enhancing Technologies Symposium*, pp. 1–18, Springer, 2010.

[10] E. Uliana, K. Stathis, and R. Jago, "Magnetdroid: security-oriented analysis for bridging privacy and law for android applications," in *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law*, pp. 123–132, 2019.

[11] G. Greenwald and E. MacAskill, "Nsa prism program taps in to user data of apple, google and others," *The Guardian*, vol. 7, no. 6, pp. 1–43, 2013.

[12] J. Coggon, "Human dignity in bioethics and law by charles foster," *Journal of Law and Society*, vol. 39, no. 4, pp. 625–630, 2012.

[13] "Scytale cipher." Available at `https://www.dcode.fr/scytale-cipher` - Retrieved: 2021-12-18.

[14] "Caesar's cipher." Available at `https://www.dcode.fr/caesar-cipher` - Retrieved: 2021-12-18.

[15] F. Graf, "Confession, secrecy, and ancient societies," *Religion im kulturellen Diskurs, Religion in Cultural Discours: Festschrift für Hans G. Kippenberg zu seinem 65. Geburtstag; Essays in Honor of Hans G. Kippenberg on the Occasion of His 65th Birthday*, pp. 259–71, 2004.

[16] S. D. Warren and L. D. Brandeis, "The right to privacy," *Harvard law review*, pp. 193–220, 1890.

[17] F. Petitcolas, "La cryptographie militaire," 1883.

[18] "Post office (revenues) act 1710." Available at http://www.gbps.org.uk/information/sources/acts/1710-11-25_Act-9-Anne-cap-10.php - Retrieved: 2021-12-18.

[19] P. Cirino, "Advertisers, celebrities, and publicity rights in new york and california," *NYL Sch. L. Rev.*, vol. 39, p. 763, 1994.

[20] "Rsa 2016 - the cryptographers panel." Available at https://www.rsaconference.com/events/us16/agenda/sessions/2720/the-cryptographers-panel - Retrieved: 2021-12-18.

[21] W. L. Prosser, "Privacy, 48 calif," *L. Rev*, vol. 383, no. 10.2307, p. 3478805383, 1960.

[22] E. Samson, "The burden to prove libel: A comparative analysis of traditional english and us defamation laws and the dawn of england's modern day," *Cardozo J. Int'l & Comp. L.*, vol. 20, p. 771, 2011.

[23] F. Pakes, *Comparative criminal justice*. Routledge, 2017.

[24] "Wainwright v. home office." Available at https://publications.parliament.uk/pa/ld200203/ldjudgmt/jd031016/wain-1.htm - Retrieved: 2021-12-18.

[25] "European convention on human rights." Available at http://www.echr.coe.int/Documents/Convention_ENG.pdf - Retrieved: 2021-12-18.

[26] A. Porretta, F. Quattrone, F. Aquino, G. Pieve, B. Bruni, G. Gemignani, M. L. Vatteroni, M. Pistello, G. P. Privitera, and P. L. Lopalco, "A nosocomial measles outbreak in italy, february-april 2017," *Eurosurveillance*, vol. 22, no. 33, p. 30597, 2017.

[27] "Regulation on the protection of natural persons with regard to the processing of personal data and on the free movement of such data." Available at http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.119.01.0001.01.ENG&toc=OJ:L:2016:119:TOC - Retrieved: 2021-12-18.

[28] "Directive on the protection of natural persons with regard to the processing of personal data by competent authorities for the purposes of the prevention, investigation, detection or prosecution of criminal offences or the execution of criminal penalties, and on the free movement of such data." Available at http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.119.01.0089.01.ENG&toc=OJ:L:2016:119:TOC - Retrieved: 2021-12-18.

[29] European Union, *Charter of Fundamental Rights of the European Union*, vol. 53. Brussels: European Union, 2012.

[30] "Treaty on the functioning of the european union." Available at http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A12012E%2FTXT - Retrieved: 2021-12-18.

[31] D. J. Solove and P. M. Schwartz, "Privacy law fundamentals," *D. Solove & P. Schwartz, PRIVACY LAW FUNDAMENTALS, International Association of Privacy Professionals*, 2011.

[32] D. E. Kyvig, *Explicit and authentic acts: Amending the US Constitution, 1776-1995.* University Press of Kansas, 1996.

[33] E. Pernot-Leplay, "China's approach on data privacy law: A third way between the us and the eu?," *Penn St. JL & Int'l Aff.*, vol. 8, p. 49, 2020.

[34] "Platform architecture — android developers." Available at https://developer.android.com/guide/platform - Retrieved: 2021-12-18.

[35] R. Mayrhofer, J. V. Stoep, C. Brubaker, and N. Kralevich, "The android platform security model," *ACM Transactions on Privacy and Security (TOPS)*, vol. 24, no. 3, pp. 1–35, 2021.

[36] "Secure an android device." Available at https://source.android.com/security - Retrieved: 2021-12-18.

[37] "Permissions on android." Available at https://developer.android.com/guide/topics/permissions/overview - Retrieved: 2021-12-18.

[38] "Ca/browser forum." Available at https://cabforum.org/ - Retrieved: 2021-12-18.

[39] "Revocation still doesn't work." Available at https://www.imperialviolet.org/2014/04/29/revocationagain.html - Retrieved: 2021-12-18.

[40] "Ca/revocation checking in firefox." Available at https://wiki.mozilla.org/CA/Revocation_Checking_in_Firefox - Retrieved: 2021-12-18.

