Security and Privacy of Protocols and Software with Formal Methods

Fabrizio Biondi and Axel Legay

Inria

Abstract. The protection of users' data conforming to best practice and legislation is one of the main challenges in computer science. Very often, large-scale data leaks remind us that the state of the art in data privacy and anonymity is severely lacking. The complexity of modern systems make it impossible for software architect to create secure software that correctly implements privacy policies without the help of automated tools. The academic community needs to invest more effort in the formal modelization of security and anonymity properties, providing a deeper understanding of the underlying concepts and challenges and allowing the creation of automated tools to help software architects and developers. This track provides numerous contributions to the formal modeling of security and anonymity properties and the creation of tools to verify them on large-scale software projects.

1 Introduction

Security and Anonymity Properties. Security and privacy are fundamental interests of computer science research. Security refers to the guarantee that the computer system being used does not act against the interest of the user, either maliciously or accidentally. The security of a system is usually associated with the following properties [33, 41, 47]:

Confidentiality Sensitive information about the user that is handled by the system cannot be accessed by unauthorized third parties;

Integrity Information handled by the system cannot be deleted, altered or modified by unauthorized third parties; and

Authentication Each agent interacting with the system is who they claim to be.

However, these basic properties are not defining which potentially sensitive information about the user is the system handling, or whether the system should have access to such information to start with [6, 11, 35]. This is the reason why we add the following property:

Anonymity The user has control on what information about them is collected by the system, and can decide how it is collected and used, by whom, and for what purpose.

While other interesting properties (e.g. availability) are considered in computer science research, this track focuses on the four properties listed above, as they allow the user to trust that their data is properly handled by the system and not used against them by third parties.

Problem: Large Attack Surface. [3, 21, 24, 44] The complexity of modern systems means that the attack surface for a malicious agent is huge. The system can be compromised at any level and leak information in hundred of possible ways, like improperly handling access rights to databases, or leaking kernel memory through improperly written implementations, or even allowing private keys to be recovered by analyzing the system's energy consumption. Complex protocol interaction means that different agents are executing their parts of the protocol on machines with different environment and operative systems.

Solution: Formal Models and Automated Tools. [15, 27, 34] The approach to secure such a complex system is the modelization of security protocols with formal languages, and of the security properties to be preserved with formal logics. This formalization effort allows us to produce automated techniques and tools that verify whether the protocols and their implementations respect the security requirements.

The formal models of systems, protocols and properties are simpler and more intuitive than their implementations, since they abstract away irrelevant details. This allows users and developers to examine the properties themselves and determine if they satisfy their needs, thus making it easier for the user to trust that the system is doing what they expect it to do and does not have any harmful behavior, either intentional or accidental.

We will discuss the use of formal methods and tools to improve the security at both the software implementation and the protocol level.

1.1 The Software Level

Attacks at the software level exploit vulnerabilities in the implementation of a secure system that are not present in the system specification. It is common for software developers and engineers to introduce bugs in an implementation, particularly when using non-strongly-typed languages [38]. Bugs that can be used to compromise the security and privacy properties of the system are known as vulnerabilities. It is common for an attacker targeting a specific system to start analyzing the system for such vulnerabilities, then writing an exploit leveraging on some vulnerabilities to steal information from or take control of the system [17].

Vulnerabilities are categorized by cause and by severity [2, 42]. Categories by cause include input and access validation errors, race conditions, secret information leaks, and many more. In particular, information leaks endanger both security and privacy properties, leaking confidential user data or even private cryptographic keys, like in the case of the Heartbleed bug.

Severity is normally categorized in High, Medium and Low, where High severity means that the vulnerability makes it possible for an attacker to violate the security properties of the system, Low severity means that the vulnerability only provides the attacker with more information to look for more severe vulnerabilities, and Medium severity is anything that is not High or Low severity.

Helping software developers writing bug-free code is a large field of computer science and software engineering. However, from the security and privacy perspective we have a slightly different view of the problem, since we are only interested in finding and reducing vulnerabilities.

