# Towards a Generic Process for Security Pattern Integration

Andreas Fuchs and Sigrid Gürgens and Carsten Rudolph
Fraunhofer Institute for Secure Information Technology
Darmstadt, Germany
{andreas.fuchs,sigrid.guergens,carsten.rudolph}@sit.fraunhofer.de
The original publication is available at www.springerlink.com.

***Abstract*—Interdependencies between different security patterns can influence the properties of a particular pattern when applied in conjunction with other patterns. The resulting properties will often be weaker due to the possibility of new attacks. In this paper we introduce a mechanism that leads towards a generic process for pattern integration. As an example, we use the interesting case in which the proper integration of two patterns provides stronger security properties than the simple unification of the two properties. Formally, this increase in security is achieved by linking parameters of refined versions of the individual properties. The example shows the combination of two different authenticity properties (authenticity of a device based on trusted platform module functionality and authenticity of a user by using SSL). Remarkably, previously proposed combinations of solutions do not satisfy the desired integrated security properties. This indicates that pattern integration requires means that go beyond simple unification. Our pattern integration process presents a first step in this direction.**

## I. Introduction

Complex security properties can often not be realised by one single security mechanism in which case it is necessary to combine several mechanisms. The combination can be easier if security mechanisms are precisely described as so-called security patterns. Nevertheless, the combination is usually far from trivial for several reasons. First, the complex security property can be different (stronger) than the union of the properties of the individual solutions. Furthermore, solutions need to be integrated in the correct way and interrelations between different solutions can also introduce additional attack possibilities.

In this paper we propose a generic process for integrating security patterns. To illustrate this process we use a relatively small example of the integration of two security solutions whose security goal is the combination of a secure channel, namely SSL, with the identification of the end-points of this channel based on TPM attestation.

Remarkably, some solutions proposed for the integration of the two patterns used in the example do not satisfy the expected properties (see [1]). Therefore we argue that the integration of security patterns can not be achieved by simply merging the patterns and the respective security properties. The information provided by the separate patterns does not include all information necessary to achieve the desired security property for the integration. Hence the integration of security patterns is actually a new security pattern by itself, describing a new security solution that has to be validated or verified independently from the validation of the individual patterns. In [2] we have applied our Security Modeling Framework (SeMF) to prove that a more sophisticated integration actually provides the desired property. In this paper however we focus on the integration process itself whose specific characteristics lie in property refinement.

In the next section we will briefly recapitulate the notion of security patterns. In Section III a step-by-step process for pattern integration is introduced. Then Section IV gives a short summary of our Security Modeling Framework and introduces the formal models of the example patterns. Then Section V demonstrates the integration process using the example of TPM attestation and SSL.

## II. Security patterns

The notion of security patterns is applied to a very broad field in security research and development. This can range from best practices recommendations expressed in terms of natural language over example code-blocks to (semi-)formal descriptions of security solutions, sometimes even in machine readable formats.

One example for the latter category are so-called Security and Dependability (S&D) Patterns developed within the SERENITY project [3]. A key feature of these S&D Patterns is their machine readable format. Further they provide an application programming interface (API) such that the usage of a pattern is basically reduced to implementing a program against its API. Each pattern includes a notion of pre-requisites such that a run-time framework can choose between different suitable implementations of a pattern and link against a respective executable component.

Additionally, each S&D Pattern contains a formal representation of the provided security and dependability properties. It refers to the APIs' actions and roles to unambiguously defines its security properties.

S&D Patterns also represent the integration of different solutions in order to provide a combined or even stronger property. Within SERENITY this special type of S&D Patterns is called

Part of this work was accomplished within the project SERENITY 27587 funded by the European Commission.

Integration Scheme. It defines a certain functional integration of existing S&D patterns. Consequently, the description of an Integration Scheme also contains a formal representation of security and dependability properties.

Whilst the integration process introduced in this paper is in principle applicable to all the abovementioned types of security patterns, the example will refer to the formal property specification of the Security Modelling Framework (SeMF) that was also used within SERENITY's S&D Pattern notion.

