The Use of Formal Methods in the Analysis of Trust (Position Paper)

Michael Butler, Michael Leuschel, Stéphane Lo Presti, Phillip Turner *

University of Southampton, SO17 1BJ, Southampton, United Kingdom
{mjb,mal,splp,pjt}@ecs.soton.ac.uk

Abstract. Security and trust are two properties of modern computing systems that are the focus of much recent interest. They play an increasingly significant role in the requirements for modern computing systems. Security has been studied thoroughly for many years, particularly the sub-domain of cryptography. The use of computing science formal methods has facilitated cryptanalysis of security protocols. At the moment, trust is intensively studied, but not well understood. Here we present our approach based on formal methods for modelling and validating the notion of trust in computing science.

1 Introduction

Recent years have seen a growing concern with security properties of computing systems. This concern is mainly caused by two reasons. First, there is an increasing number of faults in computing systems. This increase in turn ensues from two facts. The penetration of computing science in our professional and personal lives is still expanding, as new computing paradigms such as pervasive computing show. At the same time, programs become overly cluttered and computationally and semantically more complex. The second reason explaining security concerns is that the concept of security itself is widening. This is illustrated by recent problems like privacy breaches (e.g. spam) or violations of legal obligations (e.g. liability via software license).

Notions of trust are constituent in several cryptographic methods, representing the confidence in the association of a cryptographic key to the identity of a principal. Recent multidisciplinary studies on trust envisage the concept as a more general and richer notion than security. Many models of trust have been devised, each concentrating on disparate aspects, among which are recommendations and reputation, belief theory, or risk and uncertainty. It appears that the vast number of notions composing trust defies its systematic analysis.

Computing science formal methods [11] stem from mathematics and aim to help design, develop, analyse and validate software so that it is correct, error-free and robust. Formal models are built on well-known mathematical elements, like sets or functions, and can be analysed against accurate properties, such as consistency or completeness. Formal methods include Petri nets, abstract state machines, process calculi, temporal

* This work has been funded in part by the T-SAS (Trusted Software Agents and Services in Pervasive Information Environment) project of the UK Department of Trade and Industry's Next Wave Technologies and Markets Programme.

and belief logics, and languages such as Z [4], CSP [5] and Alloy [9]. The last decade has seen a trend to use formal methods in computing science, notably in the context of industrial software engineering, because they provide solid methods, produce clear models and have good tool support.

In this paper, we present in section 2 how the security field has used formal methods to solidly build some of its foundations on mathematically proven results. We show initial works in the application of formal methods in trust in section 3, arguing that trust is only at the beginning of its path to make the most of formal methods. Our structured approach based on UML [14] and B [13] formal methods is finally defined in section 4.

2 Formally Proving Security Properties

Security is one of the major problems that computer scientists have to confront nowadays. Security analysis of computing systems consists of creating models of how they operate, may be attacked, and should behave. Formal methods are helpful at modelling and validating existing computing systems with regard to security properties because they provide a structured approach and accurate notations.

In the context of security, the *system model* must not only abstract the programs implementing the system functionalities but also the communication protocols that are used. Formal approaches have been successfully applied to that latter task, for example with the Z notation or the B method [15]. Recently, the analysis [26] of layers of network protocols, involving the commonly used TLS/SSL protocols, have been a further beneficiary of the formal approach.

The model of the possible attacks to the system is called the *threat model* and defines the capabilities of the attacker. The Dolev-Yao threat model traditionally represents an attacker that can overhear, intercept, and synthesise any message and is only limited by the constraints of the cryptographic methods used. This omnipotence has been very difficult to model and most threat models simplify it, as, for example, the attacker in ubiquitous computing [22].

Next, the desired properties of the system need to be defined. Security encompasses six basic sub-properties: authentication, data integrity, confidentiality, non-repudiation, privacy, and availability. Specification of the chosen properties is in general dependent on the notation chosen for the system and threat models.

The last task is to verify that the security properties hold in the system model, complemented by the threat model if it exists. Many formal methods ease this step by applying powerful automated techniques, like test generation or model checking. General formal tools can be used, like the Coq theorem prover [3] that has been used for the verification of the confidentiality of the C-SET protocol [8], or specific ones devised, such as Casper [10] for compiling abstract descriptions to the CSP language, or SpyDer [23] to model-check security properties in the spy-calculus.

In summary, formal methods have benefited security analysis of computing systems by providing systematic methods and reusable tools in order to obtain mathematically proven results. The use of formal methods for security analysis is a very active domain, which evolves with progress from the formal methods and provides a testbed for them.

