

Secure Coding en Java

Table des matières

Chapitre 1

Rappels de sécurité

Objectifs du chapitre	2
Agenda	3
Principes de sécurité	4
Réduction des risques	5
Nécessités de sécurité	7
Répondre aux nécessités de sécurité	8
Chiffrement	10
Résistance du chiffrement	11
Chiffrement à clé symétrique	12
Chiffrement à clé publique	13
Authentification à clé publique	14
Empreinte numérique.....	15
Signature numérique.....	16
Codes d'authentification de message	18
Codage : Bonnes pratiques	19
PCI-DSS	20
Payment Card Industry (PCI) Data Security Standard (DSS)	21
Résumé des exigences PCI-DSS	22
La sécurité dans le cycle de vie logiciel	24
Cout de la correction d'une faille.....	25
Disposer d'une gouvernance sécurité logicielle	26
Les grandes étapes de sécurité dans le cycle de vie logiciel.....	27
Concevoir des applications sécurisées	32
Tester les applications	33
La sécurité logicielle appliquée au PCI-DSS.....	34

Chapitre 2

Codage sécurisé d'une application

Agenda	2
Quelques grands principes	3
Architecture de sécurité	4
Exemple de défense en profondeur	5
Contrôles	6
Geoportail	8
Google Maps.....	9
Obscurité != Sécurité	10
Fail securely.....	11
Prévoir l'imprévisible	12
Authentification	13
Implémenter une bonne stratégie de mot de passe	14
L'authentification multi-facteurs.....	16
Implémenter une bonne stratégie globale.....	17
Un bon exemple de stratégie	18
Comment s'y prendre ?.....	19
Gestion des sessions applicatives	20
Gestion des sessions.....	21
Gestion des sessions en détails.....	22
Contrôle d'accès	23
Contrôle d'accès == Habilitations.....	24
Prérequis	25
Principes de contrôle d'accès	26
Validation des données	28
Validation d'entrée	29
Frontière de validation	30
Encodage des données de sortie.....	31
Éléments cryptographiques	32
Chiffrement	33

Chapitre 2

Codage sécurisé d'une application (suite)

Gestion des erreurs et de la journalisation.....	34
Gestion des erreurs	35
Comment logger	36
Protection de l'accès aux données	37
Protection des données	38
Sécurité du transport des données	39
Sécurisation de la communication	40
Configuration de la sécurité	41
Sécuriser les interfaces d'administration.....	42
Revue des configurations	43
Code Malveillant et sécurité interne au logiciel.....	44
Code malveillant	45
Sécurité interne du logiciel	46
Complexité du code	47
Qualité de code.....	48
CERT Secure Coding	49
Notes	50

Chapitre 3

Structuration du code, déclarations, initialisation

Structuration	2
Structuration du code.....	4
Déclaration de variables	5
Déclaration des variables.....	6
Constantes.....	7
Initialisation	9
Surcharge	12
Sérialisation	14
Déclaration des variables.....	15
Notes	16

Chapitre 4

Les Expressions

Valeurs de retour	2
Comparaisons.....	4
Opérateurs de comparaison	7
Casts.....	9
Parenthèses.....	11
Accès aux tableaux	12
Expressions conditionnelles.....	14
Rappels sur les opérandes	16
Gestion des expressions.....	17
Notes	18



Chapitre 5

La portée

Portée des variables	2
Accessibilité	3
Réutilisation des termes.....	5
Expositions	6
Suppression des Warnings	8
Exhauste la RAM.....	9
La portée.....	10



Chapitre 6

Entiers, réels...

Integer Overflow.....	2
Débordements et opérations.....	3
Intervalle et cast.....	4
Reste	5
Retour de valeurs.....	6
Décalages	7
Précision et flottants.....	8
Attention aux expressions arithmétiques	9
Comparaison avec NaN.....	11
Flottant et boucle	12
BigDecimal.....	13
Chaînes	14
Entiers, Réels.....	15
Notes	16

Chapitre 7

Les Objets

Accès aux membres	2
Variables statiques.....	4
Comparaisons.....	5
Ramasse-Miettes	6
Objets	7
Notes	8



Chapitre 8

Entrées / Sorties

Encodage de sortie	2
Canonicalisation.....	4
Fermer les connecteurs	6
Permission d'ouverture	10
Entrées/Sorties	11
Notes	12



Chapitre 9

Valider les données

Données utilisateur	2
Normalisation	3
Encodage de sortie	5
Injection de commandes	6
Injection SQL	7
Injection XML	9
Injection XPath.....	11
Injection LDAP	13
Injection de code.....	15
URI.....	17
Validations	18

Chapitre 10

Gestion des erreurs et journalisation

Prévenir les exceptions lors de logs.....	2
Ne pas finir de façon abrupte	3
Perte d'informations	4
Exposition de données sensibles.....	6
Logs/Erreurs	7
Notes	8



Chapitre 11

La revue de code

Agenda	2
L'exposition a une vulnérabilité	3
Coût de la correction d'une faille.....	4
Scope – Approche Black/White/Gray Box.....	5
La revue de code : Définition	6
Méthodologie d'analyse de code	7
Secure Code review process	8
Phases.....	9
Scope.....	10
Modéliser les processus	11
Exemple 1	12
Exemple 2	13
Modéliser les menaces	14
Deux types de menaces	15
Catégoriser les attaques	16
Exemples d'attaques catégorisées	17
Contrôles et menaces	18
Prioriser / calculer le risque.....	19
Calculer le risque	20
Identifier les contre-mesures.....	21
Réduire l'impact	22
Quelle est la meilleure approche.....	23
Quelques fonctions Java à auditer.....	24
Entrées/Sorties	25
Servlets	26
XSS & Reponse Splitting	27
Redirection.....	28
SQL.....	29
Les Outils	30

Chapitre 12

Résumé du cours

Résumé du cours..... 2

Secure Coding en Java

Introduction

Objectifs du cours

Dans ce cours, nous allons :

- Présenter les bonnes pratiques permettant de sécuriser le développement d'une application**
- Présenter les erreurs d'utilisation de code**
- Fournir les éléments nécessaire à une revue de code sécurité**

Contenu du cours

Introduction et vue d'ensemble

- | | |
|--------------------|--|
| Chapitre 1 | Rappels de sécurité |
| Chapitre 2 | Codage sécurisé d'une application |
| Chapitre 3 | Structuration du code, déclarations, initialisation |
| Chapitre 4 | Les Expressions |
| Chapitre 5 | La portée |
| Chapitre 6 | Les entiers, les réels... |
| Chapitre 7 | Les objets |
| Chapitre 8 | Les entrées/sorties |
| Chapitre 9 | La validation des données |
| Chapitre 10 | La gestion des erreurs, la journalisation |
| Chapitre 11 | Revue de code sécurité |
| | Évaluation du cours |

Chapitre 1

Rappels de sécurité

Objectifs du chapitre

- Rappeler les principes de bases de chiffrement et de sécurité
- Introduire la sécurité dans le cycle de vie logiciel
- Faire le parallèle entre la sécurité logicielle et le PCI-DSS

Agenda

- **Principes de Sécurité**
- **PCI-DSS**
- **La sécurité dans le cycle de vie logiciel**

Rappels de sécurité

► **Principes de sécurité**

PCI-DSS

La sécurité dans le cycle de vie logiciel

Réduction des risques

- La sécurité des applications dépend de l'évaluation et la réduction des risques

Risque ≈ gravité x menace x valeur

- Tout risque d'endommagement ou de perte est considéré comme une menace
 - Gravité : impact maximal d'une vulnérabilité exploitée
 - Menace : probabilité d'une attaque
 - Valeur : coût de la perte ou de la compromission
- Les risques doivent être réduits à un niveau acceptable
 - Ce niveau diffère d'une organisation à une autre et pour chaque application

« L'évaluation de la sécurité ne consiste pas à éliminer les risques, mais plutôt à trouver un équilibre entre risques nécessaires et utiles en fonction de leurs avantages pour l'entreprise. »

– Peter Coffee, ancien rédacteur en chef d'eWeek Technology

Réduction des risques (suite)

- Trouver le bon équilibre entre les risques nécessaires et utiles d'un côté, et les bénéfices commerciaux de l'autre
 - Pourquoi les banques offrent-elles un service bancaire en ligne ?
 - Pourquoi les immeubles ont-ils des fenêtres ?
- Ce cours montrera comment diminuer la gravité des risques
 - En éliminant ou en réduisant les vulnérabilités



Nécessités de sécurité

- Au moment de la création d'une application, les objectifs suivants doivent être atteints :
 - Confidentialité
 - Seuls les utilisateurs autorisés ont accès à l'information
 - Exemple : *chiffrement*
 - Intégrité
 - Les informations ne sont pas modifiées pendant leur transfert, stockage ou récupération
 - Exemple : sommes de contrôle et *signatures numériques*
 - Disponibilité
 - Les utilisateurs autorisés ont accès aux informations quand ils les demandent
 - Exemple : bonnes pratiques de codage mettant en œuvre *authentification* et *autorisation* pour empêcher les abus
- En anglais, on parle de modèle de sécurité CIA pour ***Confidentiality, Integrity, Availability***

Répondre aux nécessités de sécurité

Plusieurs techniques permettent de répondre à ces besoins de sécurité :

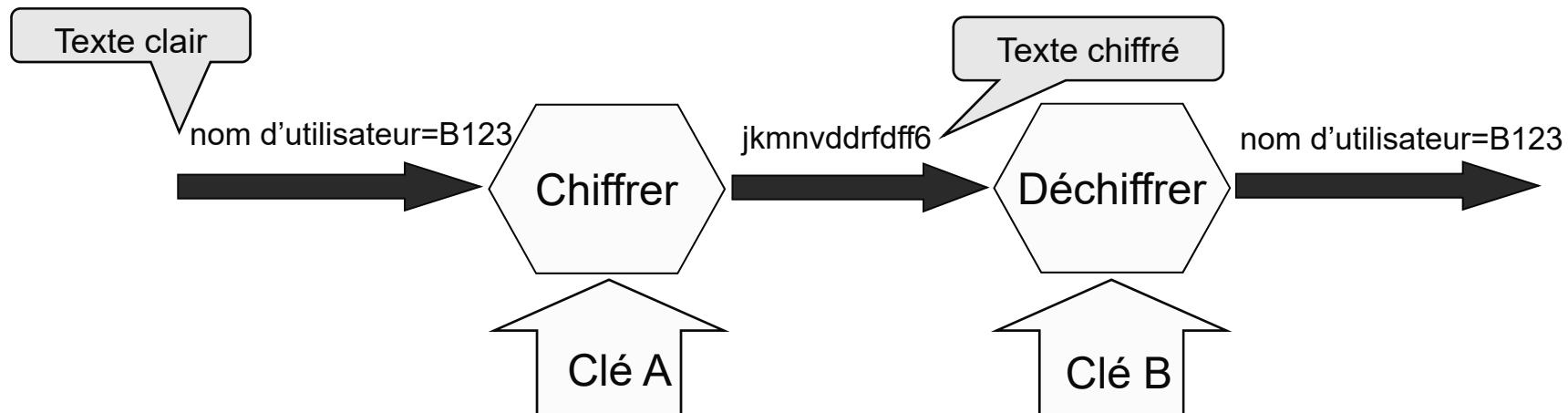
- **Authentification**
 - Vérifier l'identité de l'utilisateur
- **Autorisation**
 - Déterminer si l'utilisateur est autorisé à accéder à une ressource
- **Chiffrement**
 - Le texte clair est converti en texte chiffré difficile ou impossible à décoder sans clé
- **Signatures numériques**
 - Vérifier l'intégrité d'un message
 - Identifier l'expéditeur

Répondre aux nécessités de sécurité (suite)

- **La sécurité des applications exige aussi :**
 - Des connaissances en chiffrement
 - Une bonne expérience de la programmation
 - Configuration et gestion
 - Installation de correctifs et permissions

Chiffrement

- Toute information transmise à travers un réseau peut être interceptée
 - Tout comme une conversation téléphonique peut être écoutée
- Le chiffrement évite que les informations transmises puissent être lues au moyen d'une écoute électronique
 - Aide également à préserver l'intégrité des données
 - Implique d'employer une clé pour obscurcir les données
 - Une clé est nécessaire pour déchiffrer les données



- Deux principaux types de chiffrement
 - Clé symétrique : clé A = clé B
 - Clé publique (asymétrique) : clé A ≠ clé B

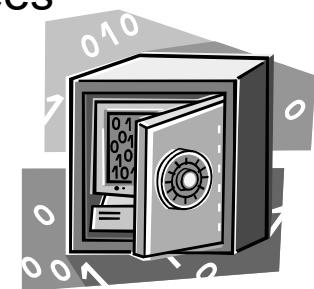
Résistance du chiffrement

- **Le degré de fiabilité du chiffrement dépend de la fiabilité de la clé qui a servi à chiffrer les données**

- Toute personne compromettant la clé peut déchiffrer les données
 - La clé est comparable au code d'un coffre-fort

- **Les clés de chiffrement sont une simple séquence de bits**

```
11011001100101010101010110101101101001
```



- **Une des méthodes des pirates consiste à essayer toutes les clés possibles**

- Plus la clé est longue, plus cette méthode est laborieuse
 - Le nombre de clés possibles augmente exponentiellement avec l'augmentation de la longueur de la clé
 - Le nombre de clés = 2^N , N étant la longueur de la clé

- **Il est impossible d'essayer toutes les clés si la clé est longue**

- $2^{128} = 340,282,366,920,938,463,463,374,607,431,768,211,456$

Chiffrement à clé symétrique

- **Emploie la même clé pour le chiffrement et le déchiffrement des données**
 - Les deux parties doivent connaître la même clé
 - Très important pour la protection de la clé
 - Nécessite une relation préalable entre les parties
- **Algorithmes de clés symétriques populaires**
 - DES (Data Encryption Standard)
 - 56 bits
 - Triple DES
 - 112/168 bits
 - AES (Advanced Encryption Standard)
 - 128-256 bits
 - RC2, RC4, RC5
 - Jusqu'à 2048 bits

