Proceedings of the 10th WSEAS International Conference on COMPUTERS, Vouliagmeni, Athens, Greece, July 13-15, 2006 (pp362-367)

Addressing Software Application Security Issues

MEHREZ ESSAFI, HENDA BEN GHEZALA
RIADI-GDL Laboratory
National School for Computer Science Studies
University of Manouba, Tunis
TUNISIA

Abstract: - Software tend to be omnipresent in all modern systems. It often manipulates critical resources which interests pirates and need to be secured. Given the fact that most common software attacks can't be stopped or detected using conventional security mechanisms, malicious intruders try hack into systems by exploiting a software vulnerability. Vulnerabilities result from the use of traditional development processes – not focusing on security concerns – and the lack of necessary knowledge and guidance on how to produce secure software. They include implementation bugs such as buffer overflows and design flaws such as inconsistent error handling. Several efforts are undertaken, to improve secure software engineering, however, developers still miss or misuse acquired knowledge due to domain immaturity, newness of the field, process complexity and absence of environments supporting such development. This paper presents our approach addressing software application security issues through its development process using a strategy oriented process model. The main feature of the proposed process model is that it provides a two level guidance: 1) a strategic guidance helping the developer to choose one among a compilations of the existing methods, standards and best practices and 2) a tactic guidance helping the developer to achieve his selection. This process model is easily extensible and allows building customized processes adapted to the context, the developer's finalities and the product state.

Key-Words: - Software, application, security, vulnerability, development, strategic, process, model, guidance.

1 Introduction

Security is a broad area. It deals with cryptography, security protocols, access control, information flow, software security, program obfuscation, etc. Traditionally, security has always been treated as an add on. An application was assumed to be secure if it uses cryptography, security protocols, etc. Attacks on protocols make the community realize that intuitive accuracy isn't enough for security [3]. This results in the development of formal verification techniques for security concerns [31][33]. But, formal verification has its own disadvantages. First, it can only verify designs, with theoretical limitations, but cannot apply to the whole system and cannot guarantee the security properties for implementations.

Software security is mainly affected by software vulnerabilities [18][19]. With the proliferation of hackers, who exploit these vulnerabilities to compromise systems [28]. Therefore, security aspects of software become a subject of concern.

Vulnerability is either a defect, or a bug, or a flaw. Defects are implementation and/or design errors [3]. A defect may lie dormant in software for several years and then surface in a fielded system with major consequences. A bug 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. A flaw is a subtle defect at a deeper level [13]. Risk is the probability that a vulnerability is actually manifested or exploited, resulting in either an impact on normal software functioning or a failure.

An attacker may take advantage of a software vulnerability to compromise the three basic security properties, i.e., confidentiality, integrity and availability [13][18][28].

Considering that software tends to be omnipresent in all modern systems, software security is now a critical feature of the system as a whole. Research is concerned about: 1) security software which is a software whose primary functionality is to implement a security protocol or mechanism, and 2) security of software (or software security) which is software that functions correctly under malicious use and that doesn't contain loopholes [26].

This work fits in the second kind of research considering that security software need to be secure themselves to ensure security otherwise, they could be used as a source of attacks. This paper describes our approach in addressing software application security issues throw the development process using a strategy oriented process model. The main feature of the proposed approach is the two level guidance it provides: a strategic guidance helping the developer

to choose one among a compilation of the existing methods[1], standards[2] and best practices[3] useful for producing secure software and a tactic guidance to achieve his selection.

2 The problem

A program is a very detailed solution to a much more abstract problem. Leading from a problem to a program is a complex process [4].

Secure software engineering requires a pro-active approach. It is not an add-on collection of techniques. It is different from security software. It is a feature of the entire software system and cannot be ensured just by using security mechanisms like access control, encryption, SSL, etc. [21].

Standard approaches, such as verification and static analysis, which deal with the analysis of programs for common security flaws once they are built [4][13], or patching techniques are not effective in ensuring a correct security [27]. Therefore, a lot of progress has to be made in studying and analysing software artifacts [1][2][4][6][7][12].

The complex and multi-faced ways in which security permeates systems led us to believe that software engineering will be much more effective in ensuring security [13]. From a software engineering perspective, we are interested in how to enhance (1) existing lifecycle phases, (2) artifacts and (3) techniques used in each phase (or perhaps introduce new techniques?) to support security [1][2][6][7][8][9]. The holy grail of this field is software that is secure by construction [23][26][29][32]. We believe that security will improve only by focusing on its development process since the early phases [20].