3 Formally Modelling Trust

Trust has recently attracted much focus, notably in the context of computing science and more specifically computer security. Marsh [24] gave an early (1992) formal model of trust, highlighting the combination of basic and general trust and agent capabilities into situational trust via ad hoc notations. Griffiths et al [21] made use of the Z formal notation to specify cooperative plans in multi-agent systems, annotating these plans with trust information. Many mathematical models have also been devised, for example in game theory (e.g. Birk's model [2]) or probability theory (e.g. Jøsang's Subjective Logic [1]).

More recently, Grandison [25] devised the SULTAN trust management system and his primitives were expressed in the manner of a logic programming language. SULTAN is similar to works on *trust policy languages*. Trust policy languages (which are inspired by security policies) specify what is permitted and prohibited regarding trust decisions, rather than expressing how. They were first devised in the context of Public Key Infrastructures, like IBM's Trust Policy Language or Fidelis [28]. Recent works exhibit more general policies, like those of the SECURE project [17] where domain theory is used to define trust policies able to specify spam filters.

Trust is a complex notion that is not well understood. Growing interest in modelling the notion of trust has given rise to a plethora of models and many aspects of trust are currently being studied. However, these models are difficult to compare directly because they are expressed in diverse ways, i.e. sociological or economic terms, and furthermore use specific notations, thus preventing an unambiguous interpretation. Identifying trust requirements is not always easy and, because they lead to a clearer model of a system and guide its analysis, formal specifications can ease that identification.

4 An Approach to the Modelling and Validation of Trust

The T-SAS (Trusted Software Agents and Services in Pervasive Information Environment) project [27] aims to identify critical trust issues in pervasive computing. In particular, it aims to develop tools and rigorous techniques for validating the trustworthiness of agent and Semantic Web/Grid technologies that support pervasive systems.

The identification of critical trust issues for pervasive environments is hampered by both the diverse literature on trust and lack of expertise by system designers and analysts at identifying issues of *trust*. As noted above, existing definitions of trust also tend to be either specific to particular problem domains, or contrarily, too general. This often leads to specifications impoverished of trust content suitable for analysis and formalisation. Finally, pervasive systems require that user-centric issues are at least as important as purely technical concerns.

To address these problems, whilst ensuring that scenarios studied are sufficiently realistic, the initial phase of this project has focused on the development of an analysis framework grounded in propitious (healthcare) scenarios and use-cases [16]. It is an iterative process of scenario validation by domain experts (e.g. clinicians), identification of trust issues with cross-scenario checking, and domain expert aided scenario maturation. As this process repeats, the scenarios become increasingly rich with trust

related detail and the taxonomy of trust derived from the input scenarios stabilises. In our analyses, trust issues have fallen into eleven basic areas. Viz., Source versus Interpretation, Accuracy, Audit trails, Authorisation, Identification, Personal Responsibility, Reliability/Integrity, Availability, Reasoning, Usability and Harm. The relationship between trust categories was broadly in agreement with the literature.

Our current work focuses upon the formal specification stage of a software and hardware prototype. The prototype healthcare application operates on a PDA to support clinicians in a pervasive environment with medical image messaging services. This application is based on a use-case representing a clinician roaming in the pervasive environment of his hospital and using his PDA to display pictures on a neighbour device or to access the information of a patient in an adjoining bed. The PDA currently has image capture, wireless transmission and receipt and can provide telemetry for location determination. The prototype PDA and infrastructure provides furtive ground for dealing with real-time and practical issues whilst retaining many trust concerns.

Using a single method (whether formal or not) to develop complex software and/or hardware systems may limit the ability to adequately tackle complex problems in the large. Unfortunately, many issues of trust are interrelated and highly context dependent. Therefore, simplification of a system which results in loss of this context or corruption of trust interdependencies and interactions is dangerous.

Formal methods are often associated with applications with some critical aspect with severe consequences of fault. For example, safety-, economic-, or security-critical. We believe that users' trust in pervasive computing environments is prone to significant collapse and also that the consequences would be equally undesirable. In short, pervasive computing applications are trust-critical. Yet, the widespread adoption of formal techniques to deal with trust issues is not solely based on risk aversion – tools must be developed that will be used by software engineers, designers and system analysts. Also, formal specifications are not readily communicable to the non-specialist.

In addition to the ability to visually communicate and simplify complex designs, semi-formal techniques such as UML offer the developer additional benefits such as maintainability and re-usability. Despite several studies showing that formal development requires approximately the same overall effort as traditional approaches [12] whilst providing the detection and correction of specification errors early in the development life-cycle, uptake has again, remained slow.

Finally, given the currently limited understanding of trust, it seems sensible to adopt an approach that automatically detects inconsistencies and enables system designers to produce unambiguous and consistent specifications.