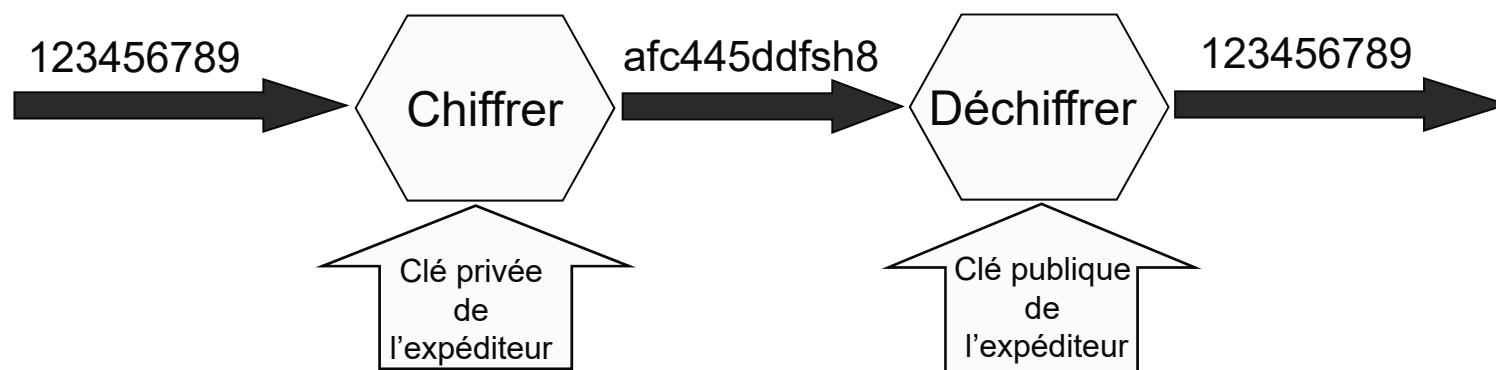
Chiffrement à clé publique

- **Permet une communication sécurisée sans relation préétablie**
 - Peut également servir à des fins d'authentification
- **Toutes les parties en communication doivent créer une paire de clés publique et privée**
 - La clé publique doit être mise à la disposition de l'autre partie
 - La clé privée doit rester privée
 - Les clés sont mises en relation
 - Les données chiffrées avec une clé publique sont déchiffrables uniquement avec la clé privée correspondante
 - Les données chiffrées avec une clé privée sont déchiffrables uniquement avec la clé publique correspondante
 - Une fois la clé publique donnée, il est pratiquement impossible d'en produire une autre
- **RSA est l'algorithme de chiffrement à clé publique le plus populaire**
- **Inconvénient : lenteur du chiffrement et du déchiffrement**
 - Jusqu'à 1000 fois plus lents qu'en cas de chiffrement à clé symétrique

RSA = Rivest, Shamir et Adleman

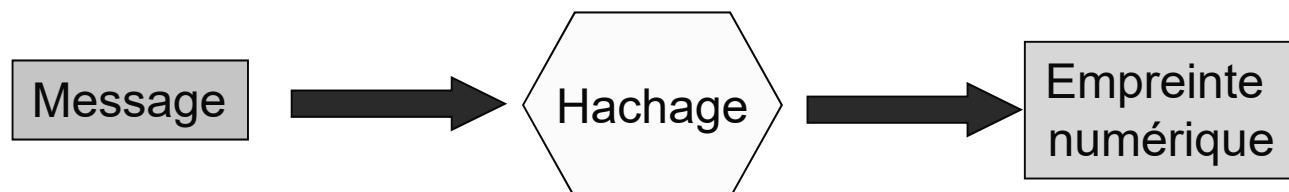
Authentification à clé publique

- **Comment prouver son identité au travail ou dans un aéroport ?**
 - En présentant un document d'identification
 - Badge de l'entreprise, permis de conduire, passeport, etc.
 - Ce type de moyen serait très utile pour une authentification électronique
- **Le chiffrement à clé publique peut servir à prouver une identité**
 - L'expéditeur chiffre le message avec une clé privée
 - Le destinataire obtient la clé publique de l'expéditeur et déchiffre le message
 - Il vérifie que le message a été chiffré avec la clé privée correspondante
 - Il vérifie que l'expéditeur a employé la bonne clé privée



Empreinte numérique

- **Mécanisme destiné à préserver l'intégrité des données**
 - Un algorithme de hachage sert à calculer une chaîne de caractères de longueur fixe (semblable à une somme de contrôle) pour tout fichier ou message
 - La chaîne de caractères de longueur fixe est appelée *empreinte numérique* ou *condensé de message*
 - On parle aussi de *hachage à sens unique*
 - Une fois l'empreinte donnée, il est impossible de recréer le message d'origine
- **Algorithmes d'empreintes numériques courants**
 - MD5 : sortie en 128 bits
 - SHA-1 : sortie en 160 bits
 - SHA-256 : sortie en 256 bits

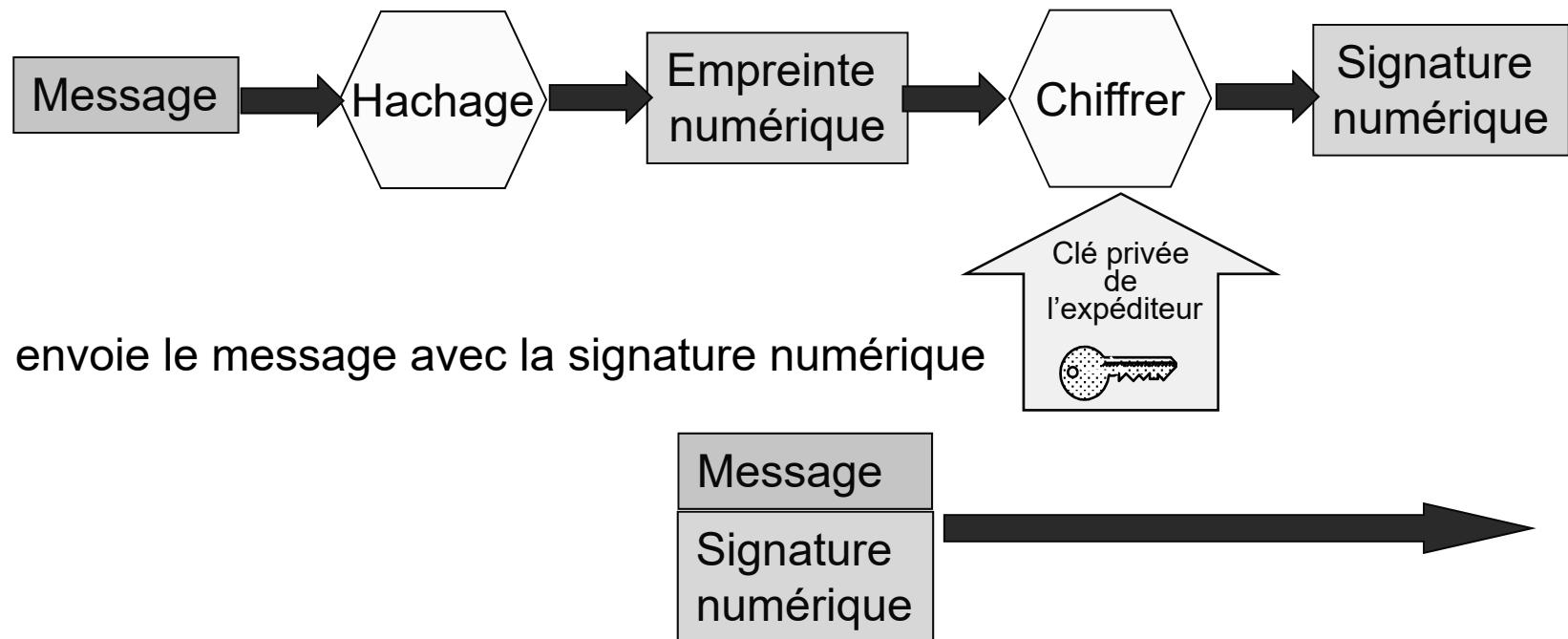


MD5 = Message Digest 5

SHA = Secure Hash Algorithm

Signature numérique

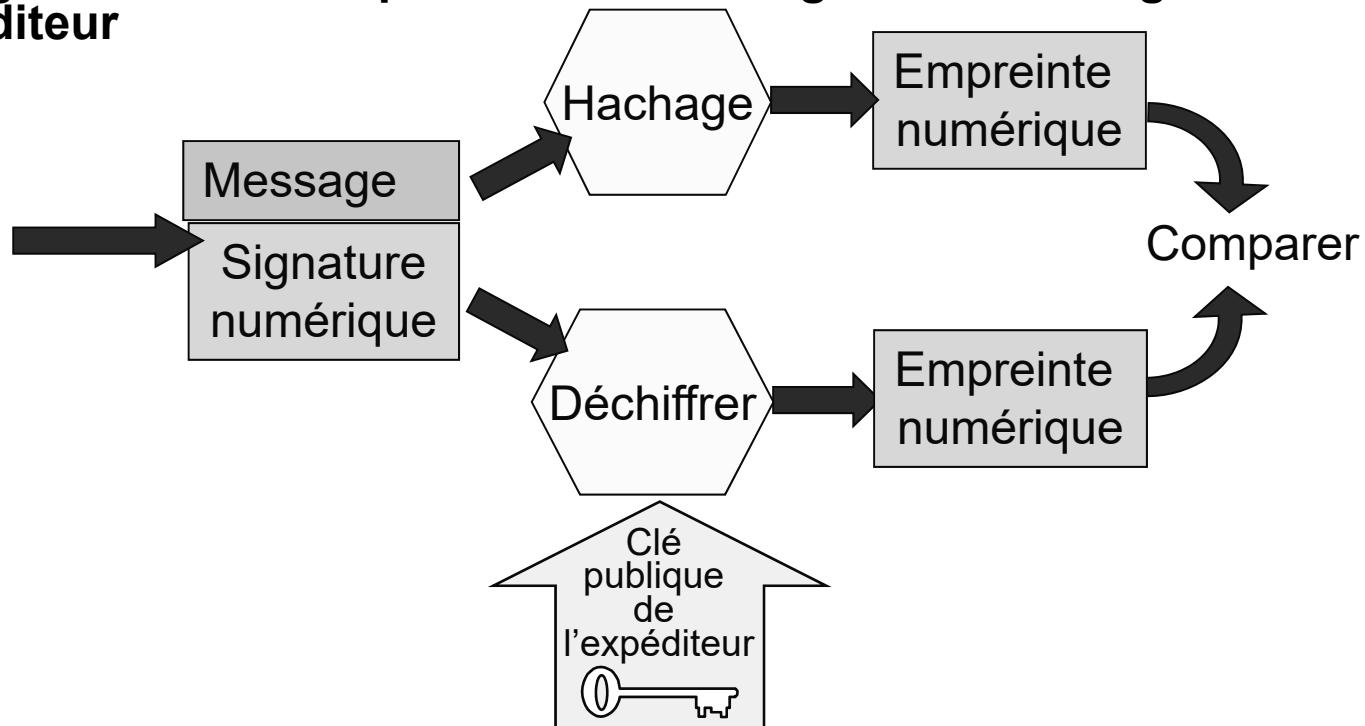
- **Empreinte numérique chiffrée par chiffrement à clé publique**
 - L'expéditeur chiffre l'empreinte numérique avec sa clé privée



- **Les signatures numériques ne préservent en aucun cas la confidentialité du message**
 - Dans cet exemple, le message n'est pas chiffré
 - Si nécessaire, le message doit être chiffré séparément

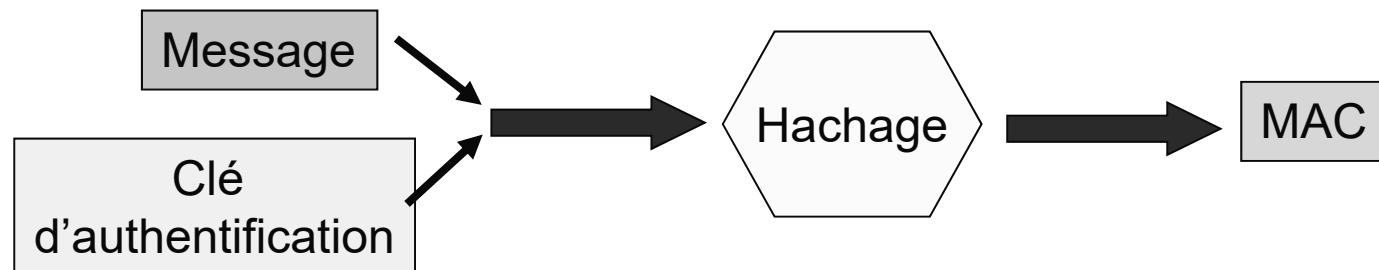
Signature numérique (suite)

- **Le destinataire vérifie l'intégrité du message**
 - Déchiffre la signature numérique pour obtenir l'empreinte numérique du message d'origine
 - Au moyen de la clé publique de l'expéditeur
 - Recalcule l'empreinte numérique et compare les valeurs des deux empreintes
- **Les signatures numériques vérifient l'intégrité du message et authentifient l'expéditeur**



Codes d'authentification de message

- **Les signatures numériques sont relativement lentes**
 - En raison du chiffrement à clé publique
- **Le code d'authentification de message ou MAC (*Message Authentication Code*) représente une alternative courante**
 - Une valeur de clé est incluse dans le message avant hachage
 - Aucun algorithme de chiffrement n'est utilisé
 - Les deux parties doivent connaître la même clé
 - Exactement comme pour un chiffrement à clé symétrique
- **MAC garantit l'intégrité du message et authentifie l'expéditeur**
 - L'expéditeur et le destinataire doivent tous deux connaître la valeur correcte de la clé



Codage : Bonnes pratiques

- **Respecter les normes de codage**
 - Nommage, formatage, commentaires, etc.
 - Améliore la lisibilité et la compréhension
 - Réduit les problèmes de maintenance
- **L'examen des codes peut aider à repérer des défaillances**
 - Un examen des codes y compris informel peut être très utile
- **Il faut consulter les directives de sécurité à plusieurs reprises pendant le développement**
 - Par exemple : validation des entrées, traitement des erreurs, autorisation
 - Pour chaque méthode, classe, etc.
 - Ne pas essayer de vérifier toute l'application en une fois
 - OWASP publie un *Guide pour construire des applications Web sécurisées* (*Guide to Building Secure Web Applications*)
 - Le SANS / MITRE
 - Le CERT

Rappels de sécurité

Principes de sécurité

► **PCI-DSS**

La sécurité dans le cycle de vie logiciel

Payment Card Industry (PCI) Data Security Standard (DSS)

- **Norme de sécurité des données du secteur des cartes de paiement.** A été développée pour optimiser la sécurité des données des titulaires de cartes de paiement et renforcer l'adoption de mesures de sécurité des données cohérentes au niveau mondial
 - Les exigences PCI sont destinées à protéger les informations des cartes de crédit
 - Réduire le vol
 - Augmenter la confiance des clients
- **L'idée de la norme PCI-DSS est d'implémenter des bonnes pratiques de sécurité**
 - Suivre des bonnes pratiques de sécurité mènera naturellement vers une conformité PCI
 - Toutefois, chercher à être en conformité avec la norme PCI exclusivement ne favorisera pas un environnement sécurisé