[41] "No, don't enable revocation checking." Available at https://www.imperialviolet.org/2014/04/19/revchecking.html - Retrieved: 2021-12-18.

[42] "Quic: A udp-based multiplexed and secure transport." Available at https://tools.ietf.org/html/draft-ietf-quic-transport-33 - Retrieved: 2021-12-18.

[43] D. Benjamin, "Tls ecosystem woes," in *Real World Crypto*, 2018. Available at https://www.youtube.com/watch?v=_mE_JmwFi1Y.

[44] "Content security policy level 3." Available at https://w3c.github.io/webappsec-csp/ - Retrieved: 2021-12-18.

[45] "Trusted types." Available at https://w3c.github.io/webappsec-trusted-types/dist/spec/ - Retrieved: 2021-12-18.

[46] F. Pauck, E. Bodden, and H. Wehrheim, "Do android taint analysis tools keep their promises?," in *Proceedings of the 2018 26th ACM Joint*

Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 331–341, ACM, 2018.

[47] G. Brewka and J. Dix, “Knowledge representation with logic programs,” in *International Workshop on Logic Programming and Knowledge Representation*, pp. 1–51, Springer, 1997.

[48] M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory, “The british nationality act as a logic program,” *Communications of the ACM*, vol. 29, no. 5, pp. 370–386, 1986.

[49] I. Bratko, *Prolog programming for artificial intelligence.* Pearson education, 2001.

[50] K. L. Clark, “Negation as failure,” in *Logic and data bases*, pp. 293–322, Springer, 1978.

[51] L. M. Pereira and J. J. Alferes, “Well founded semantics for logic programs with explicit negation.,” in *ECAI*, vol. 92, pp. 102–106, 1992.

[52] A. Consoli, J. Tweedale, and L. Jain, “The link between agent coordination and cooperation,” in *International Conference on Intelligent Information Processing*, pp. 11–19, Springer, 2006.

[53] S. Wang, J. Wan, D. Zhang, D. Li, and C. Zhang, “Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination,” *Computer Networks*, vol. 101, pp. 158–168, 2016.

[54] W. Ren, R. W. Beard, and E. M. Atkins, "A survey of consensus problems in multi-agent coordination," in *Proceedings of the 2005, American Control Conference, 2005.*, pp. 1859–1864, IEEE, 2005.

[55] S. Talukdar, L. Baerentzen, A. Gove, and P. De Souza, "Asynchronous teams: Cooperation schemes for autonomous agents," *Journal of Heuristics*, vol. 4, no. 4, pp. 295–321, 1998.

[56] P. Xuan, V. Lesser, and S. Zilberstein, "Communication decisions in multi-agent cooperation: Model and experiments," in *Proceedings of the fifth international conference on Autonomous agents*, pp. 616–623, 2001.

[57] Y. Labrou, T. Finin, and Y. Peng, "Agent communication languages: The current landscape," *IEEE Intelligent Systems and Their Applications*, vol. 14, no. 2, pp. 45–52, 1999.

[58] J. Larssan and B. Hayes-Roth, "Guardian: intelligent autonomous agent for medical monitoring and diagnosis," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 1, pp. 58–64, 1998.

[59] S. Bromuri and K. Stathis, "Situating cognitive agents in golem," in *International Workshop on Engineering Environment-Mediated Multi-Agent Systems*, pp. 115–134, Springer, 2007.

[60] "Bettercap." Available at `https://www.bettercap.org` - Retrieved: 2021-12-18.

[61] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love android: An analysis of android ssl

(in) security," in *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 50–61, 2012.

[62] A. Desnos *et al.*, "Androguard: Reverse engineering, malware and goodware analysis of android applications... and more (ninja!)," *Retrieved June*, vol. 10, p. 2014, 2011.

[63] "Virustotal." Available at https://www.virustotal.com - Retrieved: 2021-12-18.

[64] "X.509v3 transport layer security (tls) feature extension." Available at https://datatracker.ietf.org/doc/html/rfc7633 - Retrieved: 2021-12-18.

[65] N. Nilsson, "Teleo-reactive programs for agent control," *Journal of artificial intelligence research*, vol. 1, pp. 139–158, 1993.

[66] M. Shanahan, "The event calculus explained," in *Artificial intelligence today*, pp. 409–430, Springer, 1999.

[67] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*. Berlin: Springer, 5 ed., 2003.

[68] "The 2018 uk data protection act." Available at https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted - Retrieved: 2021-12-18.

[69] M. F. Kilkenny and K. M. Robinson, "Data quality:"garbage in–garbage out"," *SAGE Journals*, 2018.

[70] R. Mayrhofer, J. V. Stoep, C. Brubaker, and N. Kralevich, "The android platform security model," *ACM Transactions on Privacy and Security (TOPS)*, vol. 24, no. 3, pp. 1–35, 2021.

[71] "Qualys ssl labs." Available at `https://www.ssllabs.com` - Retrieved: 2021-12-18.

[72] D. Wichers, "Owasp top-10 2013," *OWASP Foundation, February*, 2013.

[73] "Dns queries over https (doh)." Available at `https://datatracker.ietf.org/doc/html/rfc8484` - Retrieved: 2021-12-18.

[74] "Specification for dns over transport layer security (tls)." Available at `https://datatracker.ietf.org/doc/html/rfc7858` - Retrieved: 2021-12-18.

[75] "Transport layer security (tls) extensions: Extension definitions." Available at `https://datatracker.ietf.org/doc/html/rfc6066` - Retrieved: 2021-12-18.

[76] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pp. 468–471, IEEE, ACM, 2016.