The idea behind life-cycle models is to model how software should be produced, which means that these models are prescriptive. The use of prescriptive process models to develop secure software confronts us with some inconsistencies. These may be caused by deviation in the performed process from the predefined one in order to consider eventual modifications that will enforce the software security. The necessity of eventual modifications may emerge at any level of the development life cycle.

To ensure that a software development process and practices consistently produce secure products, candidate processes and practices and the products they produce must be analyzed and tested. This is to verify and validate that, when properly used, these processes and practices can be relied upon to produce secure products.

Verifying that a software process can consistently produce a secure software is a challenge for at least seven reasons [20]:

- Security is a landscape of evolving threats. What may appear to be secure software today could be shown to be insecure tomorrow.
- While tracking the ability of a software system to withstand attack provides some confidence that it is secure, but, there are no generally accepted ways to prove that it is. With current methods, we can only prove that it is not secure.
- The number of tolerable security defects is quite low and verifying that fewer than such a small number of defects exist in a large program is extremely difficult.
- Even if secure software has been initially developed, its deployment enhancement, repair, and remediation must not compromise its security. No generally accepted ways exist to verify that such software has preserved its security properties unaltered.
- Even after one or more processes have been shown to produce secure software, these processes and practices must remain effective when used by many people in many different software organizations and development environments.
- An extensive and extended data collection effort would be required to obtain statistically significant evidence that a process consistently produces secure software.
- The methods for qualifying the capabilities of the likely number of required processes and organizations would necessarily be time consuming and expensive.

In addition software engineers and developers, who lack security-engineering expertise, need better guidance to find out the convenient and appropriate way for reaching the required software security level [3][13][22].

3 Related work

In the domain of software engineering, no processes or practices have been shown to consistently produce secure software [20]. However, efforts are undertaken, to improve software engineering practices. Indeed, substantial reduction has been demonstrated in overall software design and implementation defects, as well as in security vulnerabilities [2][6][7][8][9][10][12][13][14][19][23][26][24][25][29][31][20][32][33].

---

[1] R2SIC – Research for Information and Communication System Security: http://www.cases.public.lu/publications/recherche/r2sic/

[2] JTC 1/SC 27 - IT Security techniques Standards: http://www.iso.org/

[3] Information Security Management References http://reform.house.gov/UploadedFiles/Best%20Practices%20Bibliography.pdf

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 to integrate software security into the lifecycle, one is by Microsoft as part of its Trustworthy Computing Initiative [19] and the other by McGraw[26]. We refer to these efforts as Microsoft's SDL and McGraw's SDL. Respectively Fig.1 and Fig.2 provide a pictorial overview of the two SDLs. Obviously, both SDLs have a lot in common. They enumerate software security best practices applied to various software artefacts with required iteration but no guidance is provided. The iteration here means that these practices will be cycled through more than once as the software evolves.

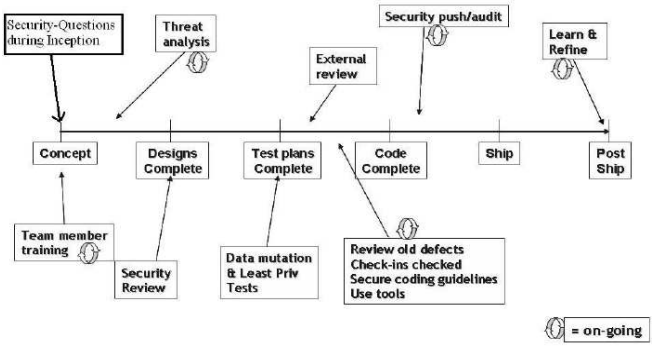

Fig.1 : Microsoft's SDL

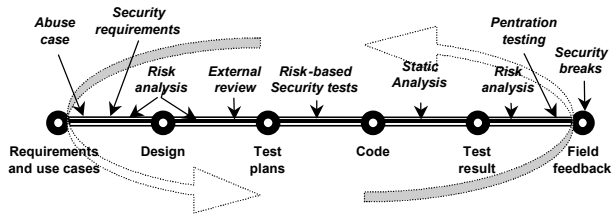

Fig.2 : McGraw's SDL

The high level of abstraction and the lack of guidance in these life-cycle models limit their usefulness in actually supporting and helping engineers in developing secure software [12].

4 Our modeling approach

In response to the challenges threatening secure software development processes and helping software engineers in this complex task, processes need to be extensible, flexible, and provide guidance during the process performance instead of being enforced with a collection of predefined process models. Such flexibility increases with the multiplicity of ways proposed to reach the desired security level. In fact, there is never one way to proceed, but several ones.