In order to successfully negotiate the problems of developers (expertise and thinking methods, visualization, re-usability, maintenance, communicability), we use UML case tools which provide a powerful visual notation which can itself be analysed, tested and validated automatically. The UML is an intuitive and powerful visual notation that decomposes a formal model of a system into various diagrams, such as class, collaboration or statechart diagrams. To automate validation of the models we need to use a method which allows formal proof. We chose the B language, which is an abstract machine notation that structures systems into hierarchy of modules. Each B module is

made of components that are themselves refined at different levels of abstraction. Figure 1 portrays an overview of this approach.

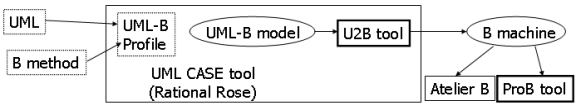

Fig. 1. Overview of approach

To annotate UML with B, we use a UML Profile, called UML-B [6], that defines a specific kind of UML model that has a particular semantics. Figure 1 illustrates a UML-B model. In UML-B, class and statechart diagrams are annotated with B code using an object-oriented dot style. A tool, called U2B [7], then automatically generates, whenever possible, an equivalent B specification from the UML-B model.

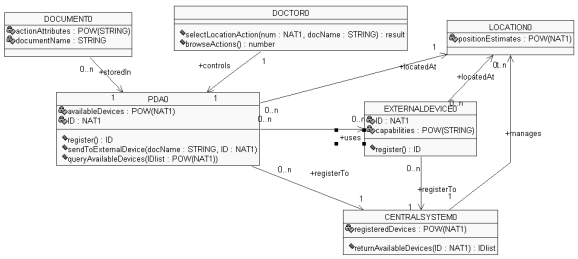

Fig. 2. UML-B screenshot

The final step is to validate our B models using a combination of automated test case generation (e.g. ProTest [19]) and model checking (e.g. ProB [18]). Figure 3 shows a screenshot of the ProB tool. The top left shows the B machine under examination, bottom left shows the current machine state and status of the invariants. Right of the figure shows the states visited during the model checking of the specification.

In the context of our use-case, the basic components of trust will be expressed by means of invariants of the B machines. At this stage during our development, the invariants represent properties of the categories Accuracy, Authorisation, Identification and Reliability/Integrity that hold between the various elements of the system (e.g. PDA, Web Services, etc.). We are testing in these models, among other aspects, whether the PDA displays a picture on a wall-mounted screen in a trustworthy manner. Figure 2 shows an example class diagram relating a doctor, his PDA, a document (for viewing on an external device), an external device (display), and a centralised system for managing services based on the user's location, gained from WiFi 802.11 signal processing.

These tools will verify that our UML-B models are consistent, thus proving the trust properties that we have specified in the B invariants. If the properties do not hold, the test case or the counter-example provided by the model checker will enable us to analyse where the problem is and formulate a solution. We would then go into another round of modelling and validation. Finally, we note that this UML and B hybridisation

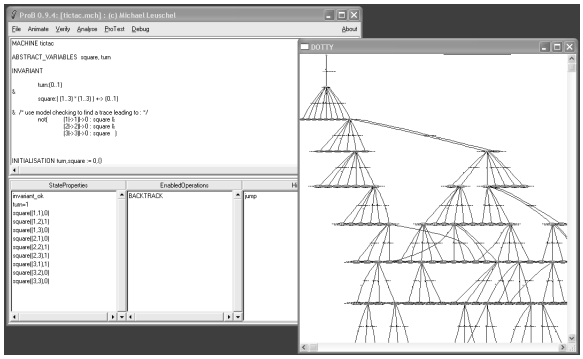

Fig. 3. ProB screenshot

has been favourably examined in an industrial setting, showing that not only are the features of the B-Method and UML complimentary, but that development with these tools was acceptable to commercial enterprise [20].

5 Conclusion

Formal methods for the specification of computer systems and their required properties have shown themselves a valuable tool for security analysis. Much work in the domain of trust devised more or less formal models, thus providing insight into the notion of trust but without formal proofs of the claimed results. The notion of trust remains elusive and has not yet achieved the same level of knowledge that security has.

The lack of formality in the followed approaches is sometimes the cause of misunderstanding and prevents the validation of the proposed models. We believe that validation is necessary to acquire a sufficient confidence in a model and formal methods can provide us with the tools to exhaustively check the proposed solutions. Here we suggest that the rigorous process of formal specification, with its associated techniques and tools for model checking and test case validation, will be as valuable to the study of trust as it has been to date for security.

Prior work in our project, based on several real-world scenarios and applications, produced a set of basic components of trust, which with tools for assisting formal specification and validation are being utilised to expedite formal analysis and test this suggestion. We believe that the practical application of formal methods can facilitate the development and evolution of the field of trust analysis in computing systems.