For instance, considering the handling of private and confidential information, we can develop tools that track how a given implementation of a protocol handles such information. Taint analysis [32, 45] and information leakage computation techniques [9, 10, 22, 46] can trace the flow of information in a particular system given its source code, thus detecting vulnerabilities that would allow private information to be inferred by unauthorized users. The track presents a new state of the art in the automated detection of information leaks in large projects.

Additionally, protocol implementations can leverage the formal specifications of the protocols. Automated tools [8, 23, 25, 48] can be used to verify whether an implementation respects the formal protocol implementation it is supposed to implement, or even automatically produce code from the protocol specifications that is guaranteed to correctly implement them, avoiding vulnerabilities caused by design errors. The track presents a new formal framework to help software developers validate their software for the complex case of cyber-physical systems.

1.2 The Protocol Level

Protocols model the exchange of communications and data between agents to obtain a common goal. The formalization of protocols is necessary to determine univocally what each agent is supposed to do, and to be able to prove that their behavior contributes to achieving the goal of the protocol. Many formal languages for protocols exist, capturing different primitives and granularity of the communications. One of the classical approaches is Burrows-Abadi-Needham (BAN) logic [13], used to model authentication systems since it allows to model what agents know and believe on each other during the protocol, and it assumes that the network itself is vulnerable to tampering and information leakage.

More recently, model checking techniques and properties have been shown to be more effective than BAN logic to model protocols and automatically verify whether they respect security properties.

Information-theoretical properties like non-interference [20, 39] can be used to prove that the communications of an agent do not leak information about the agent's secret information in any way, allowing to automatically verify whether a protocols guarantees confidentiality and anonymity. When this strong property is impossible to guarantee while achieving the protocol's goal, quantitative leakage computation [12, 16, 28, 43] can prove that the amount of secret information leaked is too small to significantly hinder the confidentiality and anonymity

properties, or at worst to exactly quantify the loss of anonymity allowing the agent to decide if it is an acceptable price to pay to run the protocol.

Temporal properties [5, 14, 36] are concerned with the sequence of operations performed by the protocol, and can be used to prove that the required steps to achieve the protocol's goal are always executed correctly and in the correct order. This enables formally verifying that if the protocol succeeds all the proper steps have been executed, and if it fails it does so graciously and properly notifying the agents of the cause of the failure. Temporal properties are very close to the protocol's flow of operations, and mature tools exist [8, 23, 25, 48] to automatically verify that they are respected even by complex system interactions.

Cryptographic properties [1, 7, 19, 30] are used to guarantee security properties of the protocol, and are often based on complexity results of problems that are hard to treat at the current state of the art. Cryptographic properties can be used to guarantee the hardness of retrieving private keys in shared-key and public-key cryptosystems, verify agents' identities in authentication schemes, provide secure key exchange protocols and multiparty computation over unsecured channels, and be used to express most security properties. While some of the hardness results they are based on are currently unproven, and sometimes technological advancements may cripple protocols previously considered secure, cryptographic properties and primitives are the building blocks of most of the successful secure protocols currently used in any computer system.

The definition of secure and anonymous protocols in terms of formal models and properties allows automated verification of the protocols' correctness. This is a fundamental requirement for a user to be able to trust that the protocol is correctly designed to defend their interests and the security of their data. While many authentication and confidentiality properties can be defined as cryptographic trace properties and analyzed with temporal logics, anonymity properties depend on the data flow and interaction between different agents and are hard to define in terms of traces. The track presents contributions to model systems with process calculi and model transformations, allowing the expression and formal verification of anonymity properties.

Privacy and anonymity policies are often defined by legislative bodies in natural language. The duty of translating these policies into formal properties falls on computer scientists. Due to the inherent ambiguity of natural language, the formalization of policies may not correspond with the legislator's intent. Also, since protocols can be distributed among different legislative jurisdictions, it is not clear that all agents involved conform to the same rules and enforce the required policies. The track presents contributions to validate whether formal security protocols correctly implement legislative policies, and to automatically negotiate security policies between agents to guarantee that data owners and consumers agree on the policies for the treatment of the data.

