HAL
open science

Un Data-Store pour la Génération de Cas de Tests

Loukmen Regainia, Cédric Bouhours, Sébastien Salva

To cite this version:

Loukmen Regainia, Cédric Bouhours, Sébastien Salva. Un Data-Store pour la Génération de Cas de Tests. 16th AFADL Approches Formelles dans l'Assistance au Développement de Logiciels, journées du GDR GPL, Jun 2017, Montpellier, France. hal-01715105

HAL Id: hal-01715105

https://uca.hal.science/hal-01715105v1

Submitted on 22 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Un Data-Store pour la Génération de Cas de Tests

Loukmen Regainia, Cédric Bouhours and Sébastien Salva
LIMOS - UMR CNRS 6158, Auvergne University, France
Email : [prenom.nom]@uca.fr

Résumé

Dans le but d'aider les concepteurs dans le choix des patrons de sécurité face aux attaques, nous présentons dans ce papier un data-store regroupant les attaques et les patrons de sécurité. Il permet également la génération automatique d'arbres d'attaques et de défenses ainsi que la génération de squelettes de cas de test. La méthodologie proposée cherche à couvrir un grand nombre de scenarii d'attaques indépendamment des contextes des applications. Nous présentons les cas de test dans une forme qui permet aux concepteurs d'avoir à la fois un support visuel décrivant les actions d'attaques et de défense, une forme compréhensible des cas tests et un ensemble de directives lui permettant le codage de chaque cas.
Mots clés : Patrons de sécurité ; Patrons d'attaque ; Cas de test ; Arbres d'attaque et de défense

1 Introduction

La conception d'une application sécurisée est une tâche difficile. D'une part la sécurité dès la phase de conception n'est pas une compétence courante dans les équipes de développement, et d'autre part la gestion des problèmes de sécurité aux dernières étapes de la vie de l'application est compliquée et manque d'efficacité. Afin d'aider les équipes de développement à gérer la sécurité des applications dès les premières phases du cycle de vie, les concepteurs ont à leur disposition la notion de patrons de sécurité qui sont des solutions communautaires génériques aux problèmes récurrents de sécurité permettant de prévenir partiellement ou complètement des attaques. La nature abstraite et le nombre croissant des patrons de sécurité permet de couvrir une myriade de problématiques de sécurité indépendamment des contextes d'application. Néanmoins, cela rend difficile le choix d'un patron de sécurité dans le but de contrer une attaque.

Dans ce papier, nous présentons une classification des patrons de sécurité pour faciliter le choix des patrons de sécurité face aux attaques. Notre classification prend en compte la non-ambiguïté, vu que nous présentons en détail la démarche de classification. Elle prend également en compte la navigabilité, vu que nous utilisons les relations inter-patrons et elle couvre 215 attaques. Nous présentons également la méthode permettant la génération de cette classification ainsi que la génération des arbres d'attaques et de défense à partir de la classification obtenue. Sur la base des informations obtenues grâce à notre classification, nous présentons une méthodologie de génération de squelettes de

cas de test permettant de tester si une application est vulnérable aux attaques, ainsi que si les conséquences des patrons de sécurité sont présents dans une application. Ceci grâce, d'une part, à l'aspect formel des ADTrees qui permet de décrire formellement le déroulement des étapes de l'attaque et les défenses associées à chaque étape avec un aspect visuellement compréhensible. D'autre part, la richesse des données extraites qui permettent de guider le concepteur dans la concrétisation des cas tests dans le contexte de son application.

Ce papier est structuré comme suit : dans la Section 2, nous présentons les notions de sécurité utilisées pour générer la classification. Ensuite, nous présentons la méthode associant les attaques et les patrons de sécurité ainsi que la génération des ADTree dans la Section 3. Puis nous présentons l'extension de notre démarche dans le but de produire des squelettes de cas de test la Section 4. Enfin, nous concluons et nous proposons quelque perspectives en Section 5.

2 Contexte

Dans le contexte de nos travaux, nous utilisons les notions des patrons de sécurité, des principes de sécurité, des patrons d'attaque et des arbres d'attaques et de défense. Un patron de sécurité est une solution générique à un problème récurrent de sécurité, caractérisée par un ensemble de propriétés structurelles et comportementales à utiliser dès phase de conception. Un patron de sécurité peut être représenté textuellement et/ou avec des diagrammes UML. Il est également caractérisé par un ensemble de points forts qui décrivent les sous propriétés du patron.

Un principe de sécurité est une propriété, une structure ou un comportement désirable d'une application qui a pour but de réduire l'impact d'une attaque [1].