Secure software development process includes additional steps dealing with risk analysis, threats modeling, vulnerability resolution, security test plans defining, etc. These processes imply many resources and necessitate knowledge that most software engineers miss or misuse.

To model our process, five kinds of process models are candidates [17][30]: activity oriented, products oriented, decision oriented, context oriented and strategy-oriented process models.

The process model we are intending to model should help developers selecting the appropriate way to produce software with required security level. It is thus a 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 developers 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. To model our process, we choose the MAP formalism which is a strategy oriented process model representing an extension to the context oriented process model NATURE[4] [5][12][30].

The key concepts of the MAP include: (1) *intention* which is a goal that can be achieved by performing a process and (2) *strategy* which is a way to achieve an intension. 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.

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 fulfil the intention selected by the engineer – it could be non-formal, tactic or strategic – whereas ISG and

[4] Novel Approaches to Theories Underlying Requirements Engineering

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.

5 Proposed solution

Our solution aims at providing flexibility and guidance. It does not impose activities to do, but enhances what can be done next and how it can be done in order to address software security issues.

The proposed solution 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 the software life cycle meta model giving an abstraction view on how to secure software during development.

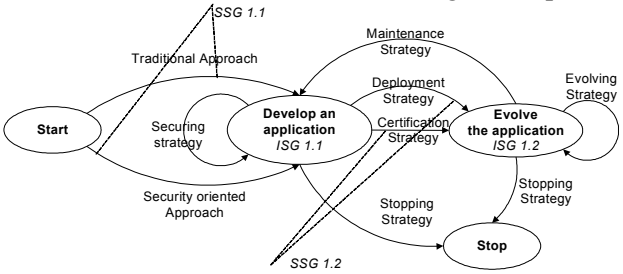

Fig.3 : Secure software development MAP

From this MAP, an application engineer can instantiate at least four 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 return 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 include our approach.

The MAP associated to our security-oriented approach is illustrated by Fig.4. On this MAP, we find identified intentions that represent required steps to the development of secure software. Strategies represent possible ways releasing a step in the process. If there are several candidate strategies when evolving from an intention to another, a strategy selection guideline is provided. For clarity reasons, we'll illustrate only one strategy for each section that represents the strategy class. So, for example, "test techniques" strategy could be refined into "functional test" – required as the lowest security assurance level[5], "structural test" – required as the second level security assurance, etc. The associated strategy selection guideline will guide the developer choosing between these strategies depending on his intentions and on the product state.

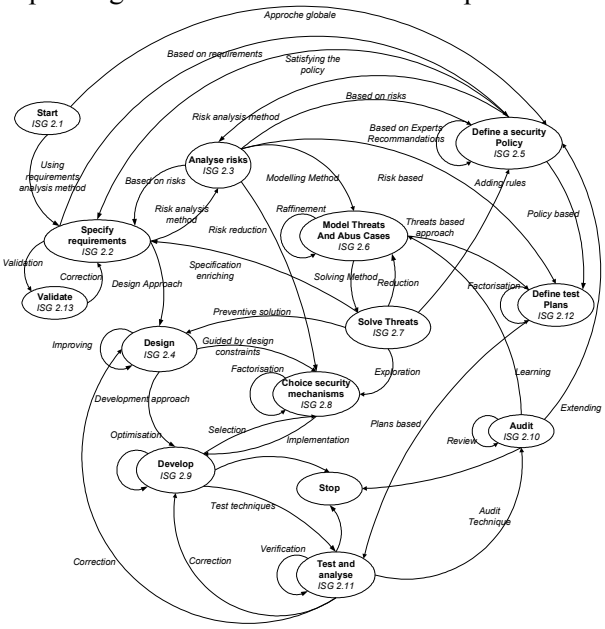

Fig.4 : Our security oriented approach MAP

The approach application consists in instantiating the proposed meta process. As instance – just to illustrate a possible process, 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. Next, the application engineer can model threats using a threat modelling technique. This will allow defining security tests scenari. 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.

6 Experimentation

The experimentation aims to verify the efficiency of guidance provided for identifying and solving security problems when instantiating the process model. Solving security problem in early phases considerably helps reducing the development costs and helps providing solutions that are independent of coding techniques and technologies. The experimentation also helps exploration of possible

[5] Common Criteria for Information Technology Security Evaluation (ISO 15408) – Part 3 : Security assurance requirements – version 2.1

ways provided to application engineers, who are not necessarily security experts, in addressing software application security issues.