2 Contribution to the Track

This track provides several contributions to apply formal methods to improve the security and privacy of system at the software and protocol levels. A summary is given here.

2.1 On Building Secure Software

The track offers two major contributions on using information flow and formal models to find vulnerabilities in software implementations:

- Information leaks in software may have devastating consequences, as demonstrated for instance by the Heartbleed bug. Academic work focuses on information theory to compute the amount of information leaked by a software implementation, but tools able to perform an automated analysis of real-world complex C code are still lacking. On the other hand, effective working solutions rely on ad-hoc principles that have little theoretical justification.
  In [29], the authors bridge this chasm between advanced theoretical work and concrete practical needs of programmers developing real world software. They present an analysis, based on clear security principles and verification tools, which is largely automatic and effective in detecting information leaks in complex C code running everyday on millions of systems worldwide.
- Cyber-physical systems are computer systems that interact with physical objects. Such systems are composed of several software components executed on different processors and interconnected through physical buses. These complex systems collocate functions operating at different security levels, which can introduce unexpected interactions that affect system security. The security policy for these systems is realized through various complex physical or logical mechanisms. The security policy, as a stakeholder goal, is then refined into system requirements and implementation constraints that guarantee security objectives. Unfortunately, verifying the correct decomposition and its enforcement in the system architecture is an overwhelming task. Because requirements are often written manually, they can be contradictory and inconsistent, which can lead to incorrect implementations. To overcome these issues, requirements must be specified using a formal and unambiguous language, traced through the system architecture, and automatically verified throughout the development process.
  In [31], the authors introduce a modeling framework for the design and validation of requirements from a security perspective. The framework is composed of a new language for requirements specification, an extension of the Architecture Analysis & Design Language, for specifying security and a set of theorems to check the requirements against the architecture. The framework provides the capability to validate the requirements of several candidate architectures and analyze the impact of changes to requirements and architecture during development. This model-based approach helps software architects and developers detect requirements and architecture issues

early in the development life cycle and avoid the propagation of their effects during integration.

2.2 On Designing Privacy-preserving Protocols

The track offers approaches to formalize the transmission of private and confidential information in protocols, guaranteeing that user privacy is respected:

- Formal, symbolic techniques for modeling and automatically analyzing security protocols are extremely successful and were able to discover many security flaws. Initially, these techniques were mainly developed to analyze authentication and confidentiality properties. Both these properties are trace properties and efficient tools for their verification exist. In more recent years anonymity-like properties have received increasing interest. Many flavors of anonymity properties are naturally expressed in terms of indistinguishability and modeled as an observational equivalence in process calculi.
  In [26], the authors present recent advances in the verification of such indistinguishability properties.
- Within distributed systems with completely distributed interactions between parties with mutual distrust, it is hard to control the (illicit) flowing of private information to unintended parties.
  In [40], the authors propose a novel model-based approach based on model transformations to build a secure-by-construction multiparty distributed system. First, starting from a component-based model of the system, the designer annotates different parts of it in order to define the security policy. Then, the security is checked and when valid, a secure distributed model, consistent with the desired security policy, is automatically generated. To illustrate the approach, the authors present a framework that implements our method and use it to secure an online social network application.
- The users of location-based services (LBSs) are always vulnerable to privacy risks since they need to disclose, at least partially, their locations in order to receive personalized services.
  In [18], the authors discuss the adaptation of differential privacy to the context of LBSs. More precisely, assuming that the LBS provider is queried with a perturbed version of the position of the user instead of his exact one, differential privacy is used to quantify the level of indistinguishability (privacy) provided for the user's position by such a perturbation. In this setting, the adaptation of differential privacy can lead to various models depending on the precise form of indistinguishability required. The authors describe an example of these models, the (D,e)-location privacy, which is directly inspired from the standard differential privacy model. In this model, they present the characterization of (D,e)-location privacy for a mechanism and also measure the utility of this mechanism with respect to an arbitrary loss function. Afterwards, they present a special class of mechanisms, called symmetric mechanisms in which all locations are perturbed in a unified manner through a noise function, focusing in particular on circular noise functions.