Résumé des exigences PCI-DSS

- **Créer un réseau sécurisé et en assurer la maintenance**
 - Exigence 1 : installer une configuration de pare-feu pour protéger les données des titulaires de carte de paiement et en assurer la maintenance
 - Exigence 2 : ne pas utiliser les mots de passe système et autres paramètres de sécurité par défaut définis par le fournisseur
- **Protéger les données des titulaires de carte de paiement**
 - Exigence 3 : protéger les données des titulaires de carte stockées
 - Exigence 4 : chiffrer la transmission des données des titulaires de carte de paiement sur les réseaux publics ouverts
- **Disposer d'un programme de gestion des vulnérabilités et en assurer la maintenance**
 - Exigence 5 : utiliser des logiciels antivirus et les mettre à jour régulièrement
 - Exigence 6 : développer des systèmes et des applications sécurisés et en assurer la maintenance

Résumé des exigences PCI-DSS (suite)

- **Implémenter des mesures de contrôle d'accès strictes**
 - Exigence 7 : restreindre l'accès aux données des titulaires de carte aux seules personnes qui ont besoin d'en avoir connaissance
 - Exigence 8 : attribuer un identifiant unique à chaque personne disposant d'un accès informatique
 - Exigence 9 : restreindre l'accès physique aux données des titulaires de carte
- **Surveiller et tester les réseaux régulièrement**
 - Exigence 10 : suivre et surveiller tous les accès aux ressources réseau et aux données des titulaires de carte
 - Exigence 11 : tester régulièrement les systèmes et processus de sécurité.
- **Disposer d'une politique de sécurité et la mettre à jour**
 - Exigence 12 : disposer d'une politique portant sur la sécurité des informations et la mettre à jour

Rappels de sécurité

Principes de sécurité

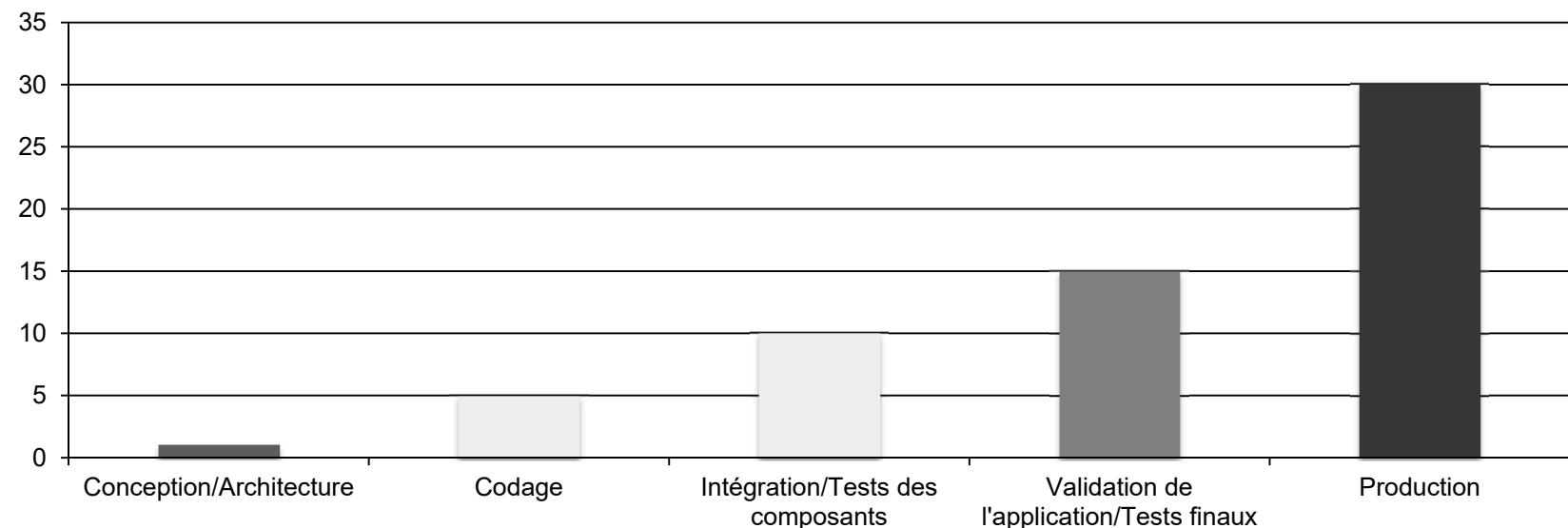
PCI-DSS

► **La sécurité dans le cycle de vie logiciel**

Cout de la correction d'une faille

- Plus tôt la faille est découverte dans le cycle de développement, plus le cout de correction est réduit.

Cout relatif pour la correction en fonction du moment de détection



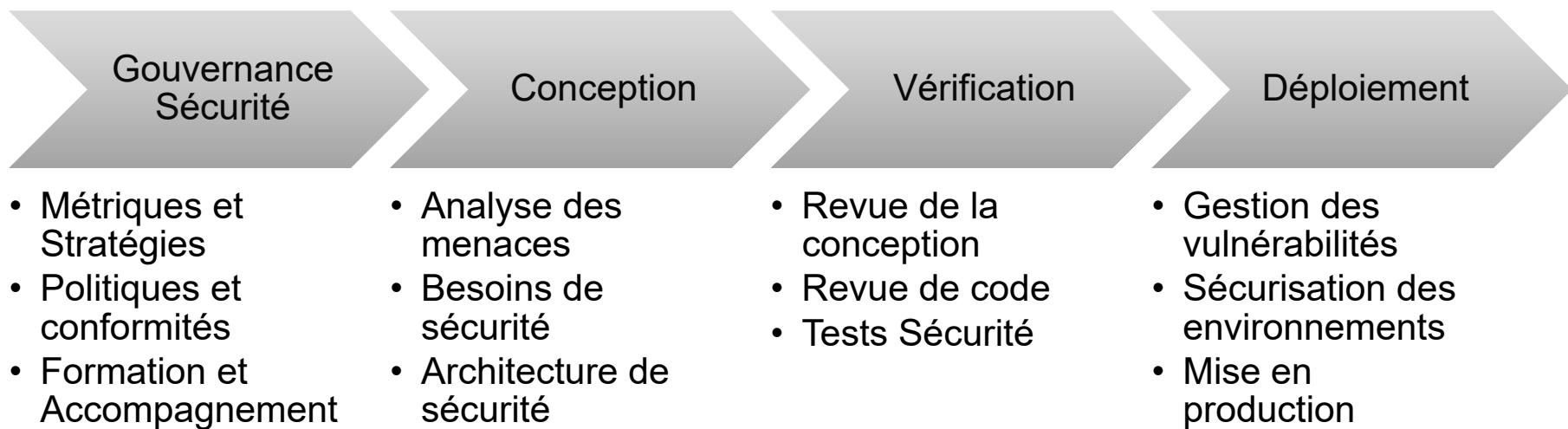
Source : NIST

Disposer d'une gouvernance sécurité logicielle

- Dictée par les normes ou les standards :
 - ISO27034 – Sécurité Applicative (*en cours de finalisation*)
 - ISO 27000 chapitre 12 – Générique sur un SMSI
 - PCI-DSS
 - NIST SP 800-64 – *Recommandation pour les US*
 - Microsoft SDL – *Propriété de Microsoft*
- Former les développeurs à la sécurité logicielle

Les grandes étapes de sécurité dans le cycle de vie logiciel

La sécurité logicielle passe par 4 fonctions importantes dans le cycle de vie logicielle



Ces pratiques sécurité se découplant ensuite en activités sécurité telles que :

- **Formation des développeurs au SecureCoding**
- **Modélisation des attaques**
- **Revue du code sécurité**
- **Test d'intrusion**
- **Mise en place d'un « bug bounty » programme**

Les grandes étapes de sécurité dans le cycle de vie logiciel (suite)



- **Métriques et Stratégies**
 - Feuille de route stratégique
 - Mesurer les risques pesant sur les données et applications et adapter la réponse
 - Disposer d'indicateurs pertinents sur la sécurité des développements logiciels.
- **Politiques et conformités**
 - Connaître les éléments de gouvernance et de conformité
 - Référentiel de sécurité
 - Audit et vérification de la conformité
- **Formation et Accompagnement**
 - Disposer de compétences formées au développement sécurisé
 - Former les intervenants sur leur rôle en sécurité
 - Certification des compétences en sécurité

Les grandes étapes de sécurité dans le cycle de vie logiciel (suite)

Gouvernance
Sécurité

Conception

Vérification

Déploiement

- **Analyse des menaces**
 - Identifier et comprendre les menaces
 - Améliorer la réponse aux menaces par une granularité plus forte de l'évaluation des menaces
- **Besoins de sécurité**
 - Insérer la sécurité explicitement lors de la phase d'exigences fonctionnelles
 - Aligner les besoins en sécurité en fonction des menaces
 - Audit et vérification des exigences sécurité
- **Architecture de sécurité**
 - Ajouter des éléments de sécurité de manière pro-active
 - Suivi des éléments de sécurité fournis, et éléments d'architecture sécurisés
 - Framework de sécurité et conformité

Les grandes étapes de sécurité dans le cycle de vie logiciel (suite)

Gouvernance
Sécurité

Conception

Vérification

Déploiement

- **Revue de la conception**
 - Mise en place de revues ad-hoc
 - Mise en place de revues formelles
 - Mise en place de revue détaillées de la conception
- **Revue de code**
 - Vérification basique des vulnérabilités dans le code
 - Mise en place d'une automatisation des revues de codes
 - Utilisation effective et globale des revues de code
- **Tests Sécurité**
 - Processus de validation de sécurité par des tests d'intrusion
 - Processus de tests sécurité au sein du cycle projet
 - Validation du niveau de sécurité avant mise en production

Les grandes étapes de sécurité dans le cycle de vie logiciel (suite)

Gouvernance
Sécurité

Conception

Vérification

Déploiement

- **Gestion des vulnérabilités**
 - Réponse aux incidents
 - Processus de réponses aux incidents
 - Analyse des incidents
- **Sécurisation des environnements**
 - Disposer d'environnement sécurisés
 - Maintenir des environnements à jour
 - Vérification des environnements
- **Mise en production**
 - Documentation des projets
 - Documentation des mises à jour
 - Assurance et/ou Intégrité des logiciels

Concevoir des applications sécurisées

- Adopter un processus pour s'assurer que les applications ne contiennent pas de failles
 - L'appliquer tout au long du cycle de développement
 - Commencer par les conditions requises et l'architecture
 - Avant d'écrire le moindre code
 - Continuer par la production et la maintenance
 - Le code le plus sûr ne le sera plus en cas d'installation non sécurisée
 - Le code le plus sûr ne le sera plus en cas de mauvaise maintenance
- Les frameworks et les outils de développement modernes sont une aide utile
 - Les éditeurs incorporent de nouvelles fonctions de sécurité
 - Utiliser les dernières versions
 - Résolvent de nombreux problèmes de sécurité sans effort de la part du développeur

Tester les applications

- **Des outils spécialisés de test et d'évaluation aident à identifier les failles et les vulnérabilités**
 - Traitent les vulnérabilités de type connu
 - Ne peuvent pas trouver tous les problèmes
- **Il est nécessaire de tester et d'inspecter les codes manuellement**
 - Rien ne remplace le zèle du développeur
 - Et un examen du code en binôme
 - Nécessite une bonne compréhension des principes de sécurité et des technologies de développement.

La sécurité logicielle appliquée au PCI-DSS

- **Explicitement dans le PCI-DSS est cité :**
 - Modélisation des attaques
 - Revue de code (6.3)
 - Tests sécurité
 - Secure Coding (6.3 && 6.5)

Il est donc impératif de mettre en place un cycle de développement logiciel sécurisé pour le développement d'application PCI

Chapitre 2

Codage sécurisé d'une application

Agenda

- Présentation des principes de codage sécurisé
- Architecture de sécurité
- Gestion de l'authentification et des habilitations
- Validation de données
- Gestion des erreurs
- Sécurité du stockage et des communications
- Protection interne du logiciel
- Secure Coding

Quelques grands principes

- La Défense en profondeur d'une application est obligatoire
- Mettre en place le moins de privilèges dans l'application/fonction métier est souvent la meilleure solution
- Mettre en place une séparation des pouvoirs/privilèges plus importantes que ce que peut penser l'utilisateur
- Rappelez-vous que l'application doivent répondre aux besoins des utilisateurs en priorité malgré tout !
- Garder le code petit et simple !

KISS : Keep it Short and Simple

Codage sécurisé d'une application

► Architecture de sécurité

Fail securely

Authentification

Gestion des sessions applicatives

Contrôle d'accès

Validation des données

Éléments cryptographiques

Gestion des erreurs et de la journalisation

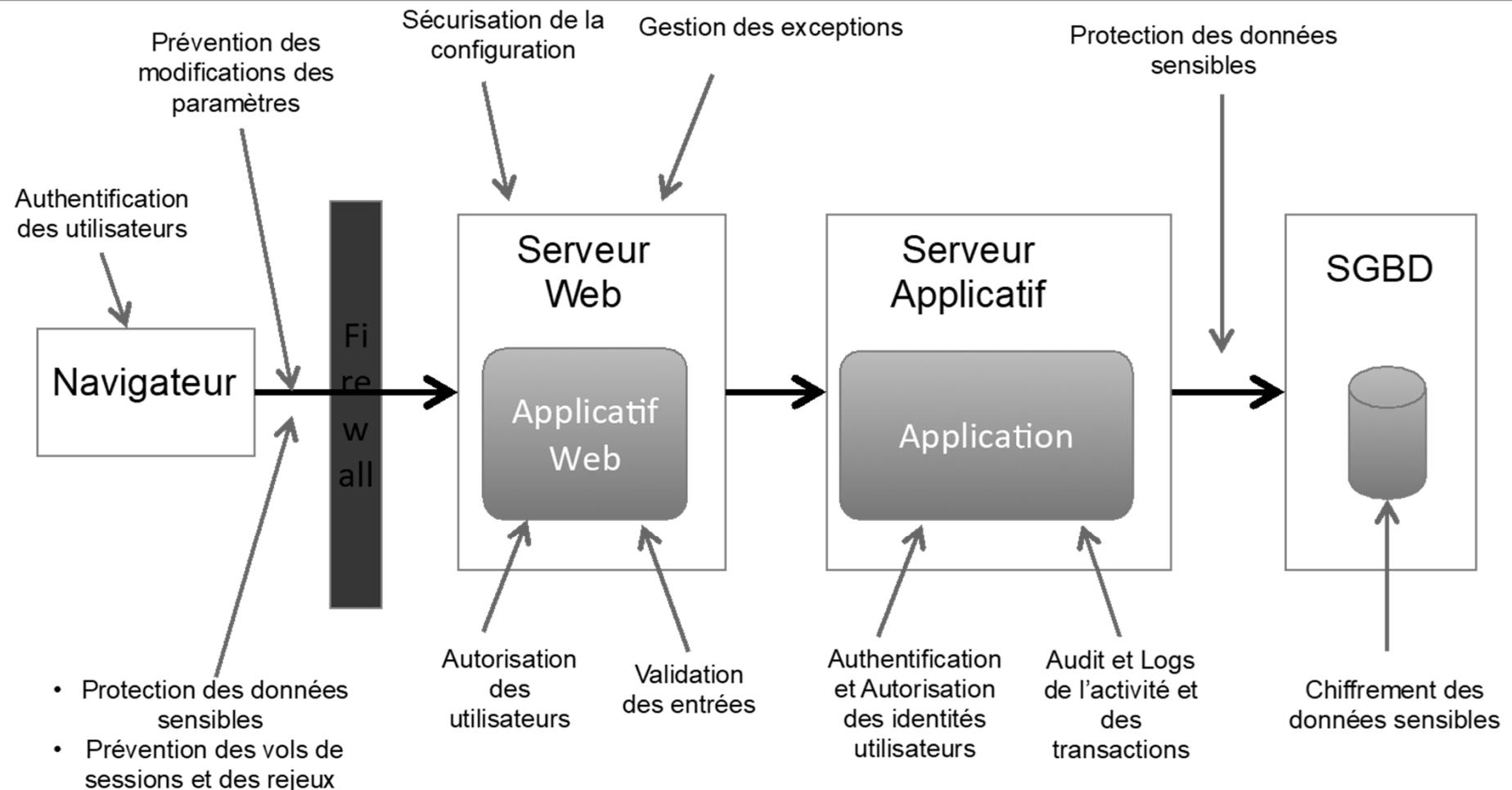
Protection de l'accès aux données

Sécurité du transport des données

Configuration de la sécurité

Code Malveillant et sécurité interne au logiciel

Exemple de défense en profondeur



Contrôles

- **Toute application doit disposer d'un minimum de contrôles. Ceux-ci doivent répondre aux critères suivants :**
 - Être simple
 - Utiliser correctement
 - Fonctionnels
 - Présent et disponible pour toute partie de l'application