The experimentation produces different artefacts. We give the following examples: 1) security test plans deduced from possible attacks scenari and 2) security solutions, which can be formally, semi-formally or informally expressed. These solutions can be considered in security requirements specification, policy definition, design and implementation. In Fig.5, we illustrates an example of artefacts that is a buffer overflow attack tree. This results from 1) risk analysis phase that helps identifying threats and associated vulnerabilities, and 2) threat modelling phase that allows understanding how threats will be transformed into attacks and required conditions for their achievement.

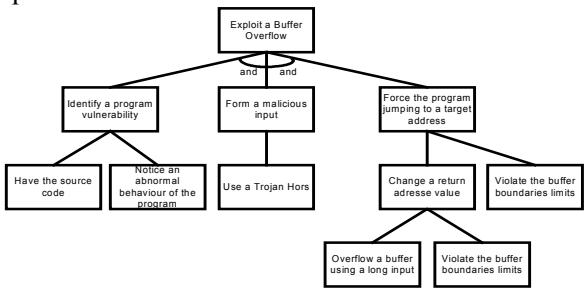

Fig.5 : Buffer overflow attack tree artefact

As we can see on Fig.5, to exploit a buffer overflow, many attack scenari are possible with logic operators between exploiting preconditions (here only AND operators are illustrated). So, to realise such kind of attacks, one must (have the source code OR Notice an abnormal behaviour of the program) AND (use a Trojan Hors to form a malicious input) AND (overflow a buffer using a long input OR violate the buffer boundaries limits to modify a return address value and so on). Some of these attack scenari could be used during security testing to verify if the defined solutions are efficient to prevent this kind of exploit. Solutions are defined in contrast with the preconditions to break them and prevent their satisfaction.

With ANDs between preconditions, partial solutions are thus possible. As sufficient solution that could be identified in the "solve threats" step using a formal strategy as solving method, the application engineer could try verifying the following constraints for each input buffer (that could be a command line argument, a parameter, a file, a register, a field, a dialog box, etc.):

Buffer*Capacity* ≤ DestinationBuffer*Length*
Buffer*Capacity* ≥ SourceBuffer*Length*
Buffer*Index* ≥ Buffer*LowerBound*
Buffer*Index* ≤ Buffer*UpperBound*
Buffer*UpperBound* ≤ Buffer*Length*

If verified, these constraints would relax the precondition "Force the program jumping to a target address" and then the attack will miss an important part for its release success.

7 Conclusion

A development process that includes security is now critical with today's software invasion. Security needs to be part of an end-to-end development. Today, it is recognized that there are different ways for producing secure software, but, the high level of abstraction in models limits their usefulness in actually supporting and helping in development.

Security policies, standards, guidelines and best practices should be inherent in the process and should be accessible to developers who lack knowledge about how to address vulnerabilities.

This paper proposes a strategic oriented process model, which supports non-formal, semi formal and informal techniques for addressing software security concerns. This model offers strategic and tactic guidance performing the development process.

Experimentation allowed demonstrating the efficiency of our approach even for reviewing the security of already existing software.

We are now developing the environment denoted "ASASI" for Addressing Software Application Security Issues aiming at assisting the application engineer in this critical and complex task. Our goal is to thwart him from taking hypothesis and ad hoc decisions including security requirements.

References:

[1] Abie H., Aredo D., Kristoffersen T., Mazaher S., Raguin T., Integrating a Security Requirement Language with UML, *In the Proc. of the Seventh International Conference on UML Modeling Languages and Applications, LNCS 3273*, Lisbon, Portugal, October, pp. 350-364, 2004.

[2] Alexander I., Misuse Cases: Use Cases with Hostile Intent, *IEEE Software*, vol. 20, no. 1, 2003, pp. 58–66.

[3] Anderson, R., Why Cryptosystems Fail, *In Proc. 1st Conf. On Computer and Communications Security*, 1993.

[4] Ball, T., The verified software challenge: A call for a holistic approach to reliability. *In Verified Software: Theories, Tools, Experiments*, October 2005.

[5] Ben Ghezala H., Bayoudh I., Jamoussi Y., Formalization of Navigation Strategies in a Map, *5th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, 2001.

[6] Besson F., Jensen T., Le Métayer D., Thorn T., Model checking security properties of control flow graphs, *Journal of Computer Security* , Volume 9, Issue 3, January 2001.

