Défi : La sécurité dans le développement logiciel

Equipe Archware/IRISA

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 la conception d'une architecture robuste. Le défi que nous proposons consiste à passer du cycle de vie du logiciel au cycle de vie du logiciel *sécurisé*, mettant en œuvre le principe *secure by design* du point de vue du génie de la programmation et du logiciel.

2 Transversalité du défi

Ce défi se place à l'intersection de la sécurité et du génie de la programmation et du logiciel. Le GDR Sécurité [1] structure la communauté de recherche en sécurité en France. Il se concentre, au travers de ses groupes de travail, essentiellement sur des problématiques liées aux réseaux et à la protection des données. À l'exception du groupe de travail « Méthodes formelles pour la sécurité [2] » et en partie du groupe « Sécurité des systèmes, des logiciels et des réseaux [3] », la question du logiciel nous semble peu traitée par ce GDR Sécurité. Or ces deux groupes de travail ne traitent pas des questions de sécurité au niveau des exigences ou de l'architecture. Le défi que nous proposons adopte un point de vue différent, qui n'est pas couvert par les groupes de travail existants.

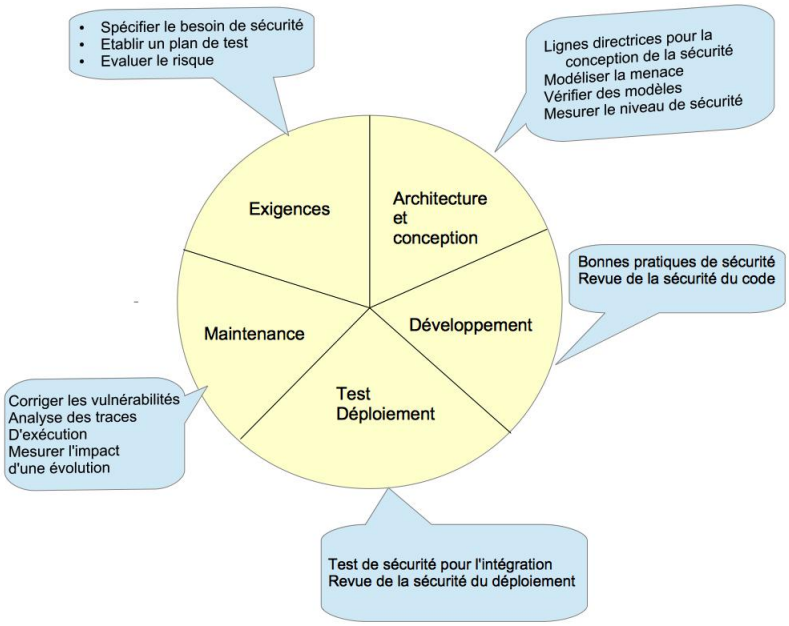
La majorité des groupes de travail du GDR/GPL pourraient contribuer à ce défi : IE (Ingénierie des exigences), RIMEL (Rétro-Ingénierie, Maintenance et Evolution des Logiciels), SDS (Systèmes de systèmes), IDM (Ingénierie dirigée par les modèles), MTM2 (Méthodes de test pour la validation et la vérification) pour ne citer que ceux-là.

Nous pensons qu'une initiative est utile pour rassembler et animer une communauté autour de ce défi.

3 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 [3] ou pourrait être prise en compte dans les différentes étapes de développement d'un logiciel ainsi que dans l'étape de maintenance et évolution. La figure ci-dessous montre des activités d'ingénierie de sécurité et où elles pourraient être intégrées dans le cycle de vie d'une application.

1. https://gdr-securite.irisa.fr/
2. http://gtmfsec.irisa.fr/
3. https://gdr-securite.irisa.fr/securite-des-systemes-des-logiciels-et-des-reseaux/



Nous allons parcourir le cycle de vie du logiciel afin de pointer les aspects de sécurité dans chaque étape et formuler les questions à résoudre pour les problématiques associées.