Une mauvaise connaissance, un oubli, une mauvaise utilisation d'un contrôle par un développeur rend l'application vulnérable

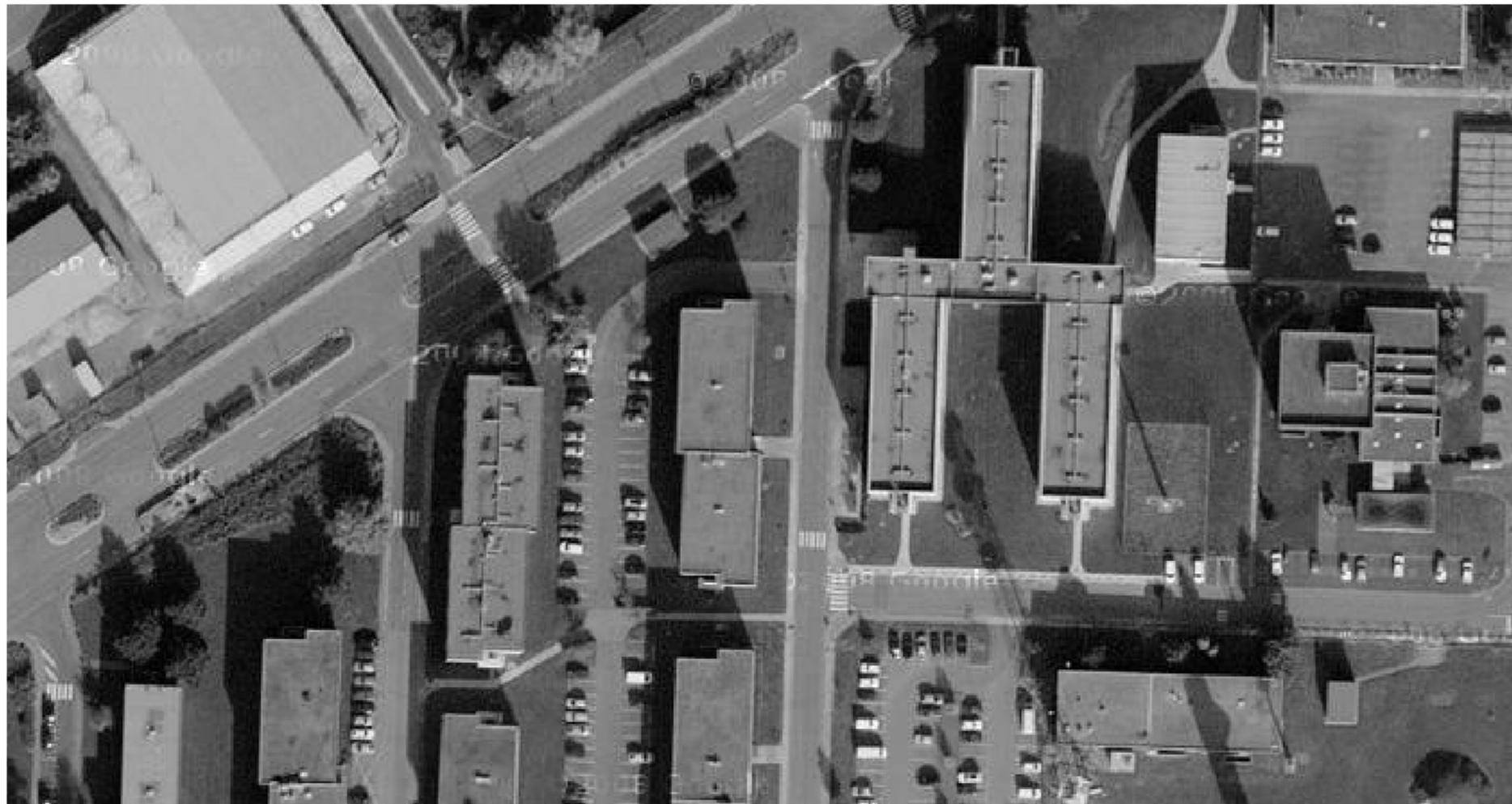
Contrôles (suite)

- **Toute application doit implémenter au minimum les contrôles suivants :**
 - Authentification
 - Habilitation
 - Traçage
 - Audit
 - Stockage sécurisé des données
 - Transport sécurité des données
 - Sécurisation des données d'entrée
 - Sécurisation des données de sortie

Geoportail



Google Maps ---



Obscurité != Sécurité

- La sécurité par l'obscurité ne fonctionne pas
- Ne pas essayer de cacher des fonctions
- Protection des données != chiffrement
- Il existe toujours la possibilité de “contourner” des mesures telles que de l'obfuscation de code.

Codage sécurisé d'une application

Architecture de sécurité

► Fail securely

Authentification

Gestion des sessions applicatives

Contrôle d'accès

Validation des données

Éléments cryptographiques

Gestion des erreurs et de la journalisation

Protection de l'accès aux données

Sécurité du transport des données

Configuration de la sécurité

Code Malveillant et sécurité interne au logiciel

Prévoir l'imprévisible

- Ne pas donner d'informations à l'utilisateur du crash
- Tracer tous les crash dans un fichier “sécurisé”
- Centraliser les traçages des crash.
- Analyser tous les crash applicatifs.
- Utiliser un framework de traçage “standard”
 - Log4J
 - Microsoft Events Systems
 - Syslog

Codage sécurisé d'une application

Architecture de sécurité

Fail securely

► Authentification

Gestion des sessions applicatives

Contrôle d'accès

Validation des données

Éléments cryptographiques

Gestion des erreurs et de la journalisation

Protection de l'accès aux données

Sécurité du transport des données

Configuration de la sécurité

Code Malveillant et sécurité interne au logiciel

Implémenter une bonne stratégie de mot de passe

- **Longueur :**
 - Catégoriser les applications :
 - Importante : au moins 6 caractères
 - Critique : au moins 8 caractères et peut être plusieurs facteurs
 - Hautement Critique : au moins 14 caractères ET plusieurs facteurs
- **Complexité :**
 - Ajouter impérativement de la complexité aux catégories précédentes :
 - Au moins : 1 minuscule, 1 majuscule, 1 numérique, 1 caractère spécial
 - Ne pas autoriser de mots de dictionnaires
 - Ne pas autoriser des caractères continus (AA, 11,)

Implémenter une bonne stratégie de mot de passe (suite)

- Laisser l'utilisateur le choisir
- Forcer l'utilisation à le changer régulièrement et ajouter la non ré-utilisation des anciens
- Ne pas autoriser les rappels de mots de passes trop fréquents
- Ne pas autoriser le changement de mot de passe sans l'approbation de l'utilisateur :
 - Cela nécessite donc le mot de passe actuel au minimum, voir plus pour les accès hautement critiques
- Ajouter une stratégie des accès dormants
- **Ne pas stocker le mot de passe en CLAIR !**
- **Ajouter un sel à la fonction de “hashage” utilisée**

L'authentification multi-facteurs

- **Les mots de passes :**
 - Sont mauvais
 - Sont facilement trouvables
- **Mais néanmoins nécessaires et acceptables par l'utilisateur....**
- **La combinaison de plusieurs facteurs c'est :**
 - Quelque chose que vous avez (un token, un mobile, ...)
 - Quelque chose que vous connaissez (un détail sur vous, un mot de passe, ...)
 - Parfois un paramètre biométrique !

À utiliser pour tout environnement hautement critique

Implémenter une bonne stratégie globale

- Demander une seconde authentification pour les transactions critiques (avec du multi facteur si possible)
- Forcer l'authentification sur un canal chiffré (utilisation, de IPSec, TLS, ...)
- Limiter les changements, modifications, rappels des logins et des mots de passe
 - Par heure
 - Par jour
 - Par canal d'accès
 - Par mois
- Mettre une surveillance sur les changements concernant les éléments des administrateurs

Un bon exemple de stratégie



Dear Sebastien goria,

The password for your Apple ID has been successfully reset.

If you believe you have received this email in error, or that an unauthorized person has accessed your account, please go to iforgot.apple.com to reset your password immediately. Then review and update your security settings at appleid.apple.com >

Questions? There are lots of answers on our [Apple ID support page](#) >

Thanks,
Apple Customer Support

Comment s'y prendre ?

- Authentifier tous les accès :
 - Présentation d'un token de transaction ou d'un identifiant valide et mot de passe
 - Ne pas autoriser plus d'un mécanisme d'authentification
- Authentifier sur le SERVEUR
- Renvoyer un message "identifiant ou mot de passe invalide" après une mauvaise authentification
- Tracer toute mauvaise et bonne authentification
- Après toute authentification réussie, donner à l'utilisateur une information sur le statut de la précédente

Codage sécurisé d'une application

Architecture de sécurité

Fail securely

Authentification

► Gestion des sessions applicatives

Contrôle d'accès

Validation des données

Éléments cryptographiques

Gestion des erreurs et de la journalisation

Protection de l'accès aux données

Sécurité du transport des données

Configuration de la sécurité

Code Malveillant et sécurité interne au logiciel

Gestion des sessions

- L'élément permettant de déterminer une session doit être :
 - Aléatoire
 - Cryptographiquement « sur »
 - Ne pas être « créer » de toute pièce ; utiliser le générateur de sessions du framework ; il a été testé et est « sûr »
- Utiliser si possible un autre nom pour l'élément de session que celui générer par le framework (exemple sur le web, si possible pas JSESSIONID....)
- Forcer l'utilisation d'un canal chiffré pour tout transport de cet élément (IPSEC, TSL,)
- Dans le cas d'une application Web utilisant les cookies :
 - Le cookie doit être SECURE
 - Le cookie doit être HTTPOnly
 - Il doit disposer d'un chemin et d'un domaine
 - Il ne doit pas disposer d'une expiration; il se doit d'être de session

Gestion des sessions en détails

- Mettre en place des expirations de session ⇔ Catégoriser les applications.
Exemple :
 - Application simple : 1 heure
 - Application critique (transactionnelle) : 20 minutes max
 - Application hautement critique (financière ou impact fort) : 5 minutes max
- Renouveler l'élément de session après chaque changement de privilège.
- Ne pas autoriser de connexion simultanées avec le même identifiant
- Ajouter des détection des attaques :
 - IP de la session,
 - Clef de transaction,
 - Autre élément aléatoire
 -
- Dans le cas d'utilisation de navigateurs Web:
 - Utiliser Javascript pour automatiquement fermer les application critiques
 - Désactiver la possibilité de partager les ID de sessions entre les fenêtres/onglets

Codage sécurisé d'une application

Architecture de sécurité

Fail securely

Authentification

Gestion des sessions applicatives

► Contrôle d'accès

Validation des données

Éléments cryptographiques

Gestion des erreurs et de la journalisation

Protection de l'accès aux données

Sécurité du transport des données

Configuration de la sécurité

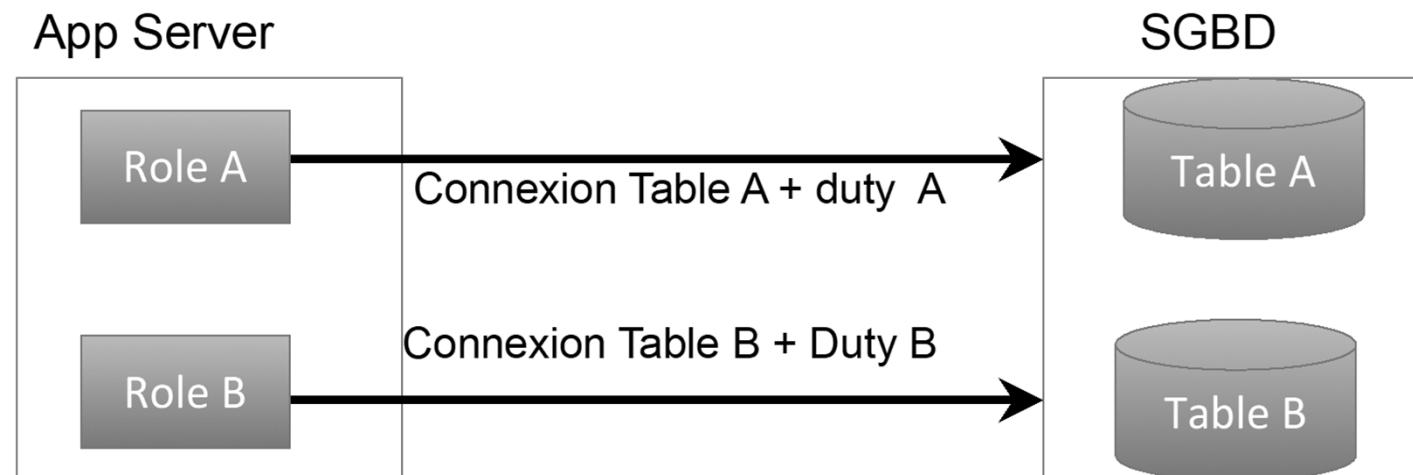
Code Malveillant et sécurité interne au logiciel