## III. Integration of security patterns

We propose the following step-by-step approach for the integration of security patterns. The goal of this approach is to set a proper basis for the verification of the integrated solution as the final step before implementation and deployment. The main elements in this process are security properties of the individual patterns and their combination, functional specification of the integration, and assumptions on the system necessary for satisfaction of the properties. We assume that at the beginning of the integration, formal specifications of security properties and of the actions of both patterns are available.

**Unification of the action sets of the individual patterns** In principle we perform the union of the two sets of actions. However, we have to ensure that names of actions and parameters are unambiguous, i.e. distinguish those actions and parameters, respectively, that are equal but shall be different. Further, we need to identify matching parts of the models, i.e. rename those action and parameter names that are different but shall be identical. This task strongly depends on the actual description of the patterns.

**Refinement and linking of properties** The properties of the individual patterns usually only refer to all actions and parameters relevant for each particular property. If this is not sufficient for expressing the targeted combined property they have to be refined in order to include all necessary information. Combining properties can be achieved by linking particular parameters that are then included in the refined properties.

**Revision of the assumptions for the individual patterns** There are cases in which combinations of patterns result in weaker security just because some of the assumptions that hold for the system being based on one pattern are violated by the other pattern. Therefore, each assumption has to be checked in the context of the combined system.

**Functional integration and additional assumptions** The pattern integration on a functional level consists in a refinement of the unified actions which is rooted in the refinement of the properties. This integration may lead to additional assumptions that span over both integrated systems.

**Verification/validation of the integrated pattern** As will be discussed in the example below, validation or verification of the integrated solution is essential. Very often, intuitive solutions do not satisfy the expected security properties.

**Implementation** In the cases in which pattern integration includes more than just composition, it has to expected that changes to the pattern implementation are required for the integrated pattern.

## IV. Building blocks for the example

In order to demonstrate the approach introduced in the previous section we choose the Security Modelling Framework (SeMF) [4] for the system and property descriptions. To this end, in this section we first give a very brief summary of those concepts of SeMF that are relevant for this paper and then use these concepts to model the two example systems. Note that for the example we use abstract versions of the SERENITY XML representations of the S&D patterns.

### A. The Security Modeling Framework SeMF

The behaviour $B$ of a discrete system can be formally described by the set of its possible sequences of actions (traces). Therefore $B \subseteq \Sigma^*$ holds where $\Sigma$ (called the alphabet) is the set of all actions of the system, $\Sigma^*$ is the set of all finite sequences (called words) of elements of $\Sigma$, including the empty sequence denoted by $\varepsilon$, and subsets of $\Sigma^*$ are called formal languages. Words can be composed: if $u$ and $v$ are words, then $uv$ is also a word. For a word $x \in \Sigma^*$, we denote the set of actions of $x$ by $alph(x)$. For more details on the theory of formal languages we refer the reader to [5].

We further extend the system specification by two components: *agents' initial knowledge* about the global system behaviour and *agents' view*. The initial knowledge $W_P \subseteq \Sigma^*$ of agent $P$ about the system consists of all traces $P$ initially considers possible, i.e. all traces that do not violate any of $P$'s assumptions about the system. Every trace that is not explicitly forbidden can happen in the system. Further, in a running system $P$ can learn from actions that have occurred. Satisfaction of security properties obviously also depends on what agents are able to learn. After a sequence of actions $\omega \in B$ has happened, every agent $P$ can use its *local view* $\lambda_P$ of $\omega$ to determine the sequences of actions it considers to have possibly happened. For a sequence of actions $\omega \in B$ and agent $P \in \mathbb{P}$ (where $\mathbb{P}$ denotes the set of all agents), $\lambda_P^{-1}(\lambda_P(\omega)) \subseteq \Sigma^*$ is the set of all sequences that look exactly the same from $P$'s local view after $\omega$ has happened. Depending on its knowledge about the system $B$, underlying security mechanisms and system assumptions, $P$ does not consider all sequences in $\lambda_P^{-1}(\lambda_P(\omega))$ possible. Thus it can use its knowledge to reduce this set: $\lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$ describes all sequences of actions $P$ considers to have possibly happened when $\omega$ has happened.

Security properties can now be defined in terms of the agents' initial knowledge and local views. For more details we refer the reader to [4].