3.1 Sécuriser les exigences

La prise en compte des aspects sécurité dans le développement logiciel (et système) nécessite de spécifier en amont quels sont les objectifs de sécurité et 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 [4] ou encore NIST-800-30 [5], 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. Donc une première étape critique est d'obtenir les exigences de sécurité dans le cycle de développement. Pour faciliter cela, il est possible d'imaginer l'élaboration d'un catalogue d'exigences de sécurité ou de catégories d'exigences. Il s'agit d'une autre direction à explorer. 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 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 éliciter des exigences fonctionnelles, ils le sont moins pour les exigences non fonctionnelles. L'élicitation de telles exigences passe donc par une collaboration entre

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

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

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.

3.2 Sécuriser l'architecture logicielle et la conception

Pour éviter de nombreuses vulnérabilités introduites par de mauvais choix de conception, l'activité de conception doit utiliser des bonnes pratiques, des modèles et des principes de conception éprouvés.

Connaître les menaces Il est important de connaître les menaces auxquelles l'architecture et la conception sont exposées. L'activité de modélisation des menaces [6] permet d'identifier et de comprendre les menaces, de comprendre les risques que chaque menace pose et de découvrir les vulnérabilités que nous pouvons utiliser pour façonner les décisions de conception et de mise en œuvre de la sécurité. Comme nous l'avons écrit précédemment, les analyses de sécurité comme EBIOS prévoient d'identifier les menaces et les objectifs susceptibles d'être visés, les objectifs recherchés par ces menaces. Une piste pourrait être de considérer qu'il existe, en plus des cas d'utilisation, des cas malicieux correspondant à ces objectifs visés.

Modéliser Depuis plus d'une dizaine d'années des propositions de langages de modélisation intégrant des aspects de sécurité ont vu le jour [7]. Les plus connus sont SecureUML [1] et UMLSec [5]. 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.

Ces langages 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 ? Une piste à explorer serait de reprendre tous ces concepts, de trouver une façon uniforme de les modéliser, de les positionner et de les associer éventuellement à des outils pour constituer un catalogue exploitable au moment de l'identification des failles de l'architecture.

Vérifier les mesures de sécurité En majorité les langages de modélisation, intégrant des aspects de sécurité, proposent une syntaxe concrète pour la représentation de la sécurité, basée sur UML, ils ciblent les aspect structurels de la sécurité dans la phase de conception détaillée. Très peu de propositions offrent l'analyse de la représentation. Or c'est bien un des objectifs de l'ingénierie d'un systèmes sécurisé : vérifier que les mesures adéquates ont été prises pour contrer les cas malicieux identifiés. Peu nombreuses sont les approches qui assurent l'analyse de l'architecture par rapport aux exigences de sécurité, et la plupart de ces approches se concentrent sur le contrôle d'accès. Il est nécessaire de proposer des analyses pour les autres types de mesures de sécurité, afin de vérifier la prise en compte des cas malicieux.

Pour répondre à ce défi de l'analyse, une piste pourrait être de proposer comment évaluer, mesurer le niveau de sécurité d'une architecture, dans sa globalité. Cela pose plusieurs questions. 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 ?

Dans un système complexe ou dans un système de systèmes, des comportements émergent de l'interaction entre les constituants. Comme l'a montré par exemple la notion d'attaque en cascade, certains de ces comportements émergeants peuvent introduire des vulnérabilités, exploitables par un acteur malveillant. Analyser ces systèmes pour détecter de telles vulnérabilités émergeantes est un réel défi.

Proposer des améliorations En supposant qu'il soit possible d'identifier les vulnérabilités ou, par analyse, de détecter des exigences de sécurité non satisfaites, resterait le défi de proposer à l'architecte des solutions potentielles, pour améliorer la sécurité au niveau de l'architecture.

Un nombre important de patrons de conception de sécurité a été proposé. Ils sont recensés dans des catalogues tels que [4]. Dans un objectif de *refactoring* d'une architecture pour la sécuriser, la question de l'intégration semi-automatique ou assistée par ordinateur des patrons 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 d'architecture ? Comment vérifier qu'un patron est correctement employé, c'est-à-dire dans le bon contexte et pour répondre aux bonnes exigences ? Comment quantifier la contribution de chaque patron au niveau de sécurité de l'architecture ?

Une piste pour répondre à certaines de ces questions pourrait être, par exemple, de s'appuyer sur le catalogue d'exigences ou de catégories d'exigences de sécurité que nous envisageons comme piste à la section précédente.