Contrôle d'accès == Habilitations



Prérequis

- **Sans contrôle d'accès, votre application est sous contrôle de l'utilisateur !**
- **Le client est par défaut incontrôlable.**
- **Deux niveaux d'habilitations sont nécessaires :**
 - Dans l'application
 - Dans l'infrastructure



Principes de contrôle d'accès

- **Garder la règle implicite suivantes :**
 - Aucune habilitation par défaut !
- **Centraliser l'ensemble du code des habilitations sur le serveur**
- **Si il est nécessaire que le client garde un état des habilitations, ajouter du chiffrement et de la vérification d'intégrité sur le server pour empêcher la modification d'état**
- **Limiter le nombre de transaction (en fonction de la catégorie des transactions) par utilisateur dans un intervalle de temps donné.**
- **Renforcer :**
 - La protection des transactions en fonction d'un compte en particulier
 - La protection des fonctions métiers en fonction des comptes
 - La protection d'accès aux système de fichiers

Principes de contrôle d'accès (suite)

- L'application doit arrêter tout traitement dès qu'il y a un problème de contrôle d'accès
- Différencier le code du contrôle d'accès administrateur du contrôle d'accès utilisateur
- Ajouter des mécanismes de gestion des comptes « dormants » :
 - Perte de privilèges,
 - « désactivation » du compte,
 - alertes,
 -

Codage sécurisé d'une application

Architecture de sécurité

Fail securely

Authentification

Gestion des sessions applicatives

Contrôle d'accès

► Validation des données

Éléments cryptographiques

Gestion des erreurs et de la journalisation

Protection de l'accès aux données

Sécurité du transport des données

Configuration de la sécurité

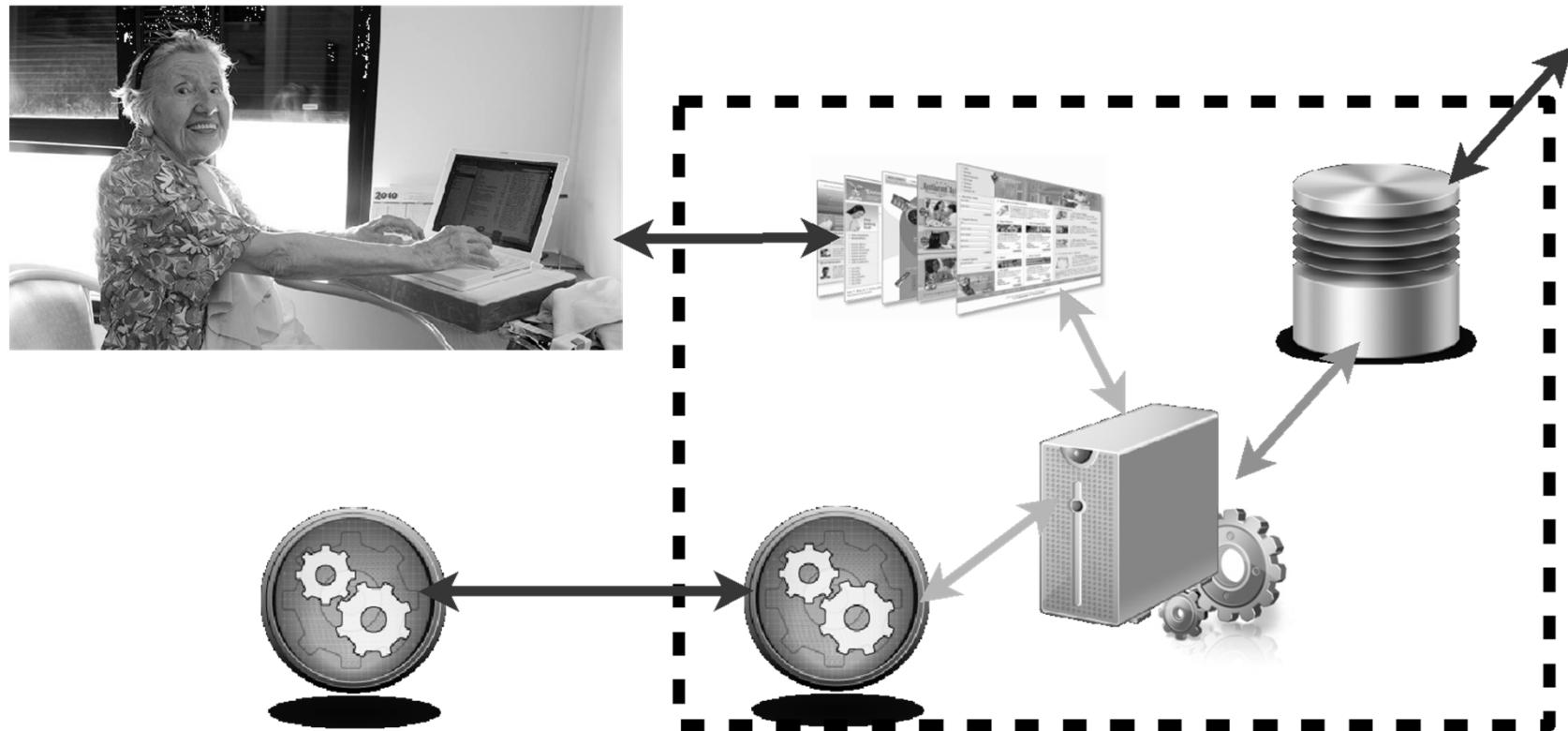
Code Malveillant et sécurité interne au logiciel

Validation d'entrée

- **S'assurer que toute validation de donnée est effectuée sur un code « sous contrôle »; par défaut sur LE SERVEUR receveur.**
- **Classer les données :**
 - Données non validées
 - Données valides
- **Mettre en place un chemin de validation, centralisé.**
- **Canonicaliser les données d'entrée !**
- **Rejecter les mauvaises données !**
- **Valider les données :**
 - En fonction du type ; utiliser les types du langage
 - En fonction d'un intervalle de données
 - En fonction de la taille
 - En fonction de la valeur
 - En fonction d'une liste blanche.

Frontière de validation

- Considérer la validation de donnée sur chacun des points d'entrée de l'application



Encodage des données de sortie

- C'est un grand principe de défense en profondeur
 - Toutes les données envoyées au client doivent l'être dans :
 - Le format
 - L'encodage
-Nécessaires à la transformation client**

Codage sécurisé d'une application

Architecture de sécurité

Fail securely

Authentification

Gestion des sessions applicatives

Contrôle d'accès

Validation des données

► Éléments cryptographiques

Gestion des erreurs et de la journalisation

Protection de l'accès aux données

Sécurité du transport des données

Configuration de la sécurité

Code Malveillant et sécurité interne au logiciel

Chiffrement

- Il ne faut pas utiliser le codage à des fins de protections
- L'algorithme employé doit avoir été revu.
- La clef la plus longue possible doit être employé
- Il est nécessaire de stocker les clefs de chiffrement sur des serveurs « surs » et si possible sur un moyen de type HSM (Hardware Security Module)

Codage sécurisé d'une application

Architecture de sécurité

Fail securely

Authentification

Gestion des sessions applicatives

Contrôle d'accès

Validation des données

Éléments cryptographiques

► Gestion des erreurs et de la journalisation

Protection de l'accès aux données

Sécurité du transport des données

Configuration de la sécurité

Code Malveillant et sécurité interne au logiciel

Gestion des erreurs

- **Votre application plantera !**
- **Il est nécessaire**
 - De gérer tout exception sans exception !
 - De nettoyer le code des exceptions de toute donnée sensible
 - De ne pas donner à l'utilisateur une information sur l'erreur
- **Les logs sont sensibles, vous DEVEZ les protéger**
- **Logger :**
 - Les erreurs de validation de données d'entrée
 - Les requêtes d'authentification; en particulier les erreurs
 - Les problèmes de contrôle d'accès
 - Les exceptions systèmes
 - Les fonctions administratives
 - Les problèmes liés au chiffrement
 - Les éléments relatifs aux sessions non valides ou expirées.

Comment logger

- Utiliser un framework standardiser de journalisation .
- Découper les fichiers de logs en fonction des catégories de logs. Exemple :
 - Logs d'accès ou de traitement
 - Logs d'erreur
 - Logs de debug
 - Logs d'audit
 - Logs réglementaires

Codage sécurisé d'une application

Architecture de sécurité

Fail securely

Authentification

Gestion des sessions applicatives

Contrôle d'accès

Validation des données

Éléments cryptographiques

Gestion des erreurs et de la journalisation

► Protection de l'accès aux données

Sécurité du transport des données

Configuration de la sécurité

Code Malveillant et sécurité interne au logiciel

Protection des données

- **La protection des données sensibles est critique :**
 - Il ne faut pas les stocker en clair
 - Il ne faut pas les transmettre en clair
 - Il ne faut pas effectuer de « cache » de la donnée sur les clients
- **Il est nécessaire de classifier les données avant toute protection :**
 - Public : la donnée peut être exposée en dehors de l'application
 - Privée : la donnée ne peut « sortir » de l'application
 - Confidentielle : la donnée ne peut sortir de l'application et doit être « chiffrée »
 - Hautement confidentielle : la donnée ne peut sortir de l'application, doit être chiffrée, et il faut vérifier son intégrité.
- **Considérer la mise en place des mécanismes de lecture seul pour les systèmes de fichiers et/ou base de données lorsque c'est nécessaire.**

Codage sécurisé d'une application

Architecture de sécurité

Fail securely

Authentification

Gestion des sessions applicatives

Contrôle d'accès

Validation des données

Éléments cryptographiques

Gestion des erreurs et de la journalisation

Protection de l'accès aux données

► Sécurité du transport des données

Configuration de la sécurité

Code Malveillant et sécurité interne au logiciel

Sécurisation de la communication

- **Ne pas exposer de données critiques sur un canal non chiffré**
 - Utilisation d'AES au minimum
 - Utilisation de clefs de 128 bits
- **Dans le cas d'utilisation de certificats :**
 - Valider les certificats avant utilisation
 - Dates
 - Signatures
 - Contenu
 - Valider auprès de l'émetteur si possible
- **Ne pas protéger uniquement la page d'authentification, mais TOUTES les pages suivantes**

Codage sécurisé d'une application

Architecture de sécurité

Fail securely

Authentification

Gestion des sessions applicatives

Contrôle d'accès

Validation des données

Éléments cryptographiques

Gestion des erreurs et de la journalisation

Protection de l'accès aux données

Sécurité du transport des données

► Configuration de la sécurité

Code Malveillant et sécurité interne au logiciel

Sécuriser les interfaces d'administration

- Utiliser un système d'authentification à facteurs multiples
- Tracer tout accès, dans un fichier autre que ceux utilisateurs
- Renforcer l'audit des fonctions :
 - Changement de privilèges
 - Transaction pour tiers,
 -
- Se souvenir que le support client n'est pas administrateur.
- Rajouter des mécanismes d'authentification autres pour les accès critiques

Revue des configurations

- **Revoir tous les fichiers de configurations :**
 - Attention aux mots de passe par défauts => les changer
 - Supprimer les modules et fonctions non nécessaires
- **Utiliser les mécanismes de bac a sables quant ils sont disponibles**
- **Attentions aux codes signés qui s'exécutent la plupart du temps avec des privilèges supplémentaires (en java particulièrement)**

Codage sécurisé d'une application

Architecture de sécurité

Fail securely

Authentification

Gestion des sessions applicatives

Contrôle d'accès

Validation des données

Éléments cryptographiques

Gestion des erreurs et de la journalisation

Protection de l'accès aux données

Sécurité du transport des données

Configuration de la sécurité

► Code Malveillant et sécurité interne au logiciel

Code malveillant

- Mettre en place des contrôles d'intégrité forts sur les configurations et le code source
- Mettre en place des gestionnaires de version de code source

Sécurité interne du logiciel

- Mettre en place des mécanismes empêchant les accès concurrents aux variables partagées
- Mettre en place des interfaces simples et accessibles aux développeurs
- Mettre en place des mécanismes protégeant les données et les politiques d'accès des contrôles d'accès d'une modification non autorisée

Complexité du code

- Thomas McCabe a défini dans les années 70 la complexité d'un code :

$$\text{CC} = \text{nombre de décision} + 1$$

CC	Risque
0-10	Code et risque acceptable
11-15	Risque moyen
16-20	Risque fort, trop de décision dans un élément de code

Qualité de code

- L'utilisation de fonctions dépréciées doit être bannie.
- Les fonctions/méthodes de plus de 30/40 lignes doivent être limitées
- L'utilisation de commentaires doit être faite aux fonctions critiques

CERT Secure Coding

- **Documents publiés par le CERT/CC :**
 - <https://www.securecoding.cert.org/>
- **Présente les principales erreurs de codage dans un langage**
 - C :
<https://www.securecoding.cert.org/confluence/display/seccode/CERT+C+Secure+Coding+Standard>
 - C++ :
<https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=637>
 - Java :
<https://www.securecoding.cert.org/confluence/display/java/The+CERT+Oracle+Secure+Coding+Standard+for+Java>
 - Perl :
<https://www.securecoding.cert.org/confluence/display/perl/CERT+Perl+Secure+Coding+Standard>
- **Présente un code « non compliant » et un code « compliant »**
- **Chacune des pratiques est « classée »**
 - Impact : faible, médium, fort
 - Probabilité : improbable, moyenne, forte
 - Remédiation : simple, medium, complexe

Notes

Chapitre 3

Structuration du code, déclarations, initialisation

Structuration

```
/*
Mauvais
*/
system.out.println(final_time, (final_time -
    initial_time) * 4);

final_price = basic_price * tax_rate +
    initial_price;

pos = pos0 + speed * time + (acceleration * time
    * time) / 2;
```

Structuration (suite)

```
/*
Mieux
*/
system.out.println(final_time,
                    (final_time - initial_time) * 4);

final_price = basic_price * tax_rate
              + initial_price;

pos = pos0 + speed * time
      + (acceleration * time * time) / 2;
```

Structuration du code

- **Respecter un ordre dans les membres d'une classe pour la déclaration**
 1. Variables de classes (static)
 2. Variables instanciables
 3. Constructeurs
 4. Methode finalize ou destructeur
 5. Méthodes de statiques de la classe
 6. Set et Get
 7. Autres méthodes
- **Mettre en place une indentation standard (4 espaces par exemple)**
- **Eviter les lignes de plus de 80 caractères**
 1. Les couper après une virgule,
 2. Les couper avant un opérateur,
 3. Préférer couper entre deux expressions de haut niveau
 4. Aligner le texte sur la nouvelle ligne