Our definition of authenticity (see [6]) uses the above described concepts and essentially states that a set of actions $\Gamma \subseteq \Sigma$ is authentic for agent $P$ if in all sequences that $P$

considers possible after a sequence of actions $\omega$ has happened, some time in the past an action of $\Gamma$ must have happened.

*Definition 1:* A set of actions $\Gamma \subseteq \Sigma$ is authentic for $P \in \mathbb{P}$ after a sequence of actions $\omega \in B$ with respect to $W_P$ if $alph(x) \cap \Gamma \neq \emptyset$ for all $x \in \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$.

In the following we present briefly the modelling of the two example systems. Here we concentrate on instantiations of the above defined property authenticity provided by these systems and disregard other properties like freshness. This facilitates the understanding of our approach, our integration process also serves to address other properties desired for the result of an integration.

### B. User Authentication

The scheme for User Authentication we use is a simplified version of an authentic channel establishment as it is provided by SSL (see e.g. [7]). We assume that the client has some means to authenticate herself during the establishment of a channel, e.g. by using a smart card with a secure private key and a certificate trusted by the server.

The assumption is that once the SSL-channel is established with the channel credentials being authenticated, this channel itself will provide the necessary authenticity for the subsequent communication. This of course does not consider attacks against established channels, but those are off the focus of this paper.

For the simplified formal model of this system, we use two clients $C_1$, $C_2$, one server $S$, and an infinite number of channels $ch_j$. We define the following actions for the abstract SSL system $B_{ssl}$:

*ssl-init*$(C_i, ch_j(C_i, S))$ This action models the initiation of the SSL handshake with server $S$ by one of the clients $C_i$ on channel $ch_j$. The first parameter $C_i$ of channel $ch_j$ indicates that channel establishment is initiated by $C_i$, and parameter $S$ indicates that the channel shall be established with server $S$.

*ssl-rec*$(S, ch_j(S, C_i))$ This models the completion of the SSL handshake by the server which establishes channel $ch_j$. The first parameter $S$ of the channel denotes that server $S$ performs this action, the second parameter denotes that $S$ considers the channel to be initiated by client $C_i$.

Since only the establishment of the channel is important here we reduce our SSL model to these two actions. Although this presents a considerable abstraction of the complex nature of the SSL session key establishment it is sufficiently detailed for our purposes.

The property this system provides and that we want to address in this paper is that each time the server performs action ***ssl-rec*** for a channel $ch_j(S, C_i)$ which it assumes to have been initiated by client $C_i$, this client $C_i$ indeed authentically initiated the handshake. In order to formally specify this property according to Definition 1, we define $\Gamma_{ssl}(C_i)$ to denote all actions *ssl-init* that contain a parameter $C_i$. Now authenticity of the client for the server can be specified as follows:

*Property 1:* For all $\omega$ in $B_{ssl}$ holds if the server $S$ has performed an action *ssl-rec* in $\omega$ with client parameter $C_i$, then $\Gamma_{ssl}(C_i)$ is authentic for $S$.

As already mentioned before, the property actually provided by an SSL channel is stronger since it additionally provides the freshness and confidentiality of messages (see [4] for the respective formal definitions). However, for the purpose of this paper it is sufficient to focus on the formal definition of authenticity alone.

The above property can be formally proven under the following assumptions that actually formalize the assumptions for an SSL channel we discussed at the beginning of this section.

*Assumption 1:* Each time the server performs an SSL handshake with action *ssl-rec*$(S, ch_j(S, C_i))$, in all sequences it considers to have possibly happened the same client $C_i$ has initiated the handshake on the same channel $ch_j$ by performing *ssl-init*$(C_i, ch_j(C_i, S))$.

*Assumption 2:* For the two SSL actions we can also assume that a channel is only established once. This is justified by the fact that both communication parties involved in an SSL-communication influence the session secret, and even if only one (the server $S$) uses a reliable random number generator, the session secret will be virtually unique to this communication session.