3.3 Sécuriser le code et son déploiement

Les exigences et l'architecture spécifient la sécurité du système. Encore faut-il que l'implémentation et le déploiement mettent effectivement en œuvre ce qui a été exprimé lors des phases précédentes du cycle de développement.

Programmer pour la sécurité Une mauvaise implantation de propriétés de sécurité voulues et spécifiées lors des phases précédentes ruinerait les efforts concédés lors de la conception de l'architecture. Comment mesurer l'adéquation d'une implémentation aux besoins exprimés à travers certaines propriétés de sécurité ?

Par ailleurs, une écriture naïve de la partie fonctionnelle laisserait apparaître des vulnérabilités potentiellement exploitables par des attaquants. Comment s'assurer qu'un code ne contient pas de vulnérabilité ? Qu'est-ce qu'une vulnérabilité ? Nous disposons d'une certaine connaissance cumulative (CVE), et il existe des règles de bonnes pratiques de codage (*cheat sheets* OWASP [6] ou les guides de l'ANSSI comme les références [7] [8] par exemple). Mais comment exploiter ces ressources de manière efficace ? Quel est leur niveau de complétude ? La plupart des travaux dans le domaine proposent des audits ou une analyse de code pour détecter des vulnérabilités et identifier des parties

6. https://cheatsheetseries.owasp.org/
7. https://www.ssi.gouv.fr/particulier/guide/recommandations-pour-la-securisation-des-sites-web/
8. https://anssi-fr.github.io/rust-guide

du code potentiellement dangereuses. Une autre piste à explorer pour sécuriser le développement du code, une fois les problèmes identifiés, est de proposer une correction de sécurité qui passera les tests et les outils de mesures. Toutefois les outils d'analyse produisent beaucoup de faux positifs existants [2] et ne détectent que les vulnérabilités connues. Proposer des scanners de code efficaces devient une urgence au vu de la taille du code des applications d'aujourd'hui.

Tests de sécurité La phase d'implémentation inclut les tests unitaires. Quels types de tests orientés sécurité peuvent être réalisés lors des tests unitaires ? Comment exploiter la sémantique (connue) de l'application pour orienter les tests de sécurité unitaires et d'intégration ? Une piste pourrait être de s'appuyer sur les cas malicieux pour élaborer des tests ciblés.

Mesure du niveau de sécurité De même qu'au niveau de l'architecture, existe-t-il des métriques au niveau du code pour qualifier ou quantifier les propriétés de sécurité ? Des outils et des algorithmes existent pour mesurer la qualité d'un code source, il faudrait prendre une démarche similaire pour trouver des mesures du niveau de sécurité.

Déploiement Lors du déploiement il s'agit d'identifier 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. Les recherches ciblant les systèmes socio-techniques vont dans ce sens.

3.4 Sécuriser la phase de maintenance

La sécurité dans la phase de maintenance consiste à analyser l'application de manière structurée et systématique pour repérer ses menaces et ses vulnérabilité. Ses deux objectifs sont : 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 d'analyse post mortem (*forensic*) et des techniques de ré-ingénierie logicielle. L'analyse post mortem est l'analyse d'un système après incident. Diverses techniques existent pour extraire la quantité maximale d'information et recueillir autant de preuves, retrouver toutes les données cachées et non tracées. Une piste peut être d'explorer ces techniques dans le cadre d'une rétro-ingénierie avec les objectifs mentionnés plus haut.

La maintenance d'une application logicielle reste l'activité la plus délicate car, en effet, suite à une évolution, il faut maintenir le niveau de sécurité tout en maîtrisant le coût de l'évolution. Finalement on peut aussi s'interroger sur l'impact de l'évolution sur la sécurité.

Il existe bien sûr d'autres aspects concernant la sécurité lors de la phase de maintenance.

4 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.

Références

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

[2] R. Benabidallah, S. Sadou, B. Le Trionnaire, and I. Borne. Designing a code vulnerability meta-scanner. In *ISPEC2019 - The 15th International Conference on Information Security Practice and Experience, Lumpur, Malaysia, November 26-28 2019, Proceedings*, pages 194–210, 2019.

[3] 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.

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

[5] 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.

[6] Adam Shostack. *Threat Modeling : Designing for Security*. Wiley Publishing, 1st edition, 2014.

[7] 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.