Building High Assurance Secure Applications using Security Patterns for Capability-Based Platforms

Paul Rimba
NICTA, Level5, 13 Garden Street, Eveleigh, NSW, Australia
School of Computer Science and Engineering, University of New South Wales, NSW, Australia
Paul.Rimba@nicta.com.au

***Abstract*— Building high assurance secure applications requires the proper use of security mechanisms and assurances provided by the underlying secure platform. However, applications are often built using security patterns and best practices that are agnostic with respect to the intricate specifics of the different underlying platforms. This independence from the underlying platform leaves a gap between security patterns and underlying secure platforms. In this PhD research abstract, we propose a novel approach to bridge this gap. Specifically, we propose reusable capability-specific design fragments for security patterns, which are specialization for patterns in a capability-based system. The focus is on systems that adhere to a capability-based security model, which we consider as the underlying platforms, to provide desired application-wide security properties. We also discuss assumptions and levels of assurance for these reusable designs and their use in the verification of application designs.**

***Index Terms*—Security Patterns, Assurance, Capability, Platform, Whole System.**

I. Introduction

Secure platforms and trusted computing bases can be the foundation for building large-scale, high-assurance, secure applications [6, 16]. Secure applications are often designed with the use of security patterns [1], tactics [2] and best practices, which are usually informally described, have vague claims [1, 10] and abstract from specific platforms. It is not trivial to *properly* use the specific security mechanisms and the assurance of the underlying platforms for real-world large applications design with these patterns, tactics, or best practices.

A study [3] to assess the applicability of security patterns reported that existing security patterns do not provide guidance on how to implement the patterns correctly. This is one reason why security patterns have not been more widely adopted. However, implementation guidance without consideration of specific underlying platforms may not be meaningful.

Our goal is to bridge the gap between informal security patterns and the underlying platform, by providing platform-specific design fragments for security patterns. The platform-specific design fragments will be associated with two things: 1) reusable design-level verification techniques to substantiate the informal security-related claims, and 2) explicit assumptions specifying the assurance required of the platform. These are intended to be able to be used to design and verify a variety of applications.

In this paper, we focus on platforms that adhere to a capability-based security model [4]. A capability is a communicable, difficult-to-forge token of authority that references an object with associated access rights [4]. We build a collection of design fragments for security patterns for capability-based platforms, which will be used in composition of each other to form the final application design.

For capability-specific design fragments of security patterns, we aim to provide reusable verifications of these design fragments. The specific approach for design-level verification depends on the security property of interest (confidentiality, integrity, or availability), the techniques chosen, and the intended way of reusing component verifications for application design verification [5]. We present our overall thinking and illustrate it with model-checking verification, but do not limit our verification approach to model checking. The selected verification approach will have an impact on how design-level verification can be reused.

Informal security patterns can serve as inspiration for capability-specific design fragments and their intended properties. However, as the patterns are informal and ambiguous, it is difficult to verify the security properties that they claim. Moreover, patterns usually have implicit assumptions. We try to make these assumptions explicit and also formalize the patterns to allow formal verification of security properties.

Underlying platforms in this paper refer to systems that adhere to a capability-based security model.We leverage our group's prior work, seL4 [6] – a formally verified capability-based operating system microkernel, and use it as the target platform. We model designs for capability-based platforms using the Serscis Access Modeller (SAM) notation [21]. Although seL4 is the target platform, our work will be applicable to other systems that comply with a capability-based security model.

We discuss related work in the next section, before describing our proposed approach of building secure systems with patterns based on capabilities. We then present our expected contributions and validation strategy before discussing about the progress made towards realizing our vision.

II. Related Work

Capabilities, introduced by Dennis and Van Horn [4], have several advantages for building security applications, such as solving the Confused Deputy problem, [7] and making it easier to apply the Principle of Least Privilege [8]. seL4 is a capability-based microkernel whose full functional correctness has been verified to the source code level. However, properly using such highly secure platform to build larger-scale systems is still a major challenge, due to the gap between the platform's mechanisms and assurances and applications' very different functionality and security requirements [5].