### C. Device Identification

The solution that we choose for device identification is based on trusted computing technology as specified by the Trusted Computing Group [8]. The Trusted Platform Module (TPM) can be used to attest the integrity of software running on the platform it is integrated in. For this the software is measured (hashed) and the resulting value is stored in so-called Platform Configuration Registers (PCR) which are only accessible by the TPM. Then the TPM_Quote command instructs the TPM to calculate a signature over these PCR values. By associating the signature key with the TPM, the result of the TPM_Quote can be used to identify the platform. This description omits many details of the very complex process, for more information see [9].

In order to model the system $B_{att}$, we use a verifier $V$ and two devices $d_1, d_2$. We then define the following actions:

*att-gen*$(d_k, quote(d_k))$ This action models the generation of $quote(d_k)$ which denotes the TPM's signature on the PCR values, using device $d_k$.

*att-send*$(d_k, V, quote(d_k))$ models the device $d_k$ sending the quote message to the verifier $V$.

*att-rec*$(V, quote(d_k))$ models the reception and verification of the attestation message by the verifier.

The property that this system provides is that each time the verifier performs action *att-rec* for a $quote(d_k)$ that it assumes to have been generated on device $d_k$, this device indeed authentically generated the quote message (again we disregard the freshness additionally provided by a TPM_Quote). Analogously to the previous section, we define $\Gamma_{att}(d_k)$ to denote all actions *att-gen* that contain a parameter $d_k$. Now

authenticity of the device for the verifier can be specified as follows:

*Property 2:* For all $\omega$ in $B_{att}$ holds if verifier $V$ has performed an action *att-rec* in $\omega$ with device parameter $d_k$, then $\Gamma_{att}(d_k)$ is authentic for $V$.

The assumptions that can be reasonably made for the system $B_{att}$ and that can be used to prove the above property are as follows:

*Assumption 3:* We assume that the attestation actions provide authenticity of the device for the verifier. Each time the verifier receives an attestation message $quote(d_k)$ presumably generated by device $d_k$, in all sequences it considers to have possibly happened, indeed this device generated the quote message by performing *att-gen*$(d_k, quote(d_k))$.

## V. Integrating the example patterns

In this section we demonstrate our step-by-step process for pattern integration introduced in Section III. We apply it to the two example patterns represented in the above formal models, however it can also be used in conjunction with other methods for system and property modelling.

In the following, we use italic font for the two individual systems and their actions while using italic boldface font for the unified system and actions.

### A. Unification of the action sets of the individual patterns

The integration of the two patterns shall result in one system that provides both an SSL channel and a quote message for the server. The first step of the integration process is therefore to unify the two sets of actions. In general, this includes to revise action names and parameters in order to achieve unique ones. This may include to rename actions and parameters (e.g. if in both systems an action $send$ with a parameter $sender$ is used but in the unified system we need two different actions in which messages are sent by different senders). In our particular case we conclude that for the unified system we keep the action names but change the parameter $V$ to $S$, since we want the server both to use the SSL channel and to receive and verify the quote message.

### B. Refinement and linking of properties

The system resulting from the integration of the above described patterns shall meet the following security requirement: Messages of a client to the server shall be authentic for the server, and at the same time the server shall be able to identify the device the message was sent from. This is not described by simply unifying the two properties 1 and 2. In order for example to identify the device an ***ssl-init*** action was performed on, the respective parameter has to be included in the action. The next step is therefore refining the properties. To this end, we define the following two sets:

*Definition 2:* We define $\mathbf{\Gamma_{ssl}}(C_i, d_k)$ to denote all actions ***ssl-init*** that contain parameters $C_i$ and $d_k$, and $\mathbf{\Gamma_{att}}(C_j, d_l)$ to denote all actions ***att-gen*** that contain parameters $C_j$ and $d_l$.

This results in the following refined properties:

*Property 3:* For all $\omega$ in $\mathbf{B_{ssl}}$ holds if the server $S$ has performed an action ***ssl-rec*** in $\omega$ with client parameter $C_i$, then $\mathbf{\Gamma_{ssl}}(C_i, d_k)$ is authentic for $S$.

*Property 4:* For all $\omega$ in $\mathbf{B_{att}}$ holds if the server $S$ has performed an action ***att-rec*** in $\omega$ with device parameter $d_l$, then $\mathbf{\Gamma_{att}}(C_j, d_l)$ is authentic for $S$.

