Towards a Rigorous IT Security Engineering Discipline

Antonio Maña.
Computer Science Department.
University of Málaga. Spain.
amg@lcc.uma.es

Abstract. Current practices for developing secure systems are still closer to art than to an engineering discipline. Security is still treated too frequently as an add-on and is therefore not integrated into IT systems development practices and tools. Experienced security artisans continue to be the key for achieving acceptable levels of security in IT systems. In this situation we could safely state that security engineering is a hype. In fact, the term security engineering has been used to denote partial approaches that cover only small parts of the processes that are required in order to create a secure system, like modelling, verification, programming, etc. Moreover, one finds in the literature that the main books about security engineering describe threat-based engineering approaches. The main drawbacks of these approaches is that they fail to provide a reasonable level of support for systematic engineering since the identification, characterization and specification of the requirements (frequently based on avoiding threats) as well as the selection of appropriate mechanisms and countermeasures depends on the experience of the engineers. Consequently, these approaches are inherently opposed to what an engineering discipline should be and represent only minor improvements over the security art. This paper discusses this vision and advocates a change of paradigm based on rigour and precision. In particular, the paper presents an approach based on the precise and formal specification of both security requirements and security solutions as the basis for the engineering of secure systems and discusses the role of contracts in security engineering.

Introduction

Current practices for developing secure systems are still closer to art than to an engineering discipline. IT Security is treated as an add-on and is therefore not integrated into software development practices and tools. With more and more aspects of our lives being affected by computing systems and with the inevitable trend towards IT systems in which humans are immersed, the aspects of assurance and certification become crucial. However, we still depend on the knowledge of experienced security artisans in order to predict the threats that the system will have to withstand and to prevent them, achieving acceptable levels of security, and this art-like approach does not allow us to have guarantees and proofs of the security of those systems. Moreover, the growing complexity of these systems is becoming larger than the capacity of humans to understand and secure an IT system.

The existence of an IT security engineering discipline could not only change that situation of lack of guarantees but also improve the security of IT systems by allowing them to be prepared to operate in unforeseen contexts and to be able to provide security to the extremely complex and dynamic IT systems that are coming in the near future.

Traditionally, the term security engineering has been used to denote partial approaches that cover only small parts of the processes that are required in order to create a secure system, like modelling, verification, programming, etc. Several approaches and research strands have tried to address this situation in order to introduce rigour and engineering approaches in the treatment of security aspects in information systems, mainly focusing on the development phases. Paradoxically, many of the best known initiatives for enhancing the security of computer systems have been based on guidelines, recommendations, best practices, certification and similar approaches lacking the necessary rigour and precision that one would expect when dealing with "engineering" and even more with "security engineering". Some examples are: Common criteria [1], traditional security patterns [2], Sarbanes-Oxley [3] and HIPA [4] Acts, etc.

Even in the cases of approaches that are closer to a methodology and have achieved a certain level of maturity, the key concepts and workflows are highly influenced by the way security has been treated by the security artisans. Therefore, one finds in the literature that the main books about security engineering describe threat-based engineering approaches. The main drawbacks of these approaches is that they fail to provide a reasonable support for systematic engineering since the identification, characterization and specification of the threats as well as the selection of appropriate mechanisms and countermeasures depends on the experience of the engineers. Consequently, these approaches represent only minor improvements over the security art. Yet, they have been used for some time with uneven popularity and results.

In this paper we discuss this aspect and advocate a change of paradigm based on the precise and formal specification of security solutions and security properties and the use of these formally verified properties as the basis for the expression of requirements and the engineering of secure systems. The paper shows that threat-based security engineering (i) is the origin of penetrate-and-patch situation; (ii) does not result in specifications that can survive evolution of systems and context; and (iii) does not capture the requirements, but the mechanisms selected to achieve them, which results in poor maintainability of the systems produced. The paper also introduces two pillars to solve the situation. On the one hand, we present the SERENITY model of secure and dependable systems and show how it supports the creation of secure and dependable systems for these new computing paradigms. On the other hand we discuss the concept of contract and the role it plays in ensuring a rigorous treatment of security. Finally some conclusions will be drawn and a proposal for establishing a renewed security engineering discipline will be advocated.

Threat-based security engineering considered harmful

Today, the current trend towards distributed and open systems, has revealed the important limitations of threat-based security engineering. The main problems that the new computing paradigms introduce are the high levels of heterogeneity, dynamism and autonomy, as well as the large scale and high complexity. The result is that engineers have to deal with runtime situations that are unpredictable at design time, which are very difficult if not impossible to secure using threat-based or attack-based approaches. In particular, threat-based security engineering creates systems that are very context-dependent, and therefore, fail to address the needs of the future open and distributed systems paradigms. This approach was acceptable and even successful in the times of closed and static systems, running on homogeneous and well-known platforms, because the runtime context was more or less predictable and therefore a security expert could possibly identify the threats that the IT system under development had to withstand and could find appropriate solutions to avoid them. However, once we face the challenges introduced by open systems, which will be built

from components at runtime; dynamic evolution, which increases unpredictability and introduces the need for security solutions to adapt to the system evolution; and extremely heterogeneous platforms, which essentially makes impossible to rely on infrastructure security, we are obliged to admit that we have reached the end of the threat-based approaches and we need to use a new approach that can deal with the aforementioned challenges.

Some of the main reasons why we consider that threat-based security engineering is not a valid approach to provide acceptable levels of security for the future IT systems are:

- **Lack of dynamism and support for evolution.** Threats change when systems change even if the protection goals remain the same. Therefore, threat-based systems require full reengineering to cope with any context change. Moreover, it becomes very difficult to identify the changes required in the system, as a result of a change in the context.
- **Poor traceability.** Threats are difficult to trace to protection goals or security requirements. This is precisely the reason why systems engineered following a threat-based approach tend to be extremely difficult to maintain and to adapt to new context conditions.
- **Expression of user requirements is lacking, not precise or context-dependent.** This is the one of the main weakness and is related to some of the others, and in particular to the previous two. When the input to design process is expressed in terms of threats and attacks, the user requirements become lost (or at least hidden) and consequently the system under development becomes weaker in terms of maintainability, traceability and resilience to evolution. Moreover, the longevity and stability of the specification of the system are reduced.
- **The complete set of threats is impossible to identify.** The impossibility to guarantee the completeness of the set of identified threats, has been traditionally a major weakness of the threat-based engineering approaches. Given the unpredictability that characterizes the future computing paradigms, it will become impossible to identify not even a small fraction of the potential threats, even for the most experienced and visionary security experts.
- **The threat-based approach is closely related to the penetrate-and-patch vicious cycle.** As it is impossible to predict at development time all attacks and threats that the system will face while in operation, the new threats and vulnerabilities that are discovered during the system operation require the system to be patched, which inevitably leads to a degradation of the quality and the introduction of new vulnerabilities.
- **Assurance and certification can not be based on threats.** This aspect is specially important in security critical systems, but also in all socio-technical systems. Stating (or even proving) that a system can withstand a threat does not say much about what can be guaranteed about the system.
- **Poor user communication.** Tell your customers that their system will be secure because you will implement state-of-the-art protection against cross-site scripting and avoid eavesdropping by using authenticated TLS connections. They may be happy to hear some fashionable buzzwords, but it is most likely that they do not understand if what you are providing solves their problem. Some advances based on the specification of abuse and misuse cases can help, but still they lack precision and do not provide the guarantees that your customer would need.

In consequence, we believe that the threat-based security engineering era has come to an end and it is time to find an appropriate replacement. The next two sections will present two

pillars of this replacement.

The SERENITY model for IT security engineering

The goal of this section is to facilitate the understanding of the SERENITY approach to secure IT systems engineering. However, due to its size and complexity, it is out of the scope of this paper to provide a complete view of SERENITY. The interested reader is referred to reference [5] for a comprehensive description of the whole system. The SERENITY project has produced two main results related to our discussion:

A model of secure system engineering that has two main features:

- *Separation between security solution development and application development.* In this way security experts develop, analyse and characterize security solutions and components while application developers and programmers concentrate on developing applications without the need to implement security solutions (which is normally a recipe for disaster given the lack of security expertise of average application developers). All application developers have to know about security is to express their security requirements and to include references to the security solutions (which are precisely and semantically described using a series of modelling artefacts called S&D Artefacts, as we will show below) that they need, in their application code. The more abstract the level of the artefact selected at development-time is, the more the flexibility for the selection of the specific implementation of the solution to use at runtime.
- *Run-time support* enabling the dynamic selection, adaptation and substitution of security solutions at run-time. Additionally, at run-time the selected solutions are monitored to ensure their correct operation. To achieve this, it is necessary to have a framework supporting the management of a library of available security solutions and the constant evolution of applications based on such solutions, taking into account the context in which they are applied. The SERENITY Runtime Framework (SRF) is the tool that provides this support. It is able to select the most appropriate S&D solution among the ones available, based on the end-user requirements and the actual runtime context.

A series of modelling artefacts used to capture knowledge about different security aspects, such as properties, services, and solutions.

In SERENITY, the main pillar of building secure and dependable solutions are the enhanced concept of S&D Properties and S&D Pattern. An **S&D Property** is a precise specification of one or more security goals that can be applied to any computational object. It is important to highlight that, opposed to the common belief, the number of security properties (i.e. confidentiality, integrity, authenticity, non-repudiation...) is not limited. We could express this more precisely saying that, even if we accept that the number of "abstract properties" could be considered limited, the number of interpretations (semantics) of these abstract properties is unbounded. The SERENITY model is based on the assumption that there will be different definitions (corresponding to interpretations) for the security properties. There are many reasons supporting this assumption: for instance, legislation, cultural etc. Under this assumption, and in order to guarantee the interoperability of systems based on those properties and of solutions fulfilling the properties, we need to be able to precisely express the interpretation (semantics) of each property and to exploit the relations between different properties. For a more complete description of the concept of S&D Property, the reader is referred to references [5] and [6].

An S&D Pattern captures security expertise in a way that, in our view, is more appropriate than other related concepts because it focuses on precision and well-defined semantics. Furthermore, because secure interoperability is an essential requisite for the widespread adoption of the SERENITY model, trust mechanisms are a central aspect of S&D Patterns. SERENITY S&D Patterns include a precise functional description of the mechanisms they represent, references to the S&D properties provided, constraints about the context that is required for deployment, and specifications of how to adapt and monitor the mechanisms, as well as trust mechanisms designed to (i) guarantee the origin and integrity of the descriptions contained in the different SERENITY artefacts; and (ii) support the evolution of artefacts and mechanisms and the maintenance of SERENITY systems.

The concept of S&D Pattern has been materialized in a series of modelling elements that we call S&D Artefacts. Abstract S&D solutions are represented by three main artefacts: S&D Classes, S&D Patterns and S&D Implementations. A fourth element, called Executable Component is part of the SERENITY artefacts, but in this case it is not a modelling artefact but an operational one. The above artefacts are briefly described as follows:

- **S&D Patterns** represent abstract S&D solutions. These solutions are well-defined mechanisms that provide one or more S&D Properties. The popular SSL protocol is an example of an S&D solution that can be represented as an S&D Pattern. One important aspect of the solutions represented as S&D Patterns is that they can be statically analysed (e.g. using verification tools based on formal methods). However, the limitations of the static analysis and the assumption that perfect security is not achievable, introduce the need to support the dynamic validation of the described solutions by means of monitoring mechanisms.
- **S&D Classes** represent S&D services (abstractions of a set of S&D Patterns characterized for providing the same S&D Properties and being compatible with a common interface). An example of an S&D Pattern Class could be a *Confidential Communication Class*, which could define an interface including for instance, an abstract method *SendConfidential(Data, Recipient)*. The purpose of introducing this artefact is to facilitate the dynamic substitution of the S&D solutions at runtime. This approach allows us to create an application bound to an S&D Class. Then, all S&D Patterns (and their respective S&D Implementations) belonging to an S&D Class will be selectable by the framework at runtime to serve that application.
- **S&D Implementations** represent operational S&D solutions, which are in turn called Executable Components. It is important to note that the expression "operational solutions" refers here to any final solution (e.g. component, web service, library, etc.) that has been implemented and tested. These solutions are made accessible to applications thanks to the SERENITY Runtime Framework (SRF). The description of either a specific dynamic library providing encryption services or a web service providing time-stamping services, are examples of S&D Implementations. S&D Implementations capture implementation-specific features, such as performance, platform, programming language or any other feature not fixed at pattern's level. The OpenSSL implementation of the SSL protocol ca be described using an S&D Implementation.

These new concepts can be useful in two different ways: at design/deployment time and at run time. In the first case, we must consider that today's large applications are built by integrating solutions from different sources and at different levels of abstraction. These applications face the existence of threats and errors that may require us to perform maintenance on them. Readers can find details on the implementation of SERENITY applications and S&D solutions in references [7] and [8]. By using the SERENITY approach

we allow that this maintenance is performed with a minimum effort, even automatically and seamlessly in some cases. In the second case (i.e., during runtime) S&D Patterns are used in order to support automated adaptation of the S&D solutions to the changing context conditions.

The role of contracts in a rigorous security engineering

Informally speaking we could define a contract as an agreement between two or more parties that establishes obligations for these parties and guarantees about those obligations. More precisely, the *BusinessDictionary.com* definition is a "Voluntary, deliberate, and legally enforceable (binding) agreement between two or more competent parties." The main difference between the two is that we do not necessarily assume that a contract has to have any legal meaning (although we do not exclude that possibility). Contracts can be used in IT security for different purposes. Among these, we highlight the following:

- **Contracts as means for agreeing on security aspects.** In this case the contract establishes the terms (e.g. mechanisms, guarantees, referees and trusted third parties, etc.) that will be used in an interaction between two or more parties. A well-known example of this is constituted by SLAs (service-level agreements) that establish aspects related to the QoS (Quality of Service) like bandwidth, uptime, throughput, etc. Also in this category we find expression of the "terms of use".
- **Contracts as specifications.** A contract can be used to specify aspects of the operation of an entity. For instance, it can be used to specify the means by which the entity ensures the confidentiality of the data processed in an application in cloud computing or service-oriented computing. Another interesting case in this category is that of software contracts, which have been used in component-based development and especially for COTS (components-off-the-shelf). The same concept has recently been applied to the field of secure coding. In fact, some mature development strategies like PCC (proof carrying code) are closely related to this. In PCC, executable code comes with proofs that demonstrate adherence to a contract. These proofs can be verified by the runtime environment prior to code execution.
- **Contracts as guarantees.** A contract can be used to state guarantees about the operation of an entity. For instance, it can be used to guarantee that an economic compensation will be available to the user should the confidentiality of the data processed in an application in cloud computing or service-oriented computing be broken.
- **Contracts as disclaimers.** The idea in this case is to make the user aware of the risks that the software introduce and to declare the limitations of the guarantees of a provider.

From the previous descriptions one can easily understand that the concept of contract constitutes a key element in providing precision, control, limitations and rigour into the security engineering discipline. Contracts reduce uncertainty and provide support for sound reasoning about dynamic, distributed and composed systems.

In the SERENITY model, contracts are mainly used to establish agreements between different SRFs, but the contents of the descriptions made using the S&D Artefacts can also be considered as contracts. In the first case, the SRF exposes a negotiation interface for external systems. The negotiation interface is used in order to establish the configuration of the interacting SRFs when two applications supported by different SRFs need to communicate using the same S&D Solution. In the second case, S&D Artefacts include means for expressing contractual facts (such as guarantees, SLAs, terms of use and disclaimers) as well as trust mechanisms designed to guarantee their origin and integrity by

means of digital signatures. These trust mechanisms can also be used to give validity to the contractual facts included in the artefact description.

Conclusions

In this paper we have discussed the drawbacks of threat-based security engineering and have advocated a change of paradigm based on the precise and formal specification of security solutions and security properties and the use of these formally verified properties as the basis for the expression of requirements and the engineering of secure systems. We have shown that threat-based security engineering (i) is the origin of penetrate-and-patch situation; (ii) does not result in specifications that can survive evolution of systems and context; and (iii) does not capture the requirements. We have introduced two pillars to solve the situation: the SERENITY model of secure and dependable systems and the concept of contract and the role it plays in ensuring a rigorous treatment of security.

We firmly believe that the trends that are driving the future computing scenarios are incompatible with threat-based security engineering, and therefore we propose to establish new models and principles that can result in a change of paradigm. The ultimate goal is to establish IT security as a fully fledged engineering discipline, based on the definition of integrated processes with well-defined goals and interfaces that combine the different techniques, methodologies and tools to support the engineering of future secure IT systems.

References

[1] Common Criteria Editorial Board. Common Criteria for Information Technology Security Evaluation. Version 3.1 Revision 1. September 2006. Available from http://www.commoncriteriaportal.org

[2] M. Schumacher, E. Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad. Security Patterns - Integrating Security and Systems Engineering. John Wiley & Sons, 2005.

[3] Congress of the United states of America. Sarbanes-Oxley Act of 2002. Available from http://www.access.gpo.gov

[4] Congress of the United states of America. Health Insurance Portability and Accountability Act of 1996. Available from http://www.hhs.gov/ocr/hipaa

[5] G. Spanoudakis, A. Maña and S. Kokolakis (2009). "Security and Dependability for Ambient Intelligence". Advances in Information Security, ISBN 978-0-387-88775-3. Springer.

[6] A. Maña, G. Pujol (2008). Towards formal specication of abstract security properties. Third International Conference on Availability, Reliability and Security (ARES2008), Barcelona, March 2008. IEEE Computer Society Press.

[7] F. Sanchez-Cid, A. Maña (2008). SERENITY Pattern-based Software Development Life-Cycle. 2nd International Workshop on Secure systems methodologies using patterns (SPATTERN'08). IEEE Computer Society.

[8] D. Serrano, A. Maña, and A. D. Sotirious (2008). Towards precise and certified security patterns. 2nd International Workshop on Secure systems methodologies using patterns (SPATTERN'08). IEEE Computer Society.