Un patron d'attaque est une description des éléments et des techniques utilisées pour attaquer une application, ainsi que les challenges auxquels les concepteurs doivent faire face afin de protéger leur application [2]. La documentation d'un patron d'attaque est composée d'un ensemble de sections. Dans notre contexte, nous nous sommes limité à la section *"Attack Execution Flow"*, qui décrit les étapes de l'attaque ainsi que les techniques, les indicateurs et les mesures de sécurité associées à chaque étape.

Parmi les méthodes qui permettent la modélisation des attaques, les arbres d'attaque et de défense (Attack Defense Tree "ADTree") représentent les mesures qu'un attaquant peut prendre pour attaquer un système ainsi que les défenses qui peuvent être utilisées pour protéger le système. Ce sont des arbres étiquetés où chaque nœud permet de présenter une action d'attaque ou de défense. Chaque nœud peut être raffiné par des nœuds-enfants qui peuvent être associés par des opérateurs "ET", "OU" et "SAND". Ce dernier est un opérateur conjonctif qui impose une relation d'ordre entre les actions.

3 La classification des patrons de sécurité et la génération des ADTree

Face à l'hétérogénéité des sources des documentations des attaques et des patrons de sécurité, la liaison directe entre un patron de sécurité et une attaque est souvent ambiguë et sujette à des imprécisions. Dans le but de réduire l'ambiguïté et l'imprécision, nous proposons une méthode qui a pour but la construction d'un data-store classifiant les attaques, issues de la base CAPEC (*Common Attack Patterns Enumeration and Classification*)[2], par patrons de sécurité, issus d'un catalogue de patrons [3]. De plus, la data-store permet la génération d'un arbre d'attaque et de défense (ADTree). La méthode proposée se déroule en quatre étapes.

1- Extraction du flux d'exécution des attaques : Depuis La base des attaques CAPEC nous extrayons automatiquement l'ensemble ordonné des étapes de chaque attaque, qui est défini dans la section *"Attack Execution Flow"* de la documentation des attaques. Cette section décrit les actions que l'attaquant doit effectuer dans l'ordre ainsi que les différentes techniques qu'il doit exécuter pour satisfaire une étape, les indicateurs de succès de l'étape ainsi que des mesures de sécurité qui doivent protéger l'application. Nous avons extrait 209 étapes pour les 215 attaques sachant que les attaques peuvent partager les étapes.

2- L'extraction des principes de sécurité de chaque étape de l'attaque : Chaque étape d'attaque est caractérisée par un ensemble de mesures de sécurité *"Security Controls"* (Détectives, Préventives et Correctives). Nous en avons extrait un ensemble de 217. Nous avons opté pour une méthode d'analyse de texte avec l'outil KHCoder [1] pour agglomérer les mesures de sécurité selon leur appartenance au même ensemble de principes de sécurité. Les 217 mesures de sécurité ont été agglomérées dans 21 groupes cloisonnés. Les éléments d'un groupe dans la classification obtenue sont proches dans le sens où ils traitent de la même problématique de sécurité. Cela facilite l'extraction manuelle des principes de sécurité traités par chaque groupe de mesures de sécurité. Ainsi, nous avons lié chaque groupe de mesures de sécurité aux principes de sécurité qu'ils couvrent.

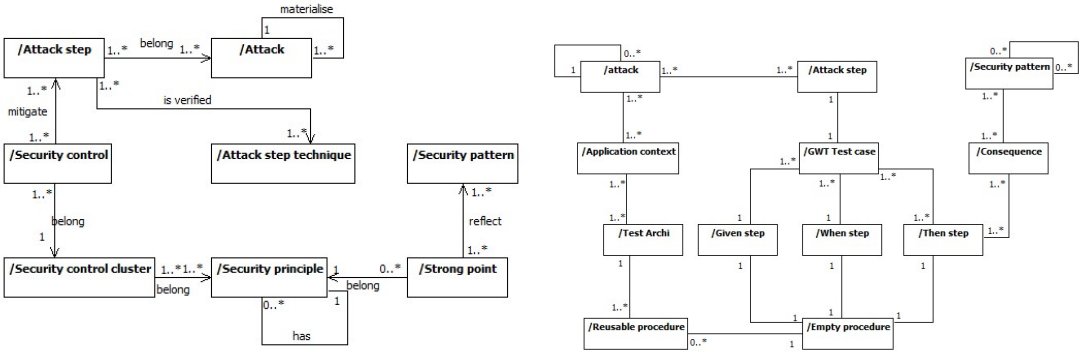
3- L'association des principes de sécurité et des patrons de sécurité : Basés sur un ensemble de travaux traitant de la notion de principes de sécurité[4, 5, 1], nous avons opté pour une organisation hiérarchique de 66 principes de sécurité afin d'améliorer leur compréhension et présenter les relations inter-principes. Un patron de sécurité est caractérisé par un ensemble de points forts définissant les raisons pour lesquelles un patron est une solution adaptée à un problème [6]. Les patrons de sécurité peuvent partager des points forts ce qui les rend similaires dans certaines situations. Chaque point fort est caractérisé par une idée élémentaire de sécurité qui peut être liée à un principe de sécurité.