Note that property 3 still just specifies the authenticity of the client for the server with respect to the SSL actions, while property 4 only refers to the device's authenticity for the server with respect to the attestation actions. So in order to link these properties and to formalize the property we actually want the unified system to provide, we need the further requirement that clients and devices in the sets $\mathbf{\Gamma_{ssl}}(C_i, d_k)$ and $\mathbf{\Gamma_{att}}(C_j, d_l)$ are equal. This results in the final formalization of the property:

*Property 5:* For all $\omega$ in $\mathbf{B_{ssl}}$ holds if the server $S$ has performed an action ***ssl-rec*** with client parameter $C_i$ and an action ***att-rec*** with device parameter $d_l$ in $\omega$, then $\mathbf{\Gamma_{ssl}}(C_i, d_k)$ and $\mathbf{\Gamma_{att}}(C_j, d_l)$ are authentic for $S$ and $C_i = C_j$ and $d_k = d_l$.

### C. Revision of the assumptions for the individual patterns

The next step of our integration process is to revise the assumptions we listed for the individual patterns with respect to their validity for the integrated pattern system.

Assumption 1 It seems reasonable to assume that in combining the two patterns no SSL keying information is revealed by the TPM attestation process. Thus Assumption 1 is not violated. Note that this assumption does not refer to the device that the channel was initiated from.

Assumption 2 Further, we can still assume that a channel is only established once. Thus Assumption 2 still holds as well.

Assumption 3 Adding the two SSL actions to the attestation model does not invalidate Assumption 3. We may still assume that the attestation actions provide authenticity of the device for the server. Note that the assumption does not include any statement about the client that initiated the channel.

It is important to note that in some cases of pattern integration specific features of one pattern can invalidate the assumptions of the other pattern thus violating the security of the overall system. Hence revising the assumptions is a crucial step of our integration process.

### D. Functional integration and additional assumptions

As we have seen in the unification step, the unified system will contain all SSL actions and all attestation actions, with two clients, two devices and one server as acting entities. However, the security property which we derived from the refinement and link step requires the actions to be modified as we need to include the device in the SSL actions and the SSL channel in the attestation actions. This is a task for a security expert since there are various possibilities to do this. In [2] we have discussed two different ways of extending actions by parameters. In both, the device $d_k$ is included in the ***ssl-init*** action, but they differ in the

way the SSL channel is included in the attestation actions. However, only one of them indeed provides property 5 (see below for more details). The respective functional integration leads to the following actions: ***ssl-init***$(C_i, ch_j(C_i, S), d_k)$, ***ssl-rec***$(S, ch_j(S, C_i))$, ***att-gen***$(d_k, quote(d_k, ch_j(C_i, S)))$, ***att-send***$(C_i, S, quote(d_k, ch_j(C_i, S)), ch_j(C_i, S))$, ***att-rec***$(S, quote(d_k, ch_j(C_i, S)), ch_j(C_i, S))$.

Based on the assumption that SSL keying information is not revealed by the TPM attestation process, we can extend the property of the SSL channel to all messages that are sent on this channel.

*Assumption 4:* By the nature of an SSL channel, if some message is received on it, there must be a respective send action on this channel. In analogy to Assumption 1 we further assume that this send action is authentic. Hence we assume that whenever the server receives an attestation message $quote(d_k, ch_j(C_i, S))$ by $d_k$, the message must have been sent on this channel by the device $d_k$.

*Assumption 5:* SSL channels can only be used after a successful handshake. Thus, whenever a client sends the attestation message on channel $ch_j(C_i, S)$, the server has established this channel before.

The most important assumption that is necessary for the proof of the property (see [2]) but is not adequate for the system in which the quote message is sent using the SSL channel is the following one:

*Assumption 6:* The channel that is contained in the quote message is initiated on the device that generates this message, that is, whenever ***att-gen***$(d_k, quote(d_k, ch_j(C_i, S)))$ has happened, ***ssl-init***$(C_i, ch_j(C_i, S), d_k)$ must have happened before.