Déclaration de variables

```
int src[], c;
```



Mauvais

Meilleur



```
int[] src; /* Tableau */  
int c;      /* Valeur */
```

Déclaration des variables (suite)

```
public class Example<T> {  
    private T a, b, c[], d;  
  
    public Example(T in){  
        a = in;  
        b = in;  
        c = (T[]) new Object[10];  
        d = in;  
    }  
}
```

Mauvais

Meilleur

```
public class Example<T> {  
    private T a;  
    private T b;  
    private T[] c;  
    private T d;  
  
    public Example(T in){  
        a = in;  
        b = in;  
        c = (T[]) new Object[10];  
        d = in;  
    }  
}
```

Constantes

```
double area(double radius) {  
    return 4.0 * 3.14 * radius * radius; }
```

```
double volume(double radius) {  
    return 4.0/3.0 * 3.14 * radius * radius * radius; }
```

```
double greatCircleCircumference(double radius) {  
    return 2 * 3.14 * radius; }
```

Mauvais

Bon

```
double area(double radius) {  
    return 4.0 * Math.PI * radius * radius; }
```

```
double volume(double radius) {  
    return 4.0/3.0 * Math.PI * radius * radius * radius; }
```

```
double greatCircleCircumference(double radius) {  
    return 2 * Math.PI * radius; }
```

Constantes (suite)

```
private final int BUFSIZE = 512;  
// ...  
nblocks = 1 + ((nbytes - 1) >> 9); /* BUFSIZE = 512 = 2^9 */
```

Mauvais

Bon



```
private final int BUFSIZE = 512;  
// ...  
nblocks = 1 + (nbytes - 1) / BUFSIZE;
```

Initialisation

```
public class Visual {  
    public static void main(String[] args) {  
        System.out.println(11111 + 1111);  
    }  
}
```



Mauvais

Bon



```
public class Visual {  
    public static void main(String[] args) {  
        System.out.println(11111 + 1111L);  
    }  
}
```

Initialisation (suite)

```
class IPaddress {  
String ipAddress = new String("172.16.254.1");  
public static void main(String[] args) {  
// ... }  
}
```

Mauvais

Bon

```
class IPaddress {  
public static void main(String[] args) throws IOException {  
char[] ipAddress = new char[100];  
BufferedReader br = new BufferedReader(new  
InputStreamReader(  
new FileInputStream("serveripaddress.txt")));  
  
// Reads the server IP address into the char array,  
// returns the number of bytes read  
int n = br.read(ipAddress);  
//validate the data from the buffer
```

Initialisation (suite)

```
array[0] = 2719;  
array[1] = 4435;  
array[2] = 0042;
```



```
array[0] = 2719;  
array[1] = 4435;  
array[2] = 34;
```

Surcharge

```
class OverloadedVarargs {  
    private static void doSomething(boolean... bool) {  
        System.out.print("Number of arguments: " + bool.length + ", Contents: ");  
  
        for (boolean b : bool)  
            System.out.print("[ " + b + " ]");  
    }  
    private static void doSomething(boolean bool1, boolean bool2) {  
        System.out.println("Overloaded method invoked");  
    }  
    public static void main(String[] args) {  
        doSomething(true, false);  
    }  
}
```

Surcharge (suite)

```
class NotOverloadedVarargs {  
    private static void doSomething1(boolean... bool) {  
        System.out.print("Number of arguments: " + bool.length + ", Contents: ");  
  
        for (boolean b : bool)  
            System.out.print("[ " + b + " ]");  
    }  
    private static void doSomething2(boolean bool1, boolean bool2) {  
        System.out.println("Overloaded method invoked");  
    }  
    public static void main(String[] args) {  
        doSomething1(true, false);  
    }  
}
```

Sérialisation

- Toute classe est sérialisable, si il n'est pas nécessaire de la laisser sérialisable, alors il faut intégrer ce code

```
private final void writeObject(ObjectOutputStream out)
throws java.io.IOException {
    throw new java.io.IOException("Object cannot be serialized");
}
private final void readObject(ObjectInputStream in)
throws java.io.IOException {
    throw new java.io.IOException("Class cannot be deserialized");
}
```

Déclaration des variables

- **Une variable par ligne**
- **Une instruction par ligne**
- **Formater les structures de contrôles comme suit :**
 - Ajouter des { et } entourant le code des structures de contrôle
 - Finaliser toute structure de contrôle par un commentaire // *fin controle*
- **Utiliser les constantes lorsque nécessaire**
- **Utiliser L pour Long**
- **Attention aux 0 en début de chaine/valeurs**
- **Ne pas surcharger les méthodes d'arguments variables**

Notes

Chapitre 4

Les Expressions

Valeurs de retour

```
File someFile = new File("someFileName.txt");
// diverses opérations sur someFile
someFile.delete();
```

Mauvais

Bon

```
File someFile = new File("someFileName.txt");
// diverses opérations sur someFile
if (!someFile.delete()) {
    // gestion des erreurs
}
```

Valeurs de retour (suite)

```
public class Ignore {  
    public static void main(String[] args) {  
        String original = "insecure";  
        original.replace( 'i', '9' );  
        System.out.println(original);  
    }  
}
```



Mauvais

Bon



```
public class DoNotIgnore {  
    public static void main(String[] args) {  
        String original = "insecure";  
        original = original.replace( 'i', '9' );  
        System.out.println(original);  
    }  
}
```

Comparaisons

```
public class ExampleComparison {  
    public static void main(String[] args) {  
        String one = new String("one");  
        String two = new String("one");  
        boolean result;  
  
        if (one == null) {  
            result = two == null;  
        }  
        else {  
            result = one == two;  
        }  
        System.out.println(result); // prints false  
    }  
}
```



Mauvais

Comparaisons (suite)

Bon



```
public class GoodComparison {  
    public static void main(String[] args) {  
        String one = new String("one");  
        String two = new String("one");  
        boolean result;  
        if (one == null) {  
            result = two == null;  
        } else {  
            result = one.equals(two);  
        }  
        System.out.println(result);  
    }  
}
```

Comparaisons (suite)

```
int[] arr1 = new int[20]; // init a 0  
int[] arr2 = new int[20]; // init a 0  
arr1.equals(arr2);
```



Mauvais

Comparer l'objet et le contenu de l'objet produit un résultat différent !

Bon



```
int[] arr1 = new int[20]; // initialized to 0  
int[] arr2 = new int[20]; // initialized to 0  
Arrays.equals(arr1, arr2); // true
```

Opérateurs de comparaison

```
public class WrapperKO {  
    public static void main(String[] args) {  
  
        Integer i1 = 100;  
        Integer i2 = 100;  
        Integer i3 = 1000;  
        Integer i4 = 1000;  
        System.out.println(i1 == i2); //true  
        System.out.println(i1 != i2); //false  
        System.out.println(i3 == i4); //false  
        System.out.println(i3 != i4); // true  
  
    }  
}
```



Mauvais

Opérateurs de comparaison (suite)

Bon



```
public class WrapperOK {  
    public static void main(String[] args) {  
        Integer i1 = 100;  
        Integer i2 = 100;  
        Integer i3 = 1000;  
        Integer i4 = 1000;  
        System.out.println(i1.equals(i2)); //true  
        System.out.println(!i1.equals(i2)); //false  
        System.out.println(i3.equals(i4)); // true  
        System.out.println(!i3.equals(i4)); // false  
    }  
}
```

Casts

```
public class ExprKO {  
    public static void main(String[] args) {  
        short s = 2;  
        s += 32766.125;  
        int x = 2147483642;  
        x += 1.0f;  
    }  
}
```



Mauvais

Bon



```
public class ExprOK {  
    public static void main(String[] args) {  
        double s = 2;  
        s += 32766.125; // s contient 32768.125  
  
        double x = 2147483642; // 0x7fffffff  
        x += 1.0; // x contient 2147483643.0 (0x7fffffff.b.0)  
    }  
}
```

Casts (suite)

```
class Test{
    public static void main(String[] args){
        int big = 1999999999;
        float one = 1.0f;
        System.out.println(big * one);
    }
}
```



Mauvais

Bon



```
class Test{
    public static void main(String[] args){
        int big = 1999999999;
        double one = 1.0d; // double à la place d'un flottant
        System.out.println(big*one);
    }
}
```

Parenthèses

```
public static final int MASK = 1337;  
public static final int OFFSET = -1337;
```

```
public static int computeCode(int x) {  
    return x & MASK + OFFSET;  
}
```



```
x & (MASK + OFFSET)
```

```
x & (1337 - 1337)
```

Mieux

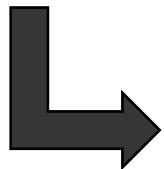
```
public static final int MASK = 1337;  
public static final int OFFSET = -1337;
```

```
public static int computeCode(int x) {  
    return (x & MASK) + OFFSET;  
}
```

Accès aux tableaux

```
import java.util.HashSet;

public class ShortSetKO {
    public static void main(String[] args) {
        HashSet<Short> s = new HashSet<Short>();
        for(short i=0; i<100;i++) {
            s.add(i);
            s.remove(i - 1);
        }
        System.out.println(s.size());
    }
}
```

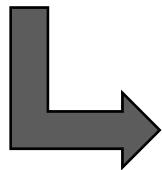


Affichera 100

Accès aux tableaux (suite)

```
import java.util.HashSet;

public class ShortSetOK {
    public static void main(String[] args) {
        HashSet<Short> s = new HashSet<Short>();
        for(short i=0; i<100;i++) {
            s.add(i);
            s.remove((short)(i-1)); //cast en short
        }
        System.out.println(s.size());
    }
}
```



Affichera 1

Expressions conditionnelles

```
public class ExprKO {  
    public static void main(String[] args) {  
        char alpha = 'A';  
        System.out.print(true ? alpha : 0);  
    }  
}
```



Mauvais

Bon



```
public class ExprOK {  
    public static void main(String[] args) {  
        char alpha = 'A';  
        System.out.print(true ? alpha : ((char) 0));  
    }  
}
```

Expressions conditionnelles (suite)

```
public class ExprKO {  
    public static void main(String[] args) {  
        char alpha = 'A';  
        System.out.print(true ? alpha : 12345);  
    }  
}
```



Mauvais

Bon



```
public class Expr {  
    public static void main(String[] args) {  
        char alpha = 'A';  
        System.out.print(true ? ((int) alpha) : 12345);  
    }  
}
```

Rappels sur les opérandes

Operand 2	Operand 3	Resultant type
type T	type T	type T
boolean	Boolean	boolean
Boolean	boolean	boolean
null	reference	reference
reference	null	reference
byte Or Byte	short Or Short	short
short Or Short	byte Or Byte	short
byte, short, char	constant int	byte, short, char if value of int is representable
constant int	byte, short, char	byte, short, char if value of int is representable
Byte	constant int	byte if int is representable as byte
constant int	Byte	byte if int is representable as byte
Short	constant int	short if int is representable as short
constant int	Short	short if int is representable as short
Character	constant int	char if int is representable as char
constant int	Character	char if int is representable as char
other numeric	other numeric	promoted type of the 2nd and 3rd operands
T1 = boxing conversion (S1)	T2 = boxing conversion(S2)	apply capture conversion to lub(T1,T2)

Gestion des expressions

- **Ne pas ignorer les valeurs de retour**
- **Attention aux comparaisons :**
 - D'objets
 - De tableaux
 - Aux opérateurs
- **Attention aux « cast » implicites**
- **Attention aux « cast » des entiers**
- **Penser aux parenthèses !**
- **Attention lors des boucles pour les accès aux tableaux**
- **Attention aux types pour les expressions conditionnelles**

Notes

Chapitre 5

La portée



Portée des variables

```
public class Scope {  
    public static void main(String[] args) {  
        int i = 0;  
        for(i = 0; i < 10; i++) {  
            // operations  
        }  
    }  
}
```



Mauvais

Meilleur



```
public class Scope {  
    public static void main(String[] args) {  
        for(int i = 0; i < 10; i++) {  
            // operations  
        }  
    }  
}
```

Accessibilité

```
class BadScope {  
    protected void doLogic() {  
        System.out.println("Super invoked");  
    }  
}  
  
public class Sub extends BadScope {  
    public void doLogic() {  
        System.out.println("Sub invoked");  
        // Do operations  
    }  
}
```



Mauvais

Accessibilité (suite)

Meilleur



```
class BadScope {  
    protected final void doLogic() {  
        System.out.println("Super invoked");  
        // Do operations  
    }  
}
```

Réutilisation des termes

```
class Vector {  
    private int val = 1;  
    private void doLogic() {  
        int val;  
        //...  
    }  
}
```



Mauvais

Bon



```
class Vector {  
    private int val = 1;  
    private void doLogic() {  
        int newValue;  
        //...  
    }  
}
```

Expositions

```
class Coordinates {  
    private int x;  
    private int y;  
  
    public class Point {  
        public void getPoint() {  
            System.out.println("(" + x + "," + y + ")");  
        }  
    }  
}  
  
class AnotherClass {  
    public static void main(String[] args) {  
        Coordinates c = new Coordinates();  
        Coordinates.Point p = c.new Point();  
        p.getPoint();  
    }  
}
```



Mauvais

Expositions (suite)

Bon



```
class Coordinates {  
    private int x;  
    private int y;  
  
    private class Point {  
        private void getPoint() {  
            System.out.println("(" + x + "," + y + ")");  
        }  
    }  
}  
  
class AnotherClass {  
    public static void main(String[] args) {  
        Coordinates c = new Coordinates();  
        Coordinates.Point p = c.new Point(); // fails à la compilation  
        p.getPoint();  
    }  
}
```

Suppression des Warnings

```
@SuppressWarnings("unchecked") class Legacy {  
    Set s = new HashSet();  
    public final void doLogic(int a,char c) {  
        s.add(a);  
        s.add(c);  
    }  
}
```



Mauvais

Bon



```
class Legacy {  
    @SuppressWarnings("unchecked") Set s = new HashSet();  
    public final void doLogic(int a,char c) {  
        s.add(a); // Produit un « unchecked warning »  
        s.add(c); // Produit un « unchecked warning »  
    }  
}
```

Exhauste la RAM.

```
public class ReadNames {  
    // Accepts unknown number of records  
    Vector<Long> names = new Vector<Long>();  
    long newID = 0L;  
    int count = 67108865;  
    int i = 0;  
    InputStreamReader input = new InputStreamReader(System.in);  
    ...  
    ....
```



Mauvais

Meilleur



```
// ...  
int count = 10000000;  
// ...
```

La portée

- Toujours déclarer des classes final, car sinon il est possible de les étendre.
- Limiter la portée des variables
- Eviter la réutilisation :
 - Des termes
 - Des noms de classes
- Limiter les expositions :
 - Des membres sensibles
 - Des méthodes métier internes.

Chapitre 6

Entiers, réels...

Integer Overflow

```
public int multAccum(int oldAcc, int newVal, int scale) {  
    return oldAcc + (newVal * scale);  
}
```



Mauvais

Meilleur

```
public int multAccum(int oldAcc, int newVal, int scale) throws ArithmeticException {  
    preMultiply(newVal, Scale);  
    final int temp = newVal * scale;  
    preAdd(oldAcc, temp);  
    return oldAcc + temp;  
}
```

Débordements et opérations

- **byte : -128 à 127, inclusive**
- **short : -32,768 à 32,767, inclusive**
- **int : -2,147,483,648 à 2,147,483,647, inclusive**
- **long : -9,223,372,036,854,775,808 à 9,223,372,036,854,775,807, inclusive**
- **char : \u0000 à \uffff inclusive,(0 à 65,535)**

Operator	Overflow	Operator	Overflow	Operator	Overflow	Operator	Overflow
+	yes	-=	yes	<<	no	<	no
-	yes	*=	yes	>>	no	>	no
*	yes	/=	yes	&	no	>=	no
/	yes	%=	no	\	no	<=	no
%	no	<<=	no	^	no	==	no
++	yes	>>=	no	~	no	!=	no
--	yes	&=	no	!	no	&&	no
=	no	=	no	un +	no		no
+=	yes	^=	no	un -	yes	? :	no

Intervalle et cast

```
class CastAway {  
    public static void main(String[] args) {  
        int i = 128;  
        byte b = (byte) i;  
    }  
}
```



Mauvais

Bon



```
class CastAway {  
    public static void main(String[] args) {  
        int i = 128;  
        if ((i < Byte.MIN_VALUE) || (i > Byte.MAX_VALUE)) {  
            throw new ArithmeticException("Value is out of range");  
        }  
        byte b = (byte) i;  
    }  
}
```

Reste

$5 \% 3$ reste 2

$5 \% (-3)$ reste 2

$(-5) \% 3$ **reste -2**

$(-5) \% (-3)$ **reste -2**

Retour de valeurs

```
char c;  
while ((c = (char) in.read()) != -1) {  
    // ...  
}
```



Mauvais

Bon



```
int c;  
while ((c = in.read()) != -1) {  
    ch = (char) c;  
}
```

Décalages

```
static int countOneBits(long value) {  
    int bits = 0;  
    while (value != 0) {  
        bits += value & 1L;  
        value >>= 1;  
    }  
    return bits;  
}
```



Mauvais

Bon



```
static int countOneBits( long value ) {  
    int bits = 0;  
    while (value != 0) {  
        bits += value & 1L;  
        value >>>= 1;  
    }  
    return bits;  
}
```

Précision et flottants

```
double dollar = 1.0;  
double dime = 0.1;  
int number = 7;  
System.out.println ("A dollar less " + number + " dimes is $" +  
    (dollar - number * dime) );
```



Mauvais

Bon



```
long dollar = 100;  
long dime = 10;  
int number = 7;  
System.out.println ("A dollar less " + number + " dimes is " +  
    (dollar - number * dime) + " cents" );
```

Attention aux expressions arithmétiques

```
double x, y, z;  
// ...  
x = (x * y) * z; // non équivalent à x *= y * z;  
z = (x - y) + y; // non équivalent à z = x;  
z = x + x * y; // non équivalent à z = x * (1.0 + y);  
y = x / 5.0; // non équivalent à y = x * 0.2;  
y = z + 0.5 + 0.5; // non équivalent à y = z + 1.0;
```

Attention aux expressions arithmétiques (suite)

```
short a = 533;  
int b = 6789;  
long c = 4664382371590123456L;
```

```
float d = a / 7;  
double e = b / 30;  
double f = c * 2;
```



Mauvais

Bon



```
short a = 533;  
int b = 6789;  
long c = 4664382371590123456L;  
float d = a;  
double e = b;  
double f = c;  
  
d /= 7;  
e /= 30;  
f *= 2;
```

Comparaison avec NaN

```
public class NaNComparison {  
    public static void main(String[] args) {  
        double x = 0.0;  
        double result = Math.cos(1/x);  
        if (result == Double.NaN) {  
            System.out.println("Egal ");  
        }  
    }  
}
```



Mauvais

Bon



```
public class NaNComparison {  
    public static void main(String[] args) {  
        double x = 0.0;  
        double result = Math.cos(1/x)  
        if (Double.isNaN(result)) {  
            System.out.println("Egal ");  
        }  
    }  
}
```

Flottant et boucle

```
for (float x = 0.1f; x <= 1.0f; x += 0.1f) {  
    // ...  
}
```



Mauvais

Bon



```
for (int count = 1; count <= 10; count += 1) {  
    float x = count/10.0f;  
    // ...  
}
```

BigDecimal

```
System.out.println(new BigDecimal(0.1));
```



Mauvais

Bon



```
System.out.println(new BigDecimal("0.1"));
```

Chaînes

```
int i = 1;  
String s = Double.valueOf (i / 1000.0).toString();  
if(s.equals("0.001")) {  
//.....  
}  
int i = 1;  
String s = Double.valueOf (i / 10000.0).toString();  
s = s.replaceFirst("[.0]*$", "");  
// ...
```



Mauvais

```
int i = 1;  
BigDecimal d = new BigDecimal(Double.valueOf (i / 1000.0).toString()).stripTrailingZeros();  
if(d.compareTo(new BigDecimal("0.001")) == 0) { // OK  
//.....  
}
```

Entiers, Réels

- Attentions aux overflow en fonction des opérations
- Ne pas utiliser des ‘char’ pour des valeurs de retour
- Le Reste peut être négatif
- Attention aux intervalles lors de cast
- Attention aux décalages !
- Convertir les entiers en flottants pour les opérations flottantes
- Utiliser les méthodes de conversion !

Notes

Chapitre 7

Les Objets

Accès aux membres

```
public class Widget {  
    public int total; // Nombre d'éléments  
    void add() {  
        if (total < Integer.MAX_VALUE) {  
            total++;  
            // ...  
        } else {  
            throw new ArithmeticException("Overflow");  
        }  
    }  
    void remove() {  
        if (total > Integer.MIN_VALUE) {  
            total--;  
            // ...  
        } else {  
            throw new ArithmeticException("Overflow");  
        }  
    }  
}
```



Mauvais

Accès aux membres (suite)

Bon



```
public class Widget {  
    private int total;  
  
    void add() {  
        // ...  
    }  
  
    void remove() {  
        // ...  
    }  
  
    public int getTotal () {  
        return total;  
    }  
}
```

Variables statiques

```
class DataSerializer implements Serializable {  
    public static long serialVersionUID = 1973473122623778747L;  
    // ...  
}
```



Mauvais

```
class DataSerializer implements Serializable {  
    private static final long serialVersionUID = 1973473122623778747L;  
}
```



Bon

Comparaisons

```
if(obj.getClass().getName().equals("Foo"))
    // nom de la classe Foo
}else{
    // autre type de classe
}
```



Mauvais

Bon



```
if(obj.getClass() == this.getClassLoader().loadClass("Foo")){
    // meme type de classe que Foo
}else{
    // autre classe
}
```

Ramasse-Miettes

```
ByteBuffer buffer = ByteBuffer.allocateDirect(8192);
```



Mauvais

Bon



```
ByteBuffer buffer = ByteBuffer.allocate(8192);
```

```
int[] buffer = new int[100];  
doSomething(buffer);  
buffer = null; //inutile
```



Mauvais

Bon



```
{  
    // limiter le scope du buffer  
    int[] buffer = new int[100];  
    doSomething(buffer);  
}
```

Objets

- **Eviter au maximum les champs publics :**
 - Dans les champs/variables
 - Dans les méthodes
 - Dans les objets
- **Mettre en place des « Setter » et des « Getter » au maximum pour les variables**
- **Faire attention sur les comparaisons d'objets.**

Notes

Chapitre 8

Entrées / Sorties

Encodage de sortie

```
FileInputStream fis = new FileInputStream("SomeFile");
DataInputStream dis = new DataInputStream(fis);
int bytesRead = 0;
byte[] data = new byte[1024];
bytesRead = dis.readFully(data);
if (bytesRead > 0) {
    String result = new String(data);
}
```



Mauvais

Encodage de sortie (suite)

Bon



```
String encoding = "UTF-16"
FileInputStream fis = new FileInputStream("SomeFile");
DataInputStream dis = new DataInputStream(fis);
int bytesRead = 0;
byte[] data = new byte[1024];
bytesRead = dis.readFully(data);
if (bytesRead > 0) {
    String result = new String(data, encoding);
}
```

Canonicalisation

```
public static void main(String[] args) {  
    File f = new File("/tmp/" + args[0]);  
    String absPath = f.getAbsolutePath();  
  
    if(!validate(absPath)) { // Validation  
        throw new IllegalArgumentException();  
    }  
}
```



Mauvais

Bon



```
public static void main(String[] args) throws IOException {  
    File f = new File("/tmp/" + args[0]);  
    String canonicalPath = f.getCanonicalPath();  
  
    if(!validate(canonicalPath)) { // Validation  
        throw new IllegalArgumentException();  
    }  
}
```

Canonicalisation (suite)

```
FileOutputStream fis = new FileOutputStream(new File("/img/" + args[0]));
```



Mauvais

Bon



```
File f = new File("/img/" + args[0]);
String canonicalPath = f.getCanonicalPath();

if(canonicalPath.equals("/img/java/file1.txt")) { // Validation
    // .....
}
if(!canonicalPath.equals("/img/java/file2.txt")) { // Validation
    // .....
}

FileOutputStream fis = new FileOutputStream(f);
```

Fermer les connecteurs

```
Statement stmt = null;  
ResultSet rs = null;  
Connection conn = getConnection();  
try {  
    stmt = conn.createStatement();  
    rs = stmt.executeQuery(sqlQuery);  
    processResults(rs);  
} catch(SQLException e) {  
    /.....  
} finally {  
    if(rs != null) {  
        rs.close();  
    }  
    if(stmt != null) {  
        stmt.close();  
    }  
}
```



Mauvais

Fermer les connecteurs (suite)

Bon



```
Statement stmt = null;  
ResultSet rs = null;  
Connection conn = getConnection();  
try {  
    stmt = conn.createStatement();  
    rs = stmt.executeQuery(sqlQuery);  
    processResults(rs);  
} catch(SQLException e) {  
    //.....  
} finally { try {  
    if(rs != null) {  
        rs.close();  
    } } finally {  
    try {  
        if(stmt != null) {  
            stmt.close();  
        } } finally {  
        conn.close();  
    } } }
```

Fermer les connecteurs (suite)

```
public int processFile(String fileName) throws IOException, FileNotFoundException {  
    FileInputStream stream = new FileInputStream(fileName);  
    BufferedReader bufRead = new BufferedReader(new InputStreamReader(stream));  
    String line;  
    while((line = bufRead.readLine()) != null) {  
        sendLine(line);  
    }  
    return 1;  
}
```



Mauvais

Fermer les connecteurs (suite)

Bon



```
FileInputStream stream = null;
BufferedReader bufRead = null;
String line;
try {
    stream = new FileInputStream(fileName);
    bufRead = new BufferedReader(new InputStreamReader(stream));

    while((line = bufRead.readLine()) != null) {
        sendLine(line);
    }
} catch (IOException e) {
    // forward to handler
} finally {
    if(stream != null) {
        stream.close();
    }
}
```

Permission d'ouverture

```
Writer out = new FileWriter("file");
```

Mauvais

```
Path file = new File("file").toPath();
```

```
// Throw exception rather than overwrite existing file  
Set<OpenOption> options = new HashSet<OpenOption>();  
options.add(StandardOpenOption.CREATE_NEW);  
options.add(StandardOpenOption.APPEND);
```

```
// File permissions should be such that only user may read/write file  
Set<PosixFilePermission> perms =  
    PosixFilePermissions.fromString("rw-----");  
FileAttribute<Set<PosixFilePermission>> attr =  
    PosixFilePermissions.asFileAttribute(perms);  
try (SeekableByteChannel sbc =  
    Files.newByteChannel(file, options, attr)) {  
// write data  
};
```

Bon



Entrées/Sorties

- **Encoder les données renvoyées dans le format attendu**
- **Fermer les descripteurs de fichiers/réseaux non utilisés**
- **Valider les paths et les noms des fichiers avant utilisation**

Notes

Chapitre 9

Valider les données

Données utilisateur

```
String tainted2 = "%3C%73%63%72%69%70%74%3E.getBytes();
```

....

....

```
String tainted = Base64.encode(tainted);
```

....

...

....

```
Pattern pattern = Pattern.compile("(%3C|<)(.*)(%3E|>)");  
if(pattern.matcher(tainted).find()) {  
    throw new ValidationException("Invalid Input");  
}
```

```
URI uri = new URI(...);
```



Mauvais

Bon



```
Pattern pattern = Pattern.compile("[\\W&&[^\\s\\.]]");  
String tainted = "%3C%73%63%72%69%70%74%3E"; // <script>  
if(!pattern.matcher(tainted).find()) {  
    throw new ValidationException( "Invalid Input");  
}  
URI uri = new URI("http://vulnerable.com/" + tainted);
```

Normalisation

```
// s peut contenir des données venant de l'utilisateur  
// \uFE64 converti en < et \uFE65 converti en > par NFKC  
String s = "\uFE64" + "script" + "\uFE65";  
  
// Validation  
Pattern pattern = Pattern.compile("[<>]");  
Matcher matcher = pattern.matcher(s);  
if(matcher.find()) {  
    System.out.println("found black listed tag");  
} else {  
    // ...  
}  
  
// Normalisation  
s = Normalizer.normalize(s, Form.NFKC);
```



Mauvais

Normalisation (suite)

Bon



```
String s = "\uFE64" + "script" + "\uFE65";  
  
// normalize  
s = Normalizer.normalize(s, Form.NFKC);  
  
//validate  
Pattern pattern = Pattern.compile("[<>]");  
Matcher matcher = pattern.matcher(s);  
if(matcher.find()) {  
    System.out.println("found black listed tag");  
    throw new IllegalStateException();  
} else {  
    // ...  
}
```

Encodage de sortie

```
public class BadOutput {  
    // description et input proviennent d'un SGBD  
    // description = "description" et  
    // input = "<script> executable code </script>"  
    public static void display(String description, String input) {  
        // affichage ou envoi a un autre processus  
    }  
}
```



Mauvais

```
public static void display(String description, String input) throws  
ValidationException {  
    ValidateOutput vo = new ValidateOutput();  
    vo.validate(description, input);  
    // affichage ou envoi a un autre processus  
}
```

Bon



Injection de commandes

```
// programmName = 'ProgramName1 || ProgramName2'  
Process proc = runtime.exec("/bin/sh" + programmName);
```



Mauvais

```
Process proc;  
int filename = Integer.parseInt(System.getProperty("program.name"))  
Runtime runtime = Runtime.getRuntime();  
switch(filename) {  
    case 1:  
        proc = runtime.exec("hardcoded\program1");  
        break; // Option 1  
    case 2:  
        proc = runtime.exec("hardcoded\program2");  
        break; // Option 2  
    default:  
        System.out.println("Invalid option!");  
        break;  
}
```

Bon



Injection SQL

```
class Login {  
    public void doPrivilegedAction(String username, String password) throws  
SQLException {  
    DriverManager.registerDriver(new  
com.microsoft.jdbc.sqlserver.SQLServerDriver());  
    Connection connection = DriverManager.getConnection(  
        "jdbc:microsoft:sqlserver://<HOST>:1433","<UID>","<PWD>");  
  
    if (connection != null) {  
        String sql = "select * from db_user where username = '" + username +  
            "' and password = '" + password + "'";  
  
        Statement stmt = connection.createStatement();  
        ResultSet rs = stmt.executeQuery(sql);  
  
        if (!rs.next()) {  
            throw new SecurityException("User name or Password incorrect");  
        }  
    }  
}
```



Mauvais

Injection SQL (suite)

Bon



```
class Login {  
    public void doPrivilegedAction(String username, String password) throws SQLException {  
        if((username.length() >= 8) || (password.length() >= 20)) {  
            DriverManager.registerDriver(new com.microsoft.jdbc.sqlserver.SQLServerDriver());  
            Connection connection = DriverManager.getConnection(  
                "jdbc:microsoft:sqlserver://<HOST>:1433","<UID>","<PWD>");  
  
            if (connection != null) {  
                String sql = "select * from db_user where username=? and password=?";  
                PreparedStatement stmt = connection.prepareStatement(sql);  
                stmt.setString(1, username);  
                stmt.setString(2, password);  
                ResultSet rs = stmt.executeQuery();  
  
                if (!rs.next()) {  
                    throw new SecurityException("User name or Password incorrect");  
                }  
            }  
        }  
    }  
}
```

Injection XML

```
private void receiveXMLStream(InputStream is, DefaultHandler defHandler) {  
    SAXParserFactory factory = SAXParserFactory.newInstance();  
    try {  
        SAXParser saxParser = factory.newSAXParser();  
        saxParser.parse(is, defHandler);  
    } catch (Throwable t) {  
        // Forward to handler  
    }  
}
```



Mauvais

Injection XML (suite)

Bon



```
private static void receiveXMLStream(InputStream inStream, DefaultHandler defHandler)
throws ParserConfigurationException, SAXException, IOException {

    SchemaFactory sf =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
    sf.setErrorHandler(new ErrorHandler());
    StreamSource ss = new StreamSource(new File("defaultschema.xsd"));
    Schema schema = sf.newSchema(ss);
    SAXParserFactory spf = SAXParserFactory.newInstance();
    spf.setSchema(schema);
    SAXParser saxParser = spf.newSAXParser();
    saxParser.parse(inStream, defHandler);
}
```

Injection XPath

```
class XpathInjection {  
    private boolean doLogin(String loginID, String password)  
        throws ParserConfigurationException, SAXException, IOException, XPathExpressionException {  
  
        DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();  
        domFactory.setNamespaceAware(true);  
        DocumentBuilder builder = domFactory.newDocumentBuilder();  
        Document doc = builder.parse("users.xml");  
  
        XPathFactory factory = XPathFactory.newInstance();  
        XPath xpath = factory.newXPath();  
        XPathExpression expr = xpath.compile("//users/user[login/text()=" +  
            loginID + "" + "and password/text()=" + password + " ]");  
        Object result = expr.evaluate(doc, XPathConstants.NODESET);  
        NodeList nodes = (NodeList) result;  
  
        // Print first names to the console  
        for (int i = 0; i < nodes.getLength(); i++) {  
            System.out.println(nodes.item(i).getNodeValue());}  
  
        return (nodes.getLength() >= 1);  
    }  
}
```



Mauvais

Injection Xpath (suite)

Bon



```
declare variable $loginID as xs:string external;
declare variable $password as xs:string external;//users/user[@loginID=
$loginID and @password=$password]

Document doc = new Builder().build("users.xml");
XQuery xquery = new XQueryFactory().createXQuery(new File("login.xry"));

Map queryVars= new HashMap();

queryVars.put("loginid", "Utah"); // User name hardcoded for illustrative purposes
queryVars.put("password", "securecoding"); // Password hardcoded for illustrative purposes

Nodes results = xquery.execute(doc, null, queryVars).toNodes();

for (int i = 0; i < results.size(); i++) {
    System.out.println(results.get(i).toXML());
}
```

Injection LDAP

```
// String userSN = "S*"; // Invalid
// String userPassword = "*"; // Invalid
public class LDAPInjection {
    private void searchRecord(String userSN, String userPassword) throws NamingException {
        Hashtable<String, String> env = new Hashtable<String, String>();
        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
        try {
            DirContext dctx = new InitialDirContext(env);

            SearchControls sc = new SearchControls();
            String[] attributeFilter = {"cn", "mail"};
            sc.setReturningAttributes(attributeFilter);
            sc.setSearchScope(SearchControls.SUBTREE_SCOPE);
            String base = "dc=example,dc=com";

            // The following resolves to (&(sn=S*)(userPassword=*)
            String filter = "(&(sn=" + userSN + ")(userPassword=" + userPassword + "))";

            NamingEnumeration<?> results = dctx.search(base, filter, sc);
            while (results.hasMore()) {
                SearchResult sr = (SearchResult) results.next();
                Attributes attrs = sr.getAttributes();
                Attribute attr = attrs.get("cn");
                System.out.println(attr.get());
                attr = attrs.get("mail");
                System.out.println(attr.get());
            }
            dctx.close();
        } catch (NamingException e) {
            // Handle
        }
    }
}
```



Mauvais

Injection LDAP (suite)

Bon



```
// String userSN = "Sherlock Holmes"; // Valid
// String userPassword = "secret2"; // Valid

sc.setSearchScope(SearchControls.SUBTREE_SCOPE);
String base = "dc=example,dc=com";

if(!userSN.matches("[\\w\\s]*") || !userPassword.matches("[\\w]*")) {
    throw new IllegalArgumentException("Invalid input");
}

String filter = "(&(sn = " + userSN + ")(userPassword=" + userPassword +
"))";
```

Injection de code

```
// Windows based target's file path is being used
String firstName = "dummy\"; var bw = new JavaImporter(java.io.BufferedReader);
    var fw = new JavaImporter(java.io.FileWriter);
    with(fw) with(bw) {
        bwr = new BufferedWriter(new FileWriter("c://somepath//somefile.txt"));
        bwr.write("some text"); bwr.close(); } // ";
evalScript(firstName);

private static void evalScript(String firstName) throws ScriptException {
    ScriptEngineManager manager = new ScriptEngineManager();
    ScriptEngine engine = manager.getEngineByName("javascript");
    engine.eval("print("+ firstName + ")");
}
```



Mauvais

Injection de code (suite)

Bon



```
// First sanitize firstName (modify if the name may include special
// characters)

if(!firstName.matches("[\\w]*")) { // String does not match white-listed
    characters
    throw new IllegalArgumentException();
}

// Restrict permission using the two-argument form of doPrivileged()
try {
    AccessController.doPrivileged(new PrivilegedExceptionAction() {
        public Object run() throws ScriptException {
            engine.eval("print("+ firstName + ")");
            return null;
        }
    }, RestrictedAccessControlContext.INSTANCE);
} catch(PrivilegedActionException pae) {
    // Handle
}
```

URI

```
public class Filter {  
    public static void main(String[] args) throws  
MalformedURLException {  
    final URL allowed = new URL("http://mailwebsite.com");  
    if(!allowed.equals(new URL(args[0]))) {  
        throw new SecurityException("Access Denied");  
    }  
    // Else proceed  
}  
}
```



Mauvais

Bon



```
public class Filter {  
    public static void main(String[] args) throws MalformedURLException,  
URISyntaxException {  
    final URI allowed = new URI("http://mailwebsite.com");  
    if(!allowed.equals(new URL(args[0]))) {  
        throw new SecurityException("Access Denied");  
    } }  
}
```

Validations

- **Valider toute donnée utilisateur avant utilisation**
 - Penser à une approche positive
 - Penser à rejeter le contenu si il n'est pas cohérent
- **Normaliser les données avant validation**
- **Encoder dans le bon format les données lors du renvoi au client**

Chapitre 10

Gestion des erreurs et journalisation

Prévenir les exceptions lors de logs

```
try {  
    // ...  
} catch (SecurityException se) {  
    System.err.println(se);  
}
```



Mauvais

Meilleur



```
try {  
    // ...  
} catch(SecurityException se) {  
    logger.log(Level.SEVERE, se);  
}
```

Ne pas finir de façon abrupte

```
class TryFinally {  
    private static boolean doLogic() {  
        try {  
            throw new IllegalStateException();  
        } finally {  
            System.out.println("logic done");  
            return true;  
        }  
    }  
}
```



Mauvais

Meilleur



```
class TryFinally {  
    private static boolean doLogic() {  
        try {  
            throw new IllegalStateException();  
        } finally {  
            System.out.println("logic done");  
        }  
    }  
}
```

Perte d'informations

```
class ExceptionExample {  
    public static void main(String[] args) throws FileNotFoundException {  
        FileInputStream fis =  
            new FileInputStream(System.getenv("APPDATA") + args[0]);  
    }  
}
```



Mauvais

Perte d'informations (suite)

Meilleur



```
class ExceptionExample {  
    public static void main(String[] args) {  
  
        File file = null;  
        try {  
            file = new File(System.getenv("APPDATA") +  
                args[0]).getCanonicalFile();  
            if (!file.getPath().startsWith("c:\\\\homepath")) {  
                System.out.println("Invalid file");  
                return;  
            }  
        } catch (IOException x) {  
            System.out.println("Invalid file");  
            return;  
        }  
  
        try {  
            FileInputStream fis = new FileInputStream(file);  
        } catch (FileNotFoundException x) {  
            System.out.println("Invalid file");  
            return;  
        }  
    }  
}
```

Exposition de données sensibles

Exception Name	Description of information leak or threat
java.io.FileNotFoundException	Underlying file system structure, user name enumeration
java.sql.SQLException	Database structure, user name enumeration
java.net.BindException	Enumeration of open ports when untrusted client can choose server port
java.util.ConcurrentModificationException	May provide information about thread-unsafe code
javax.naming.InsufficientResourcesException	Insufficient server resources (may aid DoS)
java.util.MissingResourceException	Resource enumeration
java.util.jar.JarException	Underlying file system structure
java.security.acl.NotOwnerException	Owner enumeration
java.lang.OutOfMemoryError	DoS
java.lang.StackOverflowError	DoS

Logs/Erreurs

- Prévenir l'imprévisible
- Utiliser un framework standard de logs
- Ne pas logguer d'éléments sensibles (CB, passwd,)
- Découper les catégories de logs :
 - Audit
 - Erreurs
 - Debug
 - Accès

Notes

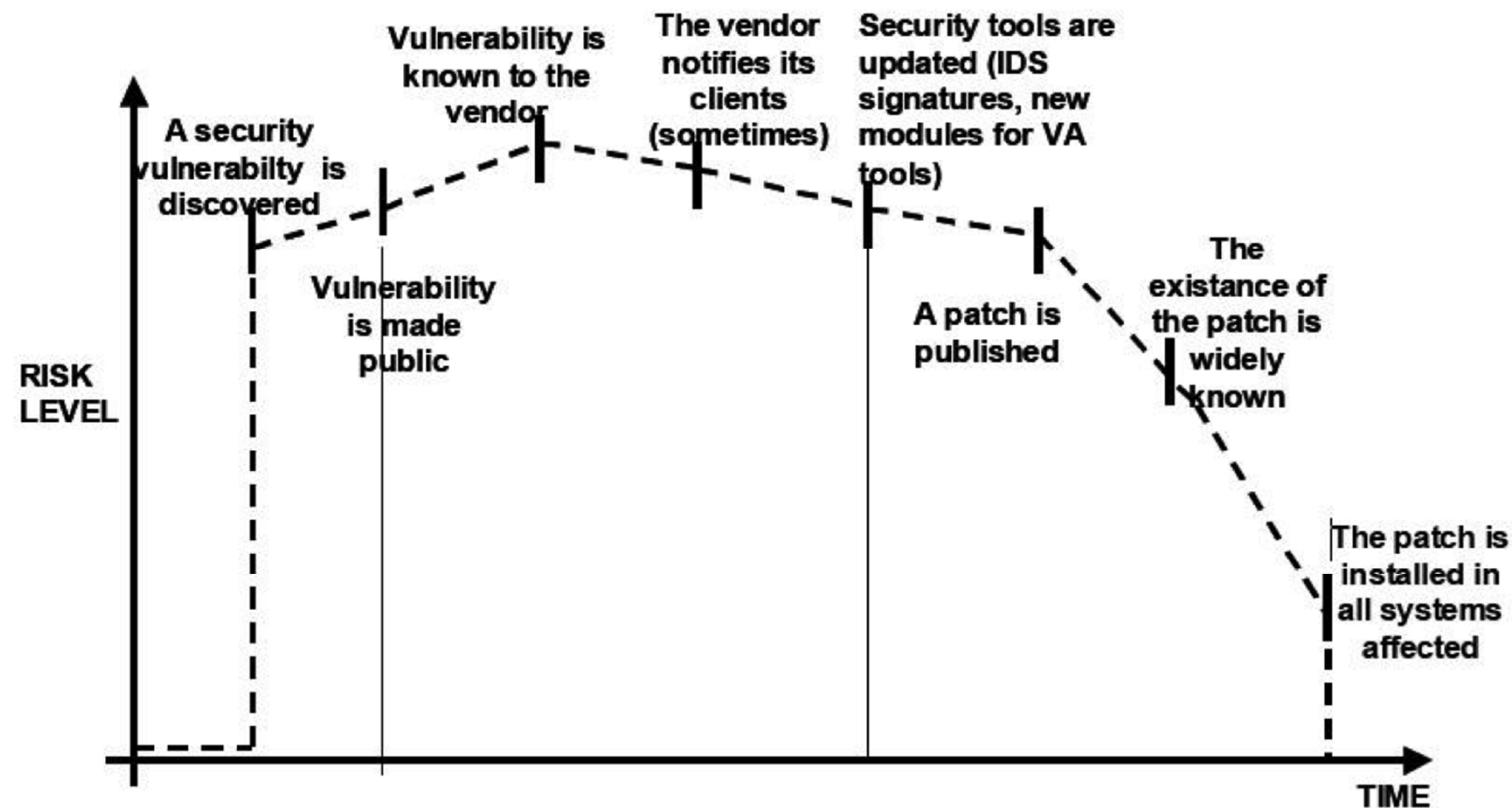
Chapitre 11

La revue de code