References

1. A. Jøsang. Trust-based decision making for electronic transactions. In *Proc. of the 4th Nordic Workshop on Secure IT Systems (NORDSEC'99)*, Sweden, November 1999.
2. Andreas Birk. Learning to Trust. In *Trust in Cyber-societies, Integrating the Human and Artificial Perspectives*, volume 2246 of *LNCS*. Springer, 2001.
3. B. Barras and al. The Coq proof assistant reference manual: Version 6.1. Technical Report INRIA RT-0203, May 1997.
4. C. Boyd. Security Architectures Using Formal Methods. *IEEE Journal on Selected Areas in Communications*, 11(5):694–701, June 1993.

5. C. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
6. C. Snook, M. Butler and I. Oliver. Towards a UML profile for UML-B. Technical Report DSSE-TR-2003-3, University of Southampton, UK, 2003.
7. Colin Snook and Michael Butler. Verifying Dynamic Properties of UML Models by Translation to the B Language and Toolkit. In *Proc of UML 200 Workshop Dynamic Behaviour in UML Models: Semantic Questions*, York, October 2000.
8. D. Bolignano. Towards the Formal Verification of Electronic Commerce Protocols. In *Proc. of the 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.
9. D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11:256–290, 2002.
10. G. Lowe. Casper: A Compiler for the Analysis of Security Protocols. In *Proc. of the 10th IEEE Computer Security Foundations Workshop*, pages 53–84, USA, 1997.
11. J. Bowen. Formal Methods. `http://www.afm.lsbu.ac.uk`.
12. J. Draper, H. Treharne, B. Ormsby B. and T. Boyce. Evaluating the B-Method on an Avionics Example. *Data Systems in Aerospace Conf (DASIA'96)*, pages 89–97, 1996.
13. J-R Abrial. *The B-Book*. Cambridge University Press, 1996.
14. J. Rumbaugh, I. Jacobson and G. Booch. *The Unified Modelling Language Reference Manual*. Addison-Wesley, 1998.
15. M. Butler. On the Use of Data Refinement in the Development of Secure Communications Systems. *Formal Aspects of Computing*, 14(1):2–34, October 2002.
16. M. Butler and al. Towards a Trust Analysis Framework for Pervasive Computing Scenarios. In *Proc of the 6th Intl Workshop on Trust, Privacy, Deception, and Fraud in Agent Societies*, Australia, July 2003.
17. M. Carbone, M. Nielsen and V. Sassone. A Formal Model for Trust in Dynamic Networks. In *Proc. of the Intl Conf on Software Engineering and Formal Methods, SEFM 2003*, pages 54–61. IEEE Computer Society, 2003.
18. M. Leuschel and M. Butler. ProB: A Model-Checker for B. In *Proc of FM 2003: 12th Intl. FME Symposium*, pages 855–874, Italy, September 2003.
19. M. Satpathy, M. Leuschel and M. Butler. ProTest: An Automatic Test Environment for B Specifications. In *International Workshop on Model Based Testing*, 2004.
20. M. Satpathy, R. Harrison, C. Snook and M. Butler. A Comparative Study of Formal and Informal Specifications through an Industrial Case Study. IEEE/ IFIP Workshop on Formal Specification of Computer Based Systems, 2001.
21. N. Griffiths, M. Luck and M. d'Inverno. Annotating Cooperative Plans with Trusted Agents. In *Trust, Reputation, and Security: Theories and Practice*, LNAI 2631. Springer, 2002.
22. S. Creese, M. Goldsmith, B. Roscoe and I. Zakiuddin. The Attacker in Ubiquitous Computing Environments: Formalising the Threat Model. In *Proc. of the 1st Intl Workshop on Formal Aspects in Security and Trust*, pages 83–97, Italy, 2003.
23. S. Lenzini, S. Gnesi and D. Latella. SpyDer, a Security Model Checker. In *Proc. of the 1st Intl Workshop on Formal Aspects in Security and Trust*, pages 163–180, Pisa, Italy, 2003.
24. S. Marsh. Trust in Distributed Artificial Intelligence. In *Artificial Social Systems, (MAAMAW'94*, LNCS 830, pages 94–112. Springer, 1994.
25. T. Grandison. *Trust Management for Internet Applications*. PhD thesis, University of London, UK, 2003.
26. The FORWARD project. Protocol Synthesis Feasibility Report, FORWARD Deliverable D2. `http://www.nextwave.org.uk/downloads/forward_psfr.pdf`.
27. University of Southampton and QinetiQ. T-SAS (Trusted Software Agents and Services in Pervasive Information Environment) project. `http://www.trustedagents.co.uk`.
28. W. Teh-Ming Yao. *Fidelis*: A Policy-Driven Trust Management Framework. In *Proc. 1st Intl Conf on Trust Management (iTrust 2003)*, Greece, May 2003.