They show that under certain assumptions, the circular functions are rich enough to provide the same privacy and utility levels as other more complex (non-circular) noise functions, while being easier to implement. Finally, the authors describe the extension of the above model to a generalized notion for location privacy, called l-privacy capturing both (D,e)-location privacy and also the recent notion of geo-indistinguishability.

2.3 On Automated Policy Enforcement

The track contributes formal approaches to policy enforcement, allowing the design of systems that respect legal bounds by design and guarantee that the other entities are treating private data properly:

- Handling personal data adequately is one of the biggest challenges of our era. Consequently, law and regulations are in the process of being released, like the European General Data Protection Regulation (GDPR), which attempt to deal with these challenging issue early on. The core question motivating this work is how software developers can validate their technical design vis-a-vis the prescriptions of the privacy legislation.
  In [4], the authors outline the technical concepts related to privacy that need to be taken into consideration in a software design. Also, the authors extend a popular design notation in order to support the privacy concepts illustrated in the previous point. Finally, they show how some of the prescriptions of the privacy legislation and standards may be related to a technical design that employs our enriched notation, which would facilitate reasoning about compliance.
- Privacy is a major concern in large of parts of the world when exchanging information. Ideally, we would like to be able to have fine-grained control about how information that we deem sensitive can be propagated and used. While privacy policy languages exist, it is not possible to control whether the entity that receives data is living up to its own policy specification.
  In [37], the authors present our initial work on an approach that empowers data owners to specify their privacy preferences and data consumers to specify their data needs. Using a static analysis of the two specifications, they find a communication scheme that complies with these preferences and needs. While applicable to online transactions, the same techniques can be used in development of IT systems dealing with sensitive data. To the best of our knowledge, no existing privacy policy languages supports negotiation of policies, but only yes/no answers.

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 36–47. ACM, 1997.

2. O. H. Alhazmi, S. Woo, and Y. K. Malaiya. Security vulnerability categories in major software systems. In S. Rajasekaran, editor, *Proceedings of the Third IASTED International Conference on Communication, Network, and Information Security, October 9-11, 2006, Cambridge, MA, USA*, pages 138–143. IASTED/ACTA Press, 2006.
3. R. Anderson. Why information security is hard-an economic perspective. In *Computer security applications conference, 2001. acsac 2001. proceedings 17th annual*, pages 358–365. IEEE, 2001.
4. T. Antignac, R. Scandariato, and G. Schneider. A privacy-aware conceptual model for handling personal data. In *these proceedings*.
5. C. Baier, J.-P. Katoen, and K. G. Larsen. *Principles of model checking*. MIT press, 2008.
6. M. Bailey. *Complete Guide to Internet Privacy, Anonymity & Security*. Nerel, 2011.
7. G. Barthe, B. Grégoire, and S. Zanella Béguelin. Formal certification of code-based cryptographic proofs. *ACM SIGPLAN Notices*, 44(1):90–101, 2009.
8. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, Oct. 1995.
9. F. Biondi, A. Legay, P. Malacaria, and A. Wasowski. Quantifying information leakage of randomized protocols. *Theoretical Computer Science*, 597:62–87, 2015.
10. F. Biondi, A. Legay, L. Traonouez, and A. Wasowski. QUAIL: A quantitative security analyzer for imperative code. In N. Sharygina and H. Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 702–707. Springer, 2013.
11. S. Bosworth. *Computer Security Handbook*. John Wiley & Sons, Inc., New York, NY, USA, 4th edition, 2002.
12. C. Braun, K. Chatzikokolakis, and C. Palamidessi. Quantitative notions of leakage for one-try attacks. *Electr. Notes Theor. Comput. Sci.*, 249:75–91, 2009.
13. M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 426, pages 233–271. The Royal Society, 1989.
14. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.
15. V. Cortier and S. Kremer, editors. *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*. IOS Press, 2011.
16. D. E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, 1976.
17. M. Dowd, J. McDonald, and J. Schuh. *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley Professional, 2006.
18. E. ElSalamouny. Differential privacy models for location-based services. In *these proceedings*.
19. N. Ferguson, B. Schneier, and T. Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing, 2010.