4- La consolidation, l'extraction des données et la génération des ADTree : En conséquence des étapes précédentes, nous avons, grâce aux mesures de sécurité, un lien entre les étapes des attaques et les principes de sécurité, et grâce aux points forts, un

1. https ://sourceforge.net/projects/khc/

lien entre l'ensemble des patrons de sécurité et les principes de sécurité. Afin de consolider les données, nous avons développé une série de flux sous "Talend" [2] qui alimentent le data-store suivant les associations établies dans les étapes précédentes. Néanmoins, la nature abstraite des patrons impose un contrôle manuel de la mise à jour du data-store. La structure du data-store résultant est présentée dans la figure 1a. Nous avons également développé un outil qui, basé sur le data-store de données, présente les étapes, les techniques, et les patrons de sécurité sous forme de ADTree. Cela en créant un fichier XML lisible par l'outil ADTool [3] qui permet l'affichage et l'analyse des ADTree. Nous avons également utilisé ZOHO Reports [4] afin de rendre automatique l'extraction et la visualisation de la classification depuis le data-store.

4 L'extension du data-store pour le test



(a) data-store initial (b) data-store etendu

FIGURE 1 – La structure des data-stores

En plus de la classification des patrons de sécurité par attaques et la génération des ADTree, le data-store étendu, dont la structure est présenté dans la figure 1b, a pour but de générer un cas de test GWT pour chaque étape "x" d'une attaque "id". Chacun de ces cas de test est décrit par un fichier "$CAPECid_Step_x.feature$". Ce dernier présente un scénario de test avec les pré-requis *Given*, les actions *When*, et l'assertion de succès de l'attaque *Then*. Le data-store étendu contient également l'ensemble de patrons de sécurité (ainsi que les relations inter-patrons) associés à chaque étape de l'attaque. Ainsi, chaque cas de test "$CAPECid_Step_x.feature$" présente également une assertion sur la présence de chaque patron de sécurité lié à l'étape de l'attaque avec une section *Then*. Nous avons généré automatiquement l'ensemble de ces éléments en intégrant les données dans le data-store puis en générant les cas de test et les procédures commentées comme suit :

2. http ://fr.talend.com/
3. http ://satoss.uni.lu/members/piotr/adtool/
4. https ://www.zoho.eu/reports/ ?src=zoho

4.1 Intégration des données

En plus des éléments contenus dans le data-store précédent, nous avons extrait les différents **contextes** dans lesquels une attaque est applicable depuis la partie "Environments" de chaque technique d'étape d'attaque. Chaque contexte implique une **architecture de test** différente regroupant un ensemble d'outils permettant de tester les application de ce contexte.

Chaque attaque CAPEC est caractérisée par un ensemble d'éléments qu'il faut satisfaire préalablement à l'exécution de l'attaque. Les sections *"Attack Prerequisites"* et *"Resources Required"* du CAPEC décrivent ce qu'il faut préparer pour chaque attaque. Par conséquent, nous avons extrait automatiquement ces informations pour chaque attaque et nous les fournissons comme commentaire pour les procédures liées aux sections ***Given***. Chacune des étapes d'une attaque est abstraite de nature et peut être implémentée de différentes façons, détaillées dans la section *"Attack Step Techniques"*. De la même manière, nous avons extrait automatiquement ces informations pour les commentaire des procédures ***When***. Chaque étape d'attaque est décrite par un ensemble d'éléments qui permet de décider si l'attaque est bien réussie. Ces éléments, présentés dans la base CAPEC sous les sections *"Indicators"* et *"Outcomes"*, sont considérés comme des directives pour les procédures ***Then***. La présence d'un patron de sécurité dans une application peut être résumée par un ensemble de conséquences sur sa structure et son comportement. Cet ensemble d'information, présenté dans la section *"Consequences"* de la documentation des patrons de sécurité, nous permet la génération de commentaires pour les procédures ***Then*** pour les verdicts sur la présence des patrons de sécurité.

