J. L. Vivas, I. Agudo, and J. Lopez, "Security Assurance During the Software Development Process", *International Conference on Computer Systems and Technologies (CompSysTech09)*, pp. II.7.1-II.7.6, 2009.
http://doi.acm.org/10.1145/1731740.1731748
NICS Lab. Publications: https://www.nics.uma.es/publications

Security Assurance during the Software Development Cycle

Isaac Agudo, José L. Vivas and Javier López

Abstract: *Assurance has been a major topic for critical systems. Assurance is usually associated with safety conditions but has also an important role for checking security requirements. Security is best assured if it is addressed holistically, systematically, and from the very beginning in the software's development process. We propose to integrate assurance and system development by letting the different stages of the system development life-cycle be mapped to the structure of the assurance case.*

Key words: *Security Engineering, Security Assurance, Assurance Cases, Common Criteria.*

INTRODUCTION

Assurance is confidence that an entity meets its requirements based on evidence provided by the application of assurance techniques. The assurance process should provide credible evidence that justifies a certain amount of confidence that the system meets its initial requirements.

Security assurance refers to security requirements. Assurance must provide evidence that the number of vulnerabilities in a software, including the presence of features that may be intentionally exploited by malicious agents, are reduced to such a degree that it justifies a certain amount of confidence that the security properties of the software meet the established security requirements, and that the degree of uncertainty involved has been reduced. The focus here is on the minimisation, not elimination, of vulnerabilities, since there can never be absolute certainty that these have been fully eliminated.

Many researchers hold that security and security assurance should be an integral part of the development process [7], and that security is best assured if it is addressed holistically, systematically, and from the very beginning in the software's development process. Hence, good systems engineering methods, techniques and practices might be seen in themselves as factors that increase confidence, and should therefore be treated as an integral part of an assurance process.

Assurance techniques may be informal, semiformal, or formal. Existing formal techniques are hard to apply and, although precise, are often inaccurate, i.e. do not target the real problem. Informal techniques, on the other hand, are often very little rigorous, but necessary. We propose to concentrate on semiformal methods as much as possible, making use of informal techniques as a complement or when semiformal techniques are not suitable or available, and formal techniques whenever they exist and are feasible.

Requirements can be functional or non-functional. Functional requirements describe interactions between the system and its environment, whereas non-functional requirements are constraints or restrictions limiting the design and implementation choices. Security is often seen as a non-functional requirement. However, non-functional requirements are typically not related, at least directly, to the functionality of the system, hence the attribute *non-functional*, but this is clearly not the case of the requirements related to security. Some of the security requirements are clearly functional in character, e.g. access control and authentication, and other requirements involve, either by design or as a side-effect, constraints to the functionality of the system, and may therefore be described as anti-functional rather than non-functional. Some approaches do justice to this feature, e.g. so called misuse or abuse cases [3], implicitly recognising the close relationship between security and functionality, since use cases are intended to express

the functionality requirements of a system, and many security vulnerabilities arise from the (mis)use of this functionality. Hence, eliminating those vulnerabilities often amount to reducing the intended or non intended functionality of the system.

The challenge here, from the assurance point of view, is that by their very nature it can never be assured that all the vulnerabilities of a system have been targeted or eliminated, since it is not possible to list every possible sequence of actions by brute force, nor are there efficient methods that may formally prove the security of a system. Testing is not enough in this context. Security testing methods are immature and testing by itself cannot gauge security as it is better suited to target the functional properties of a system and random errors rather than the vulnerabilities that can be exploited by malicious behaviour. In fact, evaluations methods such as the Common Criteria have been criticised for focusing on assuring mainly the functional security requirements of a system or product [5]. Hence, the adoption of sound system development practices might be more confidence building than testing and evaluation. Likewise, letting assurance be an integral part of system development will help in bringing on the adoption of sound development practices. Hence, what we are proposing here is in fact an integration of assurance with security engineering.

It must be noted in this context that security assurance requirements are currently almost always ignored in modern system development [6], which typically focuses on functional requirements. We have thus very little to fall back upon when it comes to best practices in the field. However, due to its importance, software assurance has become a very active area of research, and some advances have been reported in the latest 10 years. During this period, a high number of standards, techniques, methodologies, tools and initiatives have seen the light of day.

ASSURANCE BASED DEVELOPMENT (ABD)

Assurance Based Development (ABD) of critical systems [4] is an assurance methodology based on structured security assurance cases [2]. ABD has been recently proposed as an approach by which assurance is created throughout a system's development process. ABD allows developers to evaluate security goals during the development process. System development and assurance are in this way integrated. Criteria for the confidence of a development choice are provided at the time of choice, thus facilitating detection of potential security and assurance difficulties at an early stage of system development. Early developing of assurance cases can improve the development process by enabling the establishment of assurance and evidence requirements at every stage of the software development life cycle. In this way, security considerations can inform the requirements, design, architectural and implementations choices of a system development.