Existing security patterns can be used to guide design but their description and security property claims are highly informal [1, 10]. Heyman et al. [9] presented an approach to formally verify security requirements for security patterns using Alloy. This includes "expectations", which describe the expected behavior of the component, and "residual goals". Residual goals describe security concerns that need to be addressed but are not in the scope of the pattern. An example of residual goal is authenticator enforcer pattern requires the userID be unique. Ensuring the uniqueness of the userID is essential but is outside the scope of the pattern. Other prior work also advocates the use of formal methods to verify security properties [10]. Users may define the behavior of a pattern in UML, which is transformed to Promela for verification with SPIN [11]. However, this approach does not consider the security models, mechanisms and assurances provided by the underlying security platform.

Some other efforts [12, 13] in the model-driven community focus on developing platform-specific security profiles, used in application design to facilitate security property analysis and to map models to platform-specific code. We focus on providing reusable capability-specific design fragments for patterns that can be composed during application design.

The link between architecture tactics and code [14] or design patterns [15] has been explored through code mining. One observation is that there is wide variation in how tactics are implemented in different situations [15]. We limit these variabilities by providing capability-specific realizations of patterns, and by focusing on security.

Other research also advocates the importance of the underlying platform to fundamentally improve security [16]. However, they propose new hardware mechanisms to support security, whereas we rely on general purpose hardware and a formally verified microkernel.

III. Proposed Approach

Our approach is shown in Fig.1. First, we collect security patterns from the literature [1, 17, 18, 19] and assemble them into a catalogue. These patterns are analyzed and realized as concrete capability-specific design fragments, in this case for seL4. We verify the properties of these patterns at the design level, to check whether security properties are preserved and what assumptions must be made on the underlying platform. In the design of secure systems, the verified design fragments are composed with each other to build up an application design that satisfies the security requirements. Finally, we verify the composite system at the design level, to provide assurance that the system guarantees the intended properties.

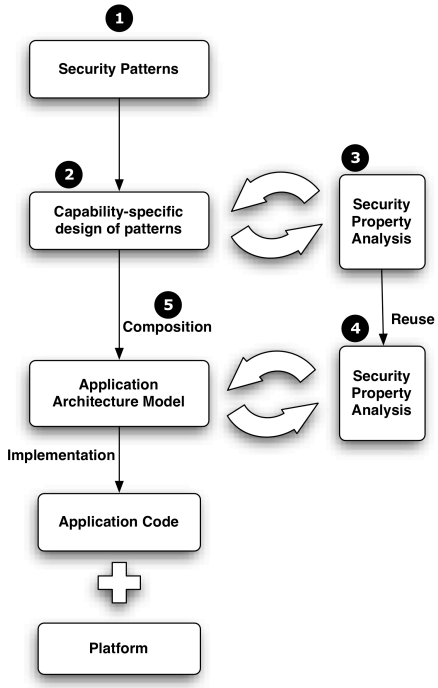

Fig. 1. Overview of our approach.

A. Capability-Specific Design Fragments for Security Patterns

In this step (Step 2 of Fig. 1), we provide design instances for the security patterns, based on our capability-based platform, seL4. The existence of an underlying platform makes this realization possible. We identify and explicitly state the assumptions of the patterns to allow better understanding of the patterns. These assumptions can be discharged by the mechanisms supported by the platform, or carried through as operational constraints. We use SAM, a modeling tool for capability-based systems, to model our realization of patterns as capability-specific design fragments. The security property analysis (Step 3 in Fig. 1) will provide feedback to improve the design realization of the patterns. This can be done by improving the design to address counter-examples identified during the analysis process.

B. Composition of Capability-Specific Design Fragments

Designing a secure system requires the composition of various security patterns (Step 4 of Fig. 1), as each pattern is concerned with different security properties. The first step to composing patterns is selecting appropriate patterns: those that correspond to the security requirements of the application. Two challenges in composing security patterns are to ensure that the patterns do not conflict, and to produce a design that provides the intended security properties.

In order to reason about whether the composed design provides the required security properties, we use our collection of reusable verified platform-specific design fragments for the security patterns. As the capability-specific design fragments for the patterns are verified to provide specific security

properties, we believe that this will aid in providing assurance of the security of the application architecture model. The security property analysis (Step 5 of Fig.1) will not only help in improving the model by pointing out its weaknesses, but also provides assurance about the security of the model.

C. Security Property Analysis

Our designs include the capability-specific design fragments for the patterns and the application architecture model. We analyze security properties at each of these two levels (Step 3 and 5 of Fig.1). This is closely related to the capability-specific design fragments for the patterns (Step 2 in Fig.1), as the analysis performed on these designs will provide pointers to improve the designs.