4.2 Génération des squelettes de cas test

Afin de pouvoir générer les scénarios exprimés par les cas de test nous nous sommes servis de l'aspect formel des ADTree qui permettent de décrire formellement les étapes d'une attaque ainsi que les défenses associées. Un ADTree T_{Att} peut être exprimé avec un ensemble d'ADTerms $t(T_{Att})$ de la forme $\vee^p(t_1, \dots t_k)$, avec t_i un ADTerm qui peut également exprimé sous cette forme, ou sous la forme $c^p(\overrightarrow{\wedge}^p(t_1, \dots t_n), \wedge^o(sp_1, \dots, sp_m))$, avec $\overrightarrow{\wedge}^p(t_1, \dots t_n)$ une séquence d'étapes d'attaque et $\wedge^o(sp_1, \dots, sp_m)$ une conjonction de patrons de sécurité.

Chaque attaque est donc un scénario qu'on va appeler séquence basique d'attaque et de défense (Basic Attack Defence sequence BADSeq). Un *BADSeq* est un ADTerm de la forme $c^p(\overrightarrow{\wedge}^p(Att_1, ;Att_n), \wedge^o(sp_1, \dots, sp_m))$ et qui exprime le séquencement des étapes réalisées avec $\overrightarrow{\wedge}^p(Att_1, ;Att_n)$ ainsi qu'une conjonction de patrons de sécurité. Nous soulignons que le concepteur doit choisir une conjonction de patrons de sécurité parmi l'ensemble de combinaisons possibles de patrons présenté dans l'ADTree.

Chaque *BADSeq* est la base pour la génération d'un cas de test *GWT*. Ainsi, depuis l'arbre d'attaque de chaque attaque *id* nous générons un *BADSeq* pour chacune de ces étapes x sous la forme d'un scénario alternant les actions d'attaque et de défense. Chacun de ces scénarios est présenté par un cas de test "$CAPECid_Step_x.feature$" avec une section *Given*, une section *When* et plusieurs sections *Then* exprimant les verdicts sur

le succès de l'attaque et la présence des conséquences de chacun des patrons de sécurité présenté dans la conjonction $\wedge^o(sp_1, \ldots, sp_m))$ des patrons de sécurité. Chaque section *Given*, *When* ou *Then* est associée à un squelette de procédure portant le même nom que la section. Chacune de ces procédures est générée et complétée automatiquement avec un ensemble de directives, placées sous forme de commentaires, aidant le concepteur a coder les cas de test et cela en utilisant les données stockées dans le data-store lors de l'intégration des données.

A l'aide d'outils de "*penetration testing*" dans le contexte des applications web, nous avons codés 28 cas de test ainsi que les verdicts sur la présence des conséquences de 11 patrons de sécurité. Ces cas de test peuvent être réutilisable de façon générique sur les applications web.

5 Conclusion

Nous avons présenté une classification associant les patrons de sécurité et les attaques. Nous avons également présenté la méthodologie permettant une reproduction systématique de la classification ainsi que la génération d'un ADTree pour chaque attaque. Notre méthodologie a permis la construction d'un data-store contenant 215 attaques, 209 étapes, 217 mesures de sécurité et 448 techniques d'attaque, qui est disponible en ligne [7]. Nous avons présenté également une méthodologie de génération de squelettes de cas de test permettant de tester les attaques ainsi que les conséquences observables des patrons de sécurité et les directives permettant aux concepteurs de concrétiser ces tests. Les cas de test que nous produisons sont réutilisables.

Au moyen terme nous travaillons sur l'utilisation du data-store de données et des cas de test dans une méthode de test bien fondée.

Références

[1] J. Meier, "Web application security engineering," *Security & Privacy, IEEE*, vol. 4, no. 4, pp. 16–24, 2006.

[2] Mitre corporation, "Common attack pattern enumeration and classification, url :https ://capec.mitre.org/," 2015.

[3] K. Yskout, T. Heyman, R. Scandariato, and W. Joosen, "A system of security patterns," 2006.

[4] J. Viega and G. McGraw, *Building Secure Software : How to Avoid Security Problems the Right Way, Portable Documents.* Pearson Education, 2001.

[5] J. Scambray and E. Olson, *Improving Web Application Security.* 2003.

[6] C. Bouhours, *Détection, Explications et Restructuration de défauts de conception : les patrons abîmés.* PhD thesis, l'Université Toulouse III – Paul Sabatier, 2010.

[7] "Security pattern classification "http ://regainia.com/research/database.html"."