Défi : La sécurité dans le développement logiciel

Equipe Archware/IRISA

Décembre 2019

1 Introduction

Sécuriser un logiciel doit être un principe de conception qui concerne chaque étape du développement, de l'ingénierie des exigences à la conception, implémentation, test et déploiement. En effet, de plus en plus le principe *secure by design* en génie logiciel devient la principale approche de développement pour assurer la sécurité et la confidentialité des logiciels. Elle s'attache à sécuriser le logiciel dès les fondations et démarre par une conception d'architecture robuste. Pour ce défi fondé sur le *secure by design* nous allons parcourir le cycle de vie (ou cycle de développement) du logiciel afin de pointer les aspects de sécurité dans chaque étape et formuler les questions à résoudre pour les problématiques associées.

2 La sécurité dans le cycle de vie du logiciel

Nous avons organisé cette proposition de défi en examinant comment la sécurité est prise en compte [2] ou pourrait être prise en compte dans les différentes étapes de développement d'un logiciel ainsi que l'étape de maintenance et évolution.

2.1 Sécuriser les exigences

La prise en compte des aspects sécurité dans le développement logiciel (et système) nécessite de spécifier en amont ce que l'on souhaite sécuriser. Cela pose plusieurs problèmes. D'une part, les exigences de sécurité sont généralement issues des analyses de sécurité. Ces analyses, menées via des méthodes telles que EBIOS[1] ou encore NIST-800-30[2], ne sont pas conçues pour éliciter des exigences mais produire une politique de sécurité. Si ces rapports identifient les biens (*assets*) à prendre en compte, ils ne définissent pas clairement les exigences. Passer de l'un à l'autre n'est pas trivial et nécessite la recherche de moyens guidant et/ou automatisant ce passage.

D'autre part, si l'utilisation de cette démarche peut s'appliquer dans des projets de grande taille, ce n'est pas le cas dans des projets plus modestes ou dans

[1] https://www.ssi.gouv.fr/guide/ebios-2010-expression-des-besoins-et-identification-des-objectifs-de-securite/

[2] https://csrc.nist.gov/publications/detail/sp/800-30/rev-1/final

des processus plus agiles dans lesquels le coût nécessaire à les conduire serait prohibitif. Cela nécessite donc des efforts afin de simplifier et/ou d'automatiser les analyses de risques, et de capturer leur résultat de manière simple et rapide. Enfin, si les spécialistes des domaines sont généralement aptes à eliciter des exigences fonctionnelles, ils le sont moins pour les exigences non fonctionnelles. L'élicitation de telles exigences passe donc par une collaboration entre experts des domaines et experts de la sécurité. Il est donc besoin de proposer des moyens collaboratifs afin de tisser un lien entre ces deux mondes.

2.2 Sécuriser l'architecture logicielle et la conception

Depuis plus d'une dizaine d'années des propositions de langages de modélisation intégrant des aspects de sécurité ont vu le jour [5]. Les plus connus sont SecureUML [1] et UMLSec [4]. Certains langages se concentrent sur un ou plusieurs attributs déclaratifs de sécurité, tels que la confidentialité, l'intégrité ou la disponibilité. D'autres langages se concentrent sur un ou plusieurs aspects opérationnels de sécurité, tels que le contrôle d'accès ou l'authentification. Finalement il existe des langages se concentrant sur un ou plusieurs domaines, tels que les architectures orientées services ou les bases de données, ou au contraire restant génériques, indépendants du domaine.

En majorité ces langages proposent une syntaxe concrète pour la représentation de la sécurité, basée sur UML, et ciblent les aspect structurels de la sécurité dans la phase de conception détaillée. Très peu de propositions offrent l'analyse de la représentation.

L'analyse est l'activité de vérification d'une représentation d'architecture par rapport aux exigences de sécurité. L'analyse peut être algorithmique ou basée sur l'expertise, c'est-à-dire sur le savoir implicite et le raisonnement humain. La plupart des peu nombreuses approches qui assurent l'analyse se concentrent sur l'accès de contrôle.

La travaux sur la sécurité au niveau architectural concernent le plus souvent la modélisation des solutions, des contre-mesures, des contrôles de sécurité. Néanmoins, nous sommes en manque d'approches utilisant des méthodes et outils théoriques et pratiques pour modéliser, définir ou identifier des failles (ou vulnérabilités) architecturales. Alors que de nombreuses bases de vulnérabilités existent au niveau du code (par exemple CVE, CWE), il est difficile de caractériser les vulnérabilités architecturales : sont-elles des *bad smells*, des *anti-patterns*, etc ? Comment les identifier: avec des approches de *model matching* ou d'apprentissage automatique, etc ? Comment proposer des solutions à ces vulnérabilités ?

La question de la mesure du niveau de sécurité d'une architecture se pose également. Comment identifier les éléments d'une architecture en lien avec la sécurité ? Comment exprimer la sémantique d'un élément d'architecture ? Quelles métriques définir pour caractériser le niveau de sécurité d'une architecture ? Serait-il possible de définir des métriques indépendantes du domaine auquel l'architecture s'applique ? Comment tester et comparer ces métriques ?

Un nombre important de patrons de conception de sécurité a été proposé.

