ASASI: An Environment for Addressing Software Application Security Issues

Mehrez Essafi, Lamia Labed, Henda Ben Ghezala
National School for Computer Studies, RIADI Laboratory
University of Manouba
Tunis - Tunisia
mehrez.essafi@riadi.rnu.tn, Lamia.Labed@isg.rnu.tn, henda.benghezala@ensi.rnu.tn

Abstract—Security is an emergent property of a software system. Several efforts are undertaken, to improve software security. However, developers still miss or misuse acquired knowledge. This is mainly due to domain immaturity, newness of the field, process complexity and absence of environments supporting such development. This paper presents our environment denoted ASASI for Addressing Software Application Security Issues. The main feature of the proposed environment is that it is based on a strategy oriented process model that provides a two level guidance. The first level guidance is strategic helping developers choosing one among compilations of the existing methods, standards and best practices. The second level guidance is tactical helping developers achieving their selection for producing secure software. The supported process model is easily extensible and allows building customized processes adapted to context, developer’s finalities and product state. This flexibility allows the environment evolving through time to support new security requirements.

Keywords—environment; software; application; security; vulnerability; development; strategic; process; model; guidance.

I. INTRODUCTION

Research in security domain is concerned with: 1) security software which is a software whose primary functionality is to implement a security protocol or mechanism, and 2) secure software which is a software that operates correctly under malicious use and that doesn’t contain loopholes [11].

Secure software engineering requires a pro-active approach which is not an add-on collection of techniques. But, it is a feature of the entire software system and cannot be ensured just by using security mechanisms like access control, encryption, SSL, etc. [1]. So, it is different from security software.

Attacks on formally specified and verified protocols make the community realize that intuitive accuracy isn’t enough for ensuring security properties [25]. In fact, formal techniques have their own drawbacks as: 1) they can only verify designs, with theoretical limitations, 2) they cannot apply to the whole system and 3) cannot guarantee the security properties for implementations.

As a long-term, complex and multifaceted problem, improving software security and safeguarding the information technology requires multiple solutions and the application of resources throughout the whole life cycle.

This paper describes our environment ASASI basics for addressing software application security issues through its development process. The main feature of the proposed environment is that it is based on a strategy oriented process model that provides two level guidance: a strategic guidance helping the developer to choose one among compilations of existing methods, standards [4] and best practices [3] useful for producing secure software and a tactical guidance on how to achieve his selection.

II. SOFTWARE SECURITY ISSUES

Software security aspects become a subject of concern. In fact, it is mainly affected by software vulnerabilities [16][17]. Indeed an attacker may take advantage of a software vulnerability to compromise the three basic security properties, i.e., confidentiality, integrity and availability [1][16]. With the proliferation of hackers, who may exploit these vulnerabilities resulting in a highest risk to compromise systems. Risk is the probability that vulnerability is actually manifested or exploited, resulting in either an impact on normal software functioning or a failure. Then vulnerability is:

1) A defect that is an implementation and/or design error [25]. It may lie dormant in software for several years and then surface in a fielded system with major consequences.

2) A bug, which is an implementation-level software error. Bugs refer to low-level implementation errors that could be remedied by limited code analysis of the external environment, or,

3) A flaw that is a subtle defect at a deeper level [1].

We assume that security software needs to be himself secure to ensure system security otherwise, since it may contain software vulnerabilities, the latter could be exploited when performing attacks.

III. STANDARD SECURING APPROACHES

Standard securing approaches, such as verification and static analysis, which deal with the analysis of programs for common security flaws once they are built [28][1], or patching
techniques are not effective in ensuring a correct security [12]. In fact, they do not cover all security problems and address security at the end of the development process which may imply highest costs Fig 1. [24]. Therefore, a lot of progress has to be made in studying and analysing software artifacts before their deployment [13][26][20].

![Cost of fixing software bugs by development phase](image)

**IV. THE PROBLEM**

Security is an evolving threats landscape and what may appear today to be secure software could be shown to be insecure tomorrow.

A software system that is able to withstand attack provides some confidence that it is secure, but there are no generally accepted ways to prove that it is absolutely secure. With current methods, we can only prove that it is not secure. Beside, we still unable to fully verify that deployed, enhanced, repaired, or remediate secure software will preserve its security properties unaltered [2].

Face to the domain immaturity, newness of the field, absence of a support environment and the process complexity, software engineers and developers who lack security-engineering expertise, need sufficient knowledge and efficient guidance on how to find out the convenient and appropriate way for reaching the required software security level [25][1][6]. This knowledge and guidance must be supported by an environment to prosper better results.

**V. RELATED WORKS**

In the software engineering domain, no processes or practices have been shown to consistently produce secure software [2]. However, efforts are undertaken, to improve software engineering practices [17][22]. From these efforts resulted many analysis techniques – such as threat modelling [26], risk analysis and code analysing tools [15], many extensions to already existing formalisms –such as UMLSec and SecureUML– and many security related standards such as the Common Criteria [1]. Indeed, substantial reduction has been demonstrated in overall software design and implementation defects, as well as in security vulnerabilities [11][2][19].

Software engineering perspective is interested in how to enhance existing lifecycle phases, artifacts and techniques used in each phase, or perhaps introduce new techniques, to support security [13][26][15]. The holy grail of this field is software that is secure by construction [22][11][19]. We believe that security will be improved only by focusing on its development process since the early phases [2][20][23].

Many researchers have focused on using so-called best practices in the software lifecycle. As the name implies, a security development lifecycle is a Software Development Lifecycle (SDL) where a special emphasis is placed on software security in each phase. Two SDLs have been proposed in the state of the art which integrates software security into the lifecycle, one is by Microsoft as part of its Trustworthy Computing Initiative [17] and the other by McGraw [11]. We refer to these efforts as Microsoft’s SDL and McGraw’s SDL [21]. Obviously, both SDLs are too recent (respectively published in 2002 and 2004) and have a lot in common. They enumerate software security best practices applied to various software artefacts such as threat analysis, security review, security push/audit, etc. Both SDLs suggest to cycle, by iteration, on the proposed practices through more than once as the software evolves.

The high level of abstraction and the lack of guidance and details in these enhanced life-cycle models limit their usefulness in actually supporting and helping engineers in developing secure software [20].

**VI. DISCUSSION**

Secure software development processes need to be accessible to all developer’s profiles (from beginners to experts). They also need to be flexible and extensible instead of being enforced with a collection of predefined process models. Such flexibility increases with the multiplicity of ways proposed in the state of the art to reach the desired security level. In fact, there is never one way to proceed, but several ones.

These processes also need including additional steps dealing with risk analysis, threats modeling, vulnerability resolution, security test plans defining, etc. Such processes imply many resources and needs knowledge that most software engineers miss or misuse, that’s why they should provide guidance support for better usefulness.

To model the intended process, five kinds of process models can be used [7]: activity oriented, products oriented, decision oriented, context oriented and strategy-oriented process models. Our process model should help developers selecting the appropriate way to produce software with required security level. Thus the decision-oriented process, and activity or product-oriented models are not appropriate. In addition, the process needs to capture knowledge about how to progress in order to allow developer construct a personalized process. The latter depends on the application domain, developer’s experience and previous choices. Many ways are then possible to achieve the desired goal, and strategy-oriented process seems to be the best candidate for this modeling.

**VII. ADOPTED MODELING FORMALISM**

We assume security-engineering processes to be intention-oriented. At any moment, the engineer has an intention, a goal in mind that he wants to fulfill.

To model our process which will be supported by ASASI environment, we choose the MAP formalism which is a strategy oriented process model representing an extension to
the context oriented process model NATURE (Novel Approaches to Theories Underlying Requirements Engineering) [14][20][7]. The MAP identifies the set of intentions that have to be achieved in order to solve the problem (in our case security problem).

The key concepts of the MAP include: (1) intention which is a goal –expressed as a natural language statement comprising a verb– that can be achieved by performing a process and (2) strategy which is a possible way to achieve an intention. It is a labelled directed graph with intentions as nodes and strategies as edges between intentions.

The MAP can be seen as a set of process descriptions. Using dynamic selections from these descriptions, the best particular prescription adapted to the current situation of the software product is chosen. In that sense, the map is a multi-model. This multi-model allows the application engineers determine, through guidelines, the best way for specifying, designing, developing, testing, verifying or deploying a product and thus the best process model to do that.

As we can see in Fig 2., a MAP consist of a number of sections each of which is a triplet <II, Ij, Sij>, where II is source intention, Ij is a target intention and Sij is a strategy defining the way to go from the source to the target intention. The are two distinct intentions called Start and Stop that represent the intentions to start navigating in the MAP and to stop doing so. Thus, it can be seen that there are a number of paths in the graph from start to stop.

Guidelines used in the map result from previous acquired experiences and provided solutions to security problems.

Three kinds of guidelines are attached to the map: “Intention Achievement Guideline” (IAG), “Intention Selection Guideline” (ISG) and “Strategy Selection Guideline” (SSG). An IAG helps to fulfill the intention selected by the engineer – it could be non-formal, tactic or strategic – whereas ISG and SSG help the engineer progressing in the MAP and selecting the right section and are always tactic.

A MAP can evolve through time to support new sections in order to satisfy new requirements.

VIII. ASASI PROCESS MODEL OVERVIEW

Our solution aims at providing an environment for addressing software security issues that offers flexibility and provide guidance features for developers and engineers especially those who lack experience on how to build security in. The proposed environment does not impose activities to do, but enhances what can be done next and how it can be done in order to reach the desired security level.

The environment is build around a process model, which can be viewed as an alternative strategy that application engineers can choose for developing an application, specifically a secure one.

Fig 3. illustrates a MAP representing a software life cycle model. It gives an abstraction view on the different ways on how an application engineer can proceed to secure software when developing.

From this MAP, an application engineer can instantiate at least four different software securing processes (1) he develops the application using a traditional approach –which does not necessarily deal with security– and deploy the developed software. Then, he returns to development intention with maintenance strategy where he can use a securing strategy (2) he develops an application with a traditional approach, then he secures it using a securing strategy, which may include our approach (3) he combines 1 and 2 or 2 and 1. With these three approaches, security is treated at the end of the development process (4) he develops an application with a security oriented approach strategy that can include the Microsoft’s SDL and McGraw’s SDL in addition to our proposed approach. These four different processes could be expressed in term of sections as follow:

- 1st securing process (SP1): <Start, Develop an application, Traditional Approach><Develop an application, Evolve the Application, Development Strategy><Evolve the Application, Develop an application, Maintenance Strategy><Develop an application, Develop an application, Security Strategy> etc.
- 2nd securing process (SP2): <Start, Develop an application, Traditional Approach><Develop an application, Develop an application, Development Strategy><Evolve the Application, Develop an application, Security Strategy> etc.
- 3rd securing process: (SP1, SP2) OR (SP2, SP1), etc.
- 4th securing process: <Start, Develop an application, Security Oriented Approach>, etc.

Securing process instantiation is mainly driven by the provided strategy selection guidelines (SSG) and intention selection guidelines (ISG).
The process model we propose as a possible intention achievement guideline (IAG) for the security-oriented approach is illustrated by Fig 4. On this MAP, we find intentions that represent required steps to the development of secure software. Strategies represent possible known ways for releasing a step (or a section) in the process. First level intentions added here to the traditional developing steps (specify, design, develop and test) in order to prevent security problems in a proactive mode are:

- “Define a security policy” seeks for putting non-formal rules in order to maintain some desired security considerations.
- “Analyse risks” is for identifying potential threats and associated vulnerabilities depending on software (or, at a lower level, component) environment.
- “Model threats” is for studying identified threats in order to guide subsequent design, coding, and testing decisions.
- “Solve threats” is intended for solving threats and/ or associated vulnerabilities and find solutions that will be considered for ensuring desired security.
- “Choose security mechanisms” is for identifying security mechanisms that satisfy identified solution requirements.
- “Define test plans” aim at preparing security tests. It is mainly driven by the threat modelling results to verify the deployed solutions efficiency.
- “Audit” allows product inspection for possible emergent issues that weren’t previously considered.

For clarity reasons, we illustrate in Fig 4, only one strategy for each section that represents the strategy class. So, for example, “test techniques” strategy, in <Develop, test and analyse, test techniques> section, could be refined into:

1) “functional test” : Thus a <Develop, test and analyse, functional test> section required as the lowest security assurance level [1],

2) “structural test” : Thus a <Develop, test and analyse, structural test> section required as the second level security assurance, etc. (see Fig 5).

3) The strategy selection guideline (SSG 2.1): Associated to the testing section and is illustrated in Fig 6. This guideline will help the developer choose on the basis of the related arguments between the two candidate strategies (1) and (2).

![Testing section strategy selection guideline](image)

Figure 6. Testing section strategy selection guideline

**IX. ASASI ENVIRONMENT EXPERIMENTATION**

The use of ASASI environment consists in instantiating the proposed process model. For instance, the application engineer can define a security policy using a global approach and then specify the application requirement with respect to the defined policy. Then he can analyse risks depending on the application domain using a risk analysis method. This phase will allow identifying possible threats and vulnerabilities to consider later in the other phases. Next, the application engineer can model threats using a threat modelling technique. This will allow defining security tests scenario. Later, he can solve threats and vulnerabilities and inject solution elements in (1) the requirements specification phase for completeness (2) the design phase for implementation fault prevention (3) the
security policy defining phase for improvement (4) and the security mechanism choosing phase for an efficient and well targeted choice and so on.

The process model instantiation produces many security related artefacts such as attack trees, attack scenari, security test plans, security charts, etc.

For example, the process model was used to study software inputs. The latter are assumed to be the most exploited for performing attacks. As shown in Fig 7., we considered an input as composed by a content and a container.

![Figure 7. Input composition](image)

Input could be a command line argument, a parameter, a file, a socket, a register, a field, a dialog box, a port, etc. Risk analysis phase has demonstrated that an attacker could operate on the content and/ or the container to violate security properties. Attack trees modeling these two forms of threats are respectively illustrated by Fig 8. and Fig 9.

![Figure 8. Disturbing input container attack tree](image)

Fig 8. explains how a pirate could disturb a container which could be an input file or a memory field. For a file, he could destroy it and thus it will not be available for future use, or, he could modify one of the file properties such as its name, its location, its access control flags, etc. which may compromise the file availability and/ or the software system integrity. For the memory field, the pirate could, for example, overflow the field limits to access other memory locations and may violate their confidentiality or integrity properties.

![Figure 9. Disturbing input content attack tree](image)

Respectively, Fig 9. refines how a pirate could disturb a content such as transforming or altering it or by introducing its canonical form.

As we can notice, attack trees artifacts help understanding how to perform attacks. This provides developers a thinking support on how and where to act to solve threats and test them later. These attack trees support the use of logic operators between nodes and could be augmented for other exploit forms and thus evolve through time.

X. CONCLUSION

With today’s software invasion, it is recognized that security is critical and needs to be part of an end-to-end development either than being considered as an afterthought.

Several ways exist to enhance producing secure software. These different ways include methods, standards and best practices that should be inherent in the development process and accessible to developers who lack knowledge about how to address vulnerabilities in each phase of the development process.

The high level of abstraction in existing process models that try to include additional steps to address security problems limits their usefulness in actually supporting and helping in secure software developing.

This paper proposes an environment based on a strategy-oriented process model, which supports non-formal, semi formal and informal techniques stated for addressing software security concerns. The process model allows instantiating a personalised process for producing a secure application and offers strategic and tactical guidance on how and why to do so.

This work try to: 1) thwart developers –especially novice ones– from taking hypothesis and ad hoc decisions when studying, defining and including software security requirements and 2) help them realizing their processing intentions in a structured and well documented way.

REFERENCES