### E. Verification/validation of the integrated pattern

The next step of the proposed integration process is the verification of the system derived from the pattern integration. This is an important step since not necessarily all functional integrations lead to systems providing the desired property. In [2] we have discussed two different ways to integrate the example patterns. In the first one the channel is added as additional parameter to the attestation send and receive actions. This models a previously proposed approach for integrating TPM attestation with secure channel establishment which suggests to send the quote message using the SSL channel (see [1]). This approach fails to provide the desired property. A security evaluation shows that the device establishing the handshake does not necessarily have to be the device that generates the quote message. The functional integration described in the previous section however does lead to a system that provides property 5, as proven in [2]. The next section discusses practical realisations.

The proof makes heavily use of the revised assumptions, hence they are essential for the integration process. Other verification methods include model checking (see for example [10], [11]). These approaches essentially model the agents of the system and their behaviour (based on some underlying formal method) and the behaviour of some (external or internal) attacker, which implicitly includes the assumptions that are considered valid for the system.

### F. Implementation

The verification process from [2] reveals that the key feature for the integrated pattern to work is the enforcement of Assumption 6. A possible realisation could include the reservation of a certain PCR for channel information, and deny direct user access to this PCR by means of separating the channel establishment from the user controlled environment. Proof of this separation might derive directly from the attested software behaviour or an access restriction to certain localities for the PCR in question. For further information we refer the reader to [9].

## VI. Conclusions

In this paper we have introduced a step-by-step process for the integration of security patterns. We have demonstrated our approach using the integration of a pattern modelling an SSL channel with a pattern modelling attestation based on TPM functionality as an example. The specific characteristic of our approach lies in property refinement: The formal specification of the properties holding for the individual patterns are refined in order to derive the property that shall be provided by the integrated pattern. Our integration process still requires expert knowledge, in particular for the functional integration of the two patterns. Thus future work will include the development of methods that facilitate the functional integration with respect to the revised properties.

## References


[1] K. Goldman, R. Perez, and R. Sailer, "Linking remote attestation to secure tunnel endpoints," in *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*. New York, NY, USA: ACM, 2006, pp. 21–24.

[2] A. Fuchs, S. Gürgens, and C. Rudolph, "On the Security Validation of Integrated Security Solutions," in *Emerging Challenges for Security, Privacy and Trust - 24th IFIP TC 11 International Information Security Conference, SEC 2009, Cyprus*, ser. IFIP Advances in Information and Communication Technology, D. Gritzalis and J. Lopez, Eds., vol. 297. Springer, 2009.

[3] "Serenity, system engineering for security & dependability," www.serenity-project.org, 2006.

[4] S. Gürgens, P. Ochsenschläger, and C. Rudolph, "On a formal framework for security properties," *International Computer Standards & Interface Journal (CSI), Special issue on formal methods, techniques and tools for secure and reliable applications*, vol. 27, no. 5, pp. 457–466, June 2005.

[5] S. Eilenberg, *Automata, Languages and Machines*. Academic Press, New York, 1974.

[6] S. Gürgens, P. Ochsenschläger, and C. Rudolph, "Authenticity and provability - a formal framework," in *Infrastructure Security Conference InfraSec 2002*, ser. Lecture Notes in Computer Science, vol. 2437. Springer Verlag, 2002, pp. 227–245.

[7] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol," Netscape Communications Corp., Nov 1996.

[8] T. C. Group, "TCG TPM Specification 1.2 revision 103," www.trustedcomputing.org, 2006.

[9] Trusted Computing Group, *TPM Main - Part 1 Design Principals*, July 2007, specification Version 1.2, Level 2 Revision 103.

[10] B. Roscoe, P. Ryan, S. Schneider, M. Goldsmith, and G. Lowe, *The modelling and Analysis of Security Protocols*. Addison Wesley, 2000.

[11] S. Gürgens, C. Rudolph, D. Scheuermann, M. Atts, and R. Plaga, "Security evaluation of scenarios based on the TCG's TPM specification," in *Computer Security - ESORICS 2007*, ser. Lecture Notes in Computer Science, J. Biskup and J. Lopez, Eds., vol. 4734. Springer Verlag, 2007.