Ils sont recensés dans des catalogues tels que [3]. La question de leur intégration semi-automatique ou assistée par ordinateur reste néanmoins tout entière. Dans quel format représenter les patrons de sécurité pour pouvoir les intégrer dans les langages d'architecture ? Comment identifier les situations ou le contexte dans les modèles d'architecture, dans lesquels les patrons seraient applicables ? Comment les appliquer (automatiquement) aux modèles concrets d'architecture ? Comment vérifier que leur application est correcte ? Comment vérifier et quantifier que l'architecture est mieux sécurisée suite à leur application ?

Finalement nous pourrions aussi évoquer les comportements émergeants et malveillants qui apparaissent dans les systèmes de systèmes.

2.3 Sécuriser le code et le déploiement

Lors de l'implémentation il existe deux sources de vulnérabilité : 1) une mauvaise implantation de propriétés de sécurité voulues et spécifiées lors des phases précédentes ; 2) une écriture naïve de la partie fonctionnelle laissant apparaître des vulnérabilités potentiellement exploitables par des attaquants. Pour le premier cas, la question de recherche et de savoir comment mesurer l'adéquation d'une implémentation aux besoins exprimés à travers certaines propriétés de sécurité ? Pour le second cas, la question de recherche est de savoir comment s'assurer qu'un code ne contient pas de vulnérabilité ? Cette dernière question nous amène à une autre question : qu'est-ce qu'une vulnérabilité ? Nous disposons d'une certaine connaissance cumulative (CVE), mais peut-elle être exploitée de manière efficace ? Quel est son niveau de complétude ? Peut-on proposer des règles de bonnes pratiques de codage permettant d'éviter certains types de vulnérabilités ? A travers toutes ces questions, loin d'être exhaustives, nous imaginons un grand champ de recherche qui ne demande qu'à être exploré.

La phase d'implémentation inclut les tests unitaires. Quels types de tests orientés sécurité peuvent être réalisés lors des tests unitaires ?

Avant le déploiement, nous réalisons les tests d'intégration. Ces derniers s'appuient sur les cas d'utilisation. Nous pourrions nous appuyer sur les cas malicieux pour échafauder des tests orientés sécurité. Ce sont des tests d'intrusion orientés sémantique d'application, par opposition aux tests d'intrusion classiques qui visent majoritairement la vulnérabilité de l'infrastructure du déploiement. La question est : peut-on avoir des techniques de test orientés intrusion sémantique ?

Une autre aide, précieuse lors de l'implémentation, concerne les métriques logicielles orientées sécurité. La question est : peut-on avoir des métriques pouvant qualifier les propriétés de sécurité ?

Lors du déploiement il s'agit d'dentifier des vulnérabilités dues à l'interaction avec l'environnement (e.g. infrastructure, OS), mais aussi avec les utilisateurs. En effet, la réussite des cyber-attaques résulte souvent d'une erreur humaine. Permettre l'évaluation des vulnérabilités humaines au même titre que des vulnérabilités système, aiderait à réduire la vulnérabilité globale.

2.4 Sécuriser la phase de maintenance

La sécurité dans la phase de maintenance a deux objectifs : 1) repérer et corriger des vulnérabilités rencontrées lors des dernières exécutions du système ; 2) sur la base du vécu (traces d'exécution), essayer d'identifier des vulnérabilités qui n'ont pas été encore mises à jour. La question de recherche ici concerne le mariage des techniques de *forensic* et des techniques de ré-ingénierie logicielle. Bien sûr, d'autres aspects qui concernent la sécurité lors de la phase de maintenance peuvent exister. Finalement on peut aussi s'interroger sur l'impact de l'évolution sur la sécurité.

3 Conclusions

Nous avons présenté notre défi sous la forme de quelques pistes pour produire des logiciels sécurisés en suivant le principe du *secure by design*. Des questions générales se posent également plus spécifiquement sur le cycle de développement : comment assurer l'intégration (continue) des aspects de sécurité tout au long du cycle de vie ? Comment se concentrer sur l'interface entre les différents phases ? Comment définir et respecter les bonnes pratiques ? Il reste également beaucoup d'aspects à prendre en compte, concernant notamment les processus agiles ou non ou d'intégration de la sécurité.

References

[1] *SecureUML:A UML-based modeling language for model-driven security*, volume 2460 of *Lecture Notes in Computer Science*. Springer, 2002.

[2] Premkumar T. Devanbu and Stuart G. Stubblebine. Software engineering for security: a roadmap. In *22nd International Conference on on Software Engineering, Future of Software Engineering Track, ICSE 2000, Limerick Ireland, June 4-11, 2000*, pages 227–239, 2000.

[3] Eduardo Fernandez-Buglioni. *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. Wiley Publishing, 1st edition, 2013.

[4] Jan Jürjens. Umlsec: Extending UML for secure systems development. In *UML 2002 - The Unified Modeling Language, 5th International Conference,Dresden, Germany, September 30 - October 4, 2002, Proceedings*, pages 412–425, 2002.

[5] Alexander Van Den Berghe, Riccardo Scandariato, Koen Yskout, and Wouter Joosen. Design notations for secure software: A systematic literature review. *Softw. Syst. Model.*, 16(3):809–831, 2017.