20. R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *International Colloquium on Automata, Languages, and Programming*, pages 354–372. Springer, 2000.
21. N. Gruschka and M. Jensen. Attack surfaces: A taxonomy for attacks on cloud services. In *IEEE CLOUD*, pages 276–279, 2010.
22. J. Heusser and P. Malacaria. Quantifying information leaks in software. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 261–269. ACM, 2010.
23. G. Holzmann. *Spin Model Checker, the: Primer and Reference Manual*. Addison-Wesley Professional, first edition, 2003.
24. M. Howard. Attack surface: Mitigate security risks by minimizing the code you expose to untrusted users. *MSDN Magazine (November 2004)*, 2004.
25. C. Jegourel, A. Legay, and S. Sedwards. A Platform for High Performance Statistical Model Checking – PLASMA. In C. Flanagan and B. König, editors, *TACAS 2012 - 18th International Conference Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *LNCS - Lecture Notes in Computer Science*, pages 498 – 503, Tallinn, Estonia, Mar. 2012. Springer.
26. S. Kremer. Automated verification of equivalence properties for cryptographic protocols. In *these proceedings*.
27. C. E. Landwehr. Formal models for computer security. *ACM Comput. Surv.*, 13(3):247–278, Sept. 1981.
28. P. Malacaria. Algebraic foundations for quantitative information flow. *Mathematical Structures in Computer Science*, 25(2):404–428, 2015.
29. P. Malacaria, M. Tautchning, and D. DiStefano. Information leakage analysis of complex C code and its application to OpenSSL. In *these proceedings*.
30. C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE journal on selected areas in communications*, 21(1):44–54, 2003.
31. M.-Y. Nam, J. Delange, and P. Feiler. Integrated modeling workflow for security assurance. In *these proceedings*.
32. J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. 2005.
33. T. Olovsson. A structured approach to computer security. Technical report, 1992. 33.
34. R. Patel, B. Borisaniya, A. Patel, D. Patel, M. Rajarajan, and A. Zisman. *Comparative Analysis of Formal Model Checking Tools for Security Protocol Verification*, pages 152–163. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
35. K. Peng. *Anonymous Communication Networks: Protecting Privacy on the Web*. Auerbach Publications, Boston, MA, USA, 1st edition, 2014.
36. A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.
37. C. W. Probst. Guaranteeing privacy-observing data exchange. In *these proceedings*.
38. B. Ray, D. Posnett, V. Filkov, and P. Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 155–165, New York, NY, USA, 2014. ACM.
39. P. Y. Ryan and S. A. Schneider. Process algebra and non-interference. *Journal of Computer Security*, 9(1-2):75–103, 2001.
40. N. B. Said, T. Abdellatif, S. Bensalem, and M. Bozga. A model-based approach to secure multiparty distributed systems. In *these proceedings*.
41. K. A. Scarfone, W. Jansen, and M. Tracy. Sp 800-123. guide to general server security. Technical report, Gaithersburg, MD, United States, 2008.

42. R. Seacord and A. Householder. A structured approach to classifying security vulnerabilities. Technical Report CMU/SEI-2005-TN-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.
43. G. Smith. On the foundations of quantitative information flow. In L. de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 288–302. Springer, 2009.
44. K. So. Cloud computing security issues and challenges. *International Journal of Computer Networks*, 3(5), 2011.
45. D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. Bitblaze: A new approach to computer security via binary analysis. In *International Conference on Information Systems Security*, pages 1–25. Springer, 2008.
46. C. G. Val, M. A. Enescu, S. Bayless, W. Aiello, and A. J. Hu. Precisely measuring quantitative information flow: 10k lines of code and beyond. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 31–46. IEEE, 2016.
47. H. Venter and J. Eloff. A taxonomy for information security technologies. *Computers & Security*, 22(4):299 – 307, 2003.
48. W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda. Model checking programs. *Automated Software Engg.*, 10(2):203–232, Apr. 2003.