Different security properties may require different verification techniques. As an example, the absence of dataflow between two components and the absence of data modification are two distinct security properties with different techniques for verifying them. Verification of dataflow [5] be achieved using SPIN, whereas mathematical proof in Hoare Logic that is verified by a theorem prover, Isabelle, may be used to verify unauthorized data modification [22]. Formal verification techniques provide a high level of assurance about the security properties of our designs. Another class of techniques is systematic evaluation, such as scenario-based evaluation [2].

We aim to reuse the verification of the capability-specific designs of the patterns (Step 3 in Fig.1) in order to verify the application architecture model (Step 5 in Fig.1). Full formal verification of the whole application code-base is very expensive and most of the time infeasible. The current limit of such formal verification is around 10,000 lines of code. However, we believe design-level verification of key security properties for large systems is feasible and can be achieved through reuse of verified designs for security patterns.

IV. Expected Contributions

The expected contributions of this research are as follows:

- an overall approach of providing capability-specific designs of security patterns associated with reusable design-verifications and explicit assumptions in relation to the underlying system;
- a collection of such reusable capability-specific designs fragments;
- a pattern composition approach that reuses design-verifications to help provide whole-system verification.

V. Evaluation Plan

There are four key parts to our evaluation. Firstly, we evaluate the capability-specific design fragments for patterns through the verification that will be done for each individual design fragments. This will evaluate whether the design fragments provide the security properties guaranteed. Secondly, the verification of the design fragments will be evaluated through simple case studies based on two criteria, namely *Feasibility* and *Correctness of produced design*. *Correctness of produced design* evaluates whether the design achieved the security property claimed by the verified design fragments. Thirdly, the composition of verified design fragments will be evaluated by performing a security property analysis (i.e. verification) on the produced application design. Finally, the verification of the composition will be evaluated through a real-world case study, Smart Meter in the Advanced Metering Infrastructure, to be validated based on three criteria. These criteria are *Feasibility*, *Correctness of produced design* and *Usability of the pattern composition approach*.

VI. Progress

This research is still in its early stages, starting in March 2012. Currently, we have collected security patterns from the literature and assembled them into a catalogue. The catalogue lists previously-known characteristics of security patterns, together with newly added fields to support this research. These fields, marked (E) for existing and (N) for newly added, are: *Pattern Name* (E), *Alias* (E), *Intent* (E), *Claimed Security Properties* (E), *Source* (E), *Abstraction Level* (E), *Capability-specific Design Fragments* (N), *Security Property Analysis* (N), *Security Properties verified* (N), and *Assumptions* (N).

The new fields are used in different steps of the approach. The *Capability-specific Design Fragments* field stores the capability-specific design fragments for the patterns, which will be used for building the application architecture model through composition (Step 5). *Security Property Analysis* and *Security Property verified* fields store the security property verified and its verification respectively and are reused during the analysis of the application architecture model (Step 6). *Assumptions* field explicitly states the assumptions embedded in the patterns.

We have found no standard template for describing these patterns. This leads to variability in terms of information about pattern, making it hard to collect the same information for different patterns. This agrees with a survey [20] which found that some patterns are under-specified while others are over-specified.

We have started building a collection of capability-specific design fragments. We illustrate this with the *Authorization* security pattern. Assume Alice (granter) wants to authorize Bob (receiver) to have a read-only capability to Carol (target). Alice needs to at least own a Grant capability to Bob and a Read capability to Carol. If that condition is satisfied, Alice can then grant Bob a read capability to Carol. The granting mechanism depends on the capability-based system. In seL4, Alice can mint a new read capability from her set of capabilities to Carol and send an IPC message to Bob's communication endpoint.

We have started building a collection of capability-specific design fragments for security patterns. These will be modeled manually using SAM. Here we illustrate the Security Logger pattern as a capability-specific design. One of the aims of the Secure Logger pattern is to decouple logging functionality from the application. The data must be cryptographically secure, with only authorized users able to view the contents of the log file. Assume Client sends a log command to secureLogger with the data. Upon receiving the log command, secureLogger encrypts the data and sends it with the log command to logManager. logManager will then ask for a new instance of

logger, the actual component that logs the data, from logFactory.

Fig. 2. Secure Logger in SAM.

From Fig. 2, logManager has a capability to logFactory and the newly created logger. The user does not have direct access to the logger instance and the file. We set an assertion in SAM that the user does not have any reference to the file. With that, we assume that the user has no access to the file, unless the user has been authorized to access it.

VII. Work Plan

The next steps are to create a more comprehensive list of capability-specific design fragments for the patterns. Then, the implementation in different capability-based systems of a subset of these will show that implementation of the designs is feasible and that the designs are applicable to other capability-based systems. We will investigate how to compose the security patterns to build secure systems and how to carry out security property analysis to provide feedback to improve the concrete platform-specific design of the patterns. Finally, we plan to investigate whether we can generalize the approach to other capability-based systems.

Acknowledgment

I would like to express my deepest gratitude to my supervisor, Dr. Liming Zhu, for his guidance and support throughout this research.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security," *Urbana*, vol. 51, p. 61801, 1998.

[2] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice.* 3rd ed. Addison-Wesley, 2012.

[3] R. Ortiz, S. Moral-García, S. Moral-Rubio, B. Vela, J. Garzás, and E. Fernández-Medina, "Applicability of security patterns," *On the Move to Meaningful Internet Systems: OTM 2010*, pp. 672–684, 2010.

[4] J. B. Dennis and E. C. Van Horn, "Programming semantics for multiprogrammed computations," *Commun. ACM*, vol. 9, no. 3, pp. 143–155, Mar. 1966.

[5] I. Kuz, L. Zhu, L. Bass, M. Staples, and X. Xu, "An architectural approach for cost effective trustworthy systems," *Working IEEE/IFIP Conference on Software Architecture (WICSA) and the 6th European Conference on Software Architecture (ECSA)*, 2012.

[6] G. Klein et al., "seL4: Formal verification of an OS kernel," *ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 207–220, 2009.

[7] N. Hardy, "The Confused Deputy: (or why capabilities might have been invented)," *ACM SIGOPS Operating Systems Review*, vol. 22, no. 4, pp. 36–38, 1988.

[8] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[9] T. Heyman, R. Scandariato, and W. Joosen, "Reusable formal models for secure software architectures," *Working IEEE/IFIP Conference on Software Architecture (WICSA) and the 6th European Conference on Software Architecture (ECSA)*, 2012.

[10] S. Konrad, B. H. Cheng, L. A. Campbell, and R. Wassermann, "Using security patterns to model and analyze security requirements," *Requirements Engineering for High Assurance Systems (RHAS'03)*, p. 11, 2003.

[11] G. J. Holzmann, "The model checker SPIN," *Software Engineering, IEEE Transactions on*, vol. 23, no. 5, pp. 279 –295, May 1997.

[12] J. Jürjens, "UMLsec: Extending UML for Secure Systems Development," *5th International Conference on the Unified Modeling Language*, London, UK, UK, 2002, pp. 412–425.

[13] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A UML-based modeling language for model-driven security," *the 5th International conference on The Unified Modeling Language*, pp. 426–441, 2002.

[14] M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Cinar, "A tactic-centric approach for automating traceability of quality concerns," in *Proceedings of the 2012 International Conference on Software Engineering*, 2012, pp. 639–649.

[15] M. Mirakhorli, P. Mader, and J. Cleland-Huang, "Variability Points and Design Pattern Usage in Architectural Tactics," *in press*.

[16] P. G. Neumann and R. N. M. Watson, "Capability Revisited: A Holistic Approach to Bottom-to-Top Assurance of Trustworthy Systems," *Fourth Annual Layered Assurance Workshop,* 2010.

[17] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons. Ltd, 2006.

[18] C. Steel, R. Nagappan, and R. Lai, *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management.* Prentice Hall, 2006.

[19] B. Blakley and C. Heath. Security design patterns technical guide, Technical report, Open Group (OG), 2004.

[20] M.-A. Laverdiere, A. Mourad, A. Hanna, and M. Debbabi, "Security Design Patterns: Survey and Evaluation," in *Electrical and Computer Engineering, 2006. CCECE '06. Canadian Conference on*, 2006, pp. 1605 –1608.

[21] T. Leonard, M. Hall-May, and M. Surridge, "Modelling Access Propagation in Dynamic Systems," unpublished.

[22] T. Sewell, S. Winwood, P. Gammie, T. Murray, J. Andronick, and G. Klein, "seL4 enforces integrity," *Interactive Theorem Proving*, pp. 325–340, 2011.