[7] Bieber P., *Interprétation d'un modèle de sécurité*, Techniques et Sciences Informatiques, Editions Hermes, Avril 1996.

[8] Burke R., Mobasher B., Zabicki R., Bhaumik R., Identifying Attack Models for Secure Recommendation, *Workshop: Beyond Personalization 2005, IUI'05*, January 9, 2005, San Diego, California, USA.

[9] Chen, H., Wagner, D., MOPS : An Infrastructure for Examining Security Properties of Software, *CCS'02, ACM*, USA, November 2002.

[10] Davis N., Humpphery W., Samuel T., Redwine JR., Zibulski, G., McGraw, G., Processes for Producing Secure Software, *Published By The IEEE Computer Society, IEEE Security & Privacy*, May/June 2004.

[11] Dowson M., Iteration in the Software Process, *Proceedings of the 9th International Conference on Software Engineering*, pp. 36-41, California, USA, May 1987

[12] Essafi M., Ben Ghezala H., Secure Software Engineering Processes, *3rd International Conference on Computing, Communications and Control Technologies (CCCT '05)*, July 24-27, 2005 - Austin, Texas, USA.

[13] Evans, D., Larochelle, D., Improving Security Using Extensible Lightweight Static Analysis, *IEEE Software*, January/February 2002

[14] Gilliam D., Kelly J., Bishop M., Reducing Software Security Risk Through an Integrated Approach, *Proc. of the Ninth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (June, 2000), Gaithersburg, MD, pp.141-146.

[15] Gilliam, D. P., Wolfe, T. L., Sherif J. S., Bishop, M., Software Security Checklist for the Software Life Cycle, *Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03)*, 2003

[16] Gilliam, D., Powell, J., Haugh, E., Bishop, M., Addressing Software Security and Mitigations in the Life Cycle, *Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop (SEW'03), IEEE Computer Society*, 2003.

[17] Grosz, G., Modeling and Engineering the Requirements Engineering Process : An overview of the Nature Approach, *Requirements Engineering Journal, Vol2 n°3*, pp. 115-131, 1997.

[18] Hoglund, G., McGraw, G., *Exploiting Software How to Break Code*, Addison Wesley, February 17, 2004.

[19] Howard, M., LeBlanc, D., *Writing Secure Code*, Microsoft Press- Microsoft Corporation, 2002.

[20] Improving Security Across The Software Development Life Cycle, *Task Force Report*, April 1, 2004.

[21] Jaquith A., The Security of Applications: Not All Are Created Equal, *Research Report, @Stake*, February 2002, pp. 1-12.

[22] Mann, C., Why Software Is so Bad, *Technology Review*, July/August 2002.

[23] Mark G. Graff, Kenneth R. van Wyk, *Secure Coding : Principles & Practices*, O'Reilly, June 2003

[24] McGraw G., Misuse and Abuse Cases: Getting Past the Positive, *published by The IEEE Computer Society, IEEE Security & Privacy*, May/June 2004.

[25] McGraw G., Software Assurance for Security, *IEEE Computer* 32(4), pp. 103-105 (April, 1999).

[26] McGraw G., Software Security : Building Security IN, *In IEEE Computer Society, IEEE Security and Privacy*, 2004.

[27] McGraw, G., Testing for Security During Development : Why We Should Scrap Penetrate-and–Patch, *IEEE Aerospace and Electronic Systems*, vol. 13, no. 4, 1998, pp.13-15.

[28] NIST. The Economic Impacts of Inadequate Infrastructure for Software Testing, *Planning Report*, *Gaithersburg, MD: National Institute of Standards and Technology*, 2002.

[29] Pescatore, J., Keys to Achieving Secure Software Systems, *Gartner Inc.*, September 22, 2004.

[30] Rolland C., Prakash N., Benjamen A., A Multi-Model View of Process Modelling, *Requirements Engineering Journal*, 1999.

[31] Siemens, Current Formal Security Models, *Information and Communication Security*, 2003.

[32] Viega J., McGraw G., *Building Secure Software: How to avoid Security Problems the Right Way*, Addison-Wesley, Boston, 2001.

[33] Wimmel G., Wisspeintner A., Extended description techniques for security engineering, *In Michel Dupuy and Pierre Paradinas, editors, Trusted Information-The New Decade Challenge, IFIP TC11 16th International Conference on Information Security (IFIP/Sec'01)*, June 11-13, 2001, Paris, France, pages 470-485. Kluwer Academic Publishers, 2001.