Assurance cases

Assurance case is the central concept of ABD. Assurance cases has been defined as "a documented body of evidence that provides a convincing and valid argument that a specified set of critical claims regarding a system's properties are adequately justified for a given application in a given environment." [1] It is not a method in competition with other security certification or evaluation methods, tools, or techniques. Security assurance cases "provide a general framework in which to incorporate and integrate existing and future certification and evaluation methods into a unified argument and evidentiary structure." [2] Assurance cases should be developed alongside the software component or system itself,

and the information they contain should depend on the level of assurance required at each life cycle phase. A security assurance case is a document that changes as the system it documents changes.

A security assurance case presents arguments showing how a top-level claim is supported by objective evidence, and considers people, processes, and technology. Assurance cases typically consist of at least three parts:

1. *Claims*. Claims embody what is to be shown. Top-level claims should "provide the 'take-home' message for the reviewers and should convince them that the required attributes have been satisfied." [2]
2. *Arguments*. Arguments show how a top-level claim is supported by evidence. An argument gives evidence about a claim based upon a set of other claims or sub-claims, until we reach a point where evidence is immediate
3. *Evidence*. Evidence may include testing, code review, formal mathematical proofs, arguments about the nature of the development process, the reputation of the development organisation, and the trustworthiness of the developers, among others.

An assurance case shows how a top-level claim is supported by lower level claims, which recursively are shown to be supported by other subclaims.

The consequences of a security breach will affect how much effort is put into developing arguments and claims, and some cases may therefore require a higher standard of evidence and argumentation than others. Security cases "provide also a framework for evaluating the impact of changes to the system and can help ensure that changes do not adversely impact security." [2]

Evidence may consist of any confidence enhancing element: programmer training credentials, results of the code review, testing results. Confidence in the argument depends on how convincing the argument and the evidence are in our eyes. Further evidence may be nevertheless required whenever we find the current one insufficient.

Claims should be stated succinctly and be unambiguous, and further information can be provided in an optional element called *context*. Optionally, the *strategy* used to develop a sub-claim from a claim can be explicitly stated, thus providing an additional cue to understand the form that an argument is going to take as well as information on how to substantiate a stated claim. Other optional elements, which we ignore here, are *justifications* and *assumptions*.

Assurance cases and the Common Criteria (CC)

The Common Criteria is an international standard for the evaluation of security products and systems. There have been proposals to use CC artefacts, such as *Security Targets*, as a basis for the definition software assurance cases. The security case is regarded as a more general framework than the CC, as the results of CC evaluations could be placed as evidence of assurance. [2] However, CC has been criticized for not providing "a meaningful basis for documentation of assurance cases that can be used to verify security as a property of software, but rather to prove the correctness of the security functionality provided by system components." [5]

A **Security Target** (ST) is the central CC document that specifies security evaluation criteria to substantiate the claims for the product's security properties. An ST may be described as a security case that gives a description of a security problem in terms of the product's description, threats, assumptions, security objectives, security functional requirements, and security assurance requirements. However, a CC evaluation indicates the degree of confidence that can be achieved, at a system level view, about the conformance of the security functions to the security policy with respect to the claims made in the ST. However, nothing is assured with respect to robustness against attacks or lack of exploitable vulnerabilities in the implementation of security functions at a component level view. [5]

In CC the required evidence is specified but, in contrast to assurance cases, the arguments linking this evidence to the goals and requirements of a system are commonly not given explicitly. Moreover, unlike CC evaluations, an assurance case is well suited to be maintained over time as a system development artefact. Assurance cases can thus evolve along with the system and in this way reflect its current state and configuration.

SECURITY ENGINEERING AND ASSURANCE

In accordance with the main guidelines of the ABD approach, we believe that assurance should address the requirements, design, architecture, and implementation issues of the system. We propose here an integration of security engineering and ABD with the help of assurance cases. The idea is to integrate assurance and system development by letting the different stages of the system development life-cycle be reflected in the structure of the assurance case, thus turning assurance cases themselves into a system development tool. This is done by identifying assurance claims with high level requirements and system goals, and building assurance case trees where the nodes at the highest levels correspond to system requirements devoid of references to architectural concepts, followed by lower level nodes denoting assurance claims that now refer mainly to design and architectural entities, and at a subsequently lower levels by letting corresponding assurance claims refer to concepts related to later stages of development, e.g. implementation or deployment. Hence, in the context of assurance cases requirements are identified with claims, the only difference being that whereas a requirement is stated in the subjunctive mode (e.g. confidentiality SHOULD BE enforced by the system), a claim is enunciated in the indicative mode (e.g. confidentiality IS enforced by the system).

In the requirements phase, the high-level security requirements or goals are established, and those can be further refined throughout the whole development process. In general, at the requirements phase the assurance process should target the definitions, omissions, mistakes, inconsistencies, completeness and soundness of the established requirements.

Following the requirements phase, during the design phase of system development the assurance techniques must provide evidence that the design meets the high-level security requirements, principles and goals established in the requirements phase. Claims should be made about the correctness of the design with respect to the high-level requirements, and the result should be a new design enriched this time with security features, thus facilitating the later implementation of the security requirements.

Later, during the ensuing implementation phase, implementation assurance should provide evidence that the implementation is consistent with the security design requirements by establishing implementation claims that guarantee that the implementation conforms to the

design. Testing and proof of correctness techniques may be used in this assessment. Later phases, such as deployment, should be treated similarly.

Hence, goals and requirements are intended to become the high level claims of the system. These goals should later be reduced throughout the different phases of development into lower level goals. As a result, an assurance case is built where lower level claims are given as evidence that higher level ones have been met, and arguments are given linking this evidence to the corresponding higher level claims. In this way, a hierarchy of goals arises that encompasses different levels of abstraction corresponding to different phases of system development, hence enabling also linking and tracing. *Assurance links* are established and documented between assurance arguments and development artefacts at every phase of development.

The choice of strategy for deriving subclaims from claims that correspond to an earlier phase of development (e.g. deriving design claims from system goals or implementation claims from design claims) is very important in our approach. This strategy should be based on a vulnerability analysis of the system and its components at each phase of development. For instance, if the requirements are given in the form of use cases, the strategy for deriving architectural or design claims from the requirements should include an analysis of the eventual vulnerabilities in the functionality of these use cases in the spirit of e.g. abuse or misuse cases. This analysis should then give rise to architectural claims that support the initial requirements in a way that is specific for each kind of system. At each phase of development the basic functionality of the system can be similarly analysed with regard to possible threats and vulnerabilities, and new claims derived until we reach the level of evidence. The end product of this approach should be an assurance case that provides a high degree of confidence about the robustness of the deployed system against attacks and the lack of exploitable vulnerabilities in the implementation of the security functions.

CONCLUSIONS AND FUTURE WORK

We have proposed the integration of security engineering and assurance based development with the help of assurance cases. This is done by integrating assurance and system development, in fact turning assurance cases themselves into a system development tool. In this methodology, requirements and system goals become high level claims in the assurance case tree, which is subsequently extended in a way that reflects each stage of development, later stages corresponding to lower level claims in the assurance tree. Strategies for deriving subclaims from parent claims can be based on the strategies for deriving more concrete views and models of the system under development, and should include an extensive vulnerability and risk analysis of the system view at hand. The subclaims produced in this way at one stage of development should be regarded as requirements for the subsequent stages. For reuse, patterns of decomposition of claims into subclaims for different types of systems and security requirements might turn out to be a useful by-product. The proposed methodology, which we believe can be advantageously integrated into software development methods such as Model-Driven Development, is now being applied in the development of a platform for online communities.

REFERENCES

[1] R E Bloomfield, P G Bishop, C C M Jones, P K D Froome, ASCAD—Adelard Safety Case Development Manual, Adelard 1998, ISBN 0 9533771 0 5.

[2] J. Goodenough, H. Lipson, and C. Weinstock: Arguing Security - Creating Security As-surance Cases, Carnegie Mellon University, 2007. Available at https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/assurance/643-BSI.html (last accessed September 19th, 2008).

[3] P. Hope, G. McGraw and A. I. Antón: Misuse and Abuse Cases: Getting Past the Positive. IEEE Security and Privacy, 2:3, pp. 90-92, May-June, 2004.

[4] E. A. Strunk and J. C. Knight: The Essential Synthesis of Problem Frames and Assurance Cases. In: Proceedings of 2nd International Workshop on Applications and Advances in Problem Frames, co-located with 29th International Conference on Software Engineering, Shanghai, May 2006.

[5] Software Security Assurance: A State-of-the-Art Report (SOAR) July 31th, 2007.

[6] J. Wil and J. Gustavsson : Security requirements – a field study of current practice. In: E-Proceedings of the Symposium on Requirements Engineering for Information Security, 2005.

[7] C. Woody: Process Improvement Should Link to Security. SERG 2007 Security Track Recap. Pitts-burgh, PA: Software Engineering Institute, Carnegie Mellon University, 2007.

ABOUT THE AUTHOR

Postdoctoral Research Assitant. Isaac Agudo, PhD, Department of Computer Science, University of Malaga (Spain). Phone: (+34)952134186, email: isaac@lcc.uma.es

Postdoctoral Research Assitant. Jose L. Vivas, PhD, Department of Computer Science, University of Malaga (Spain). Phone: (+34)952134186, email: jlvivas@lcc.uma.es

Professor. Javier Lopez, PhD, Department of Computer Science, University of Malaga (Spain). Phone: (+34)952134186, email: jlm@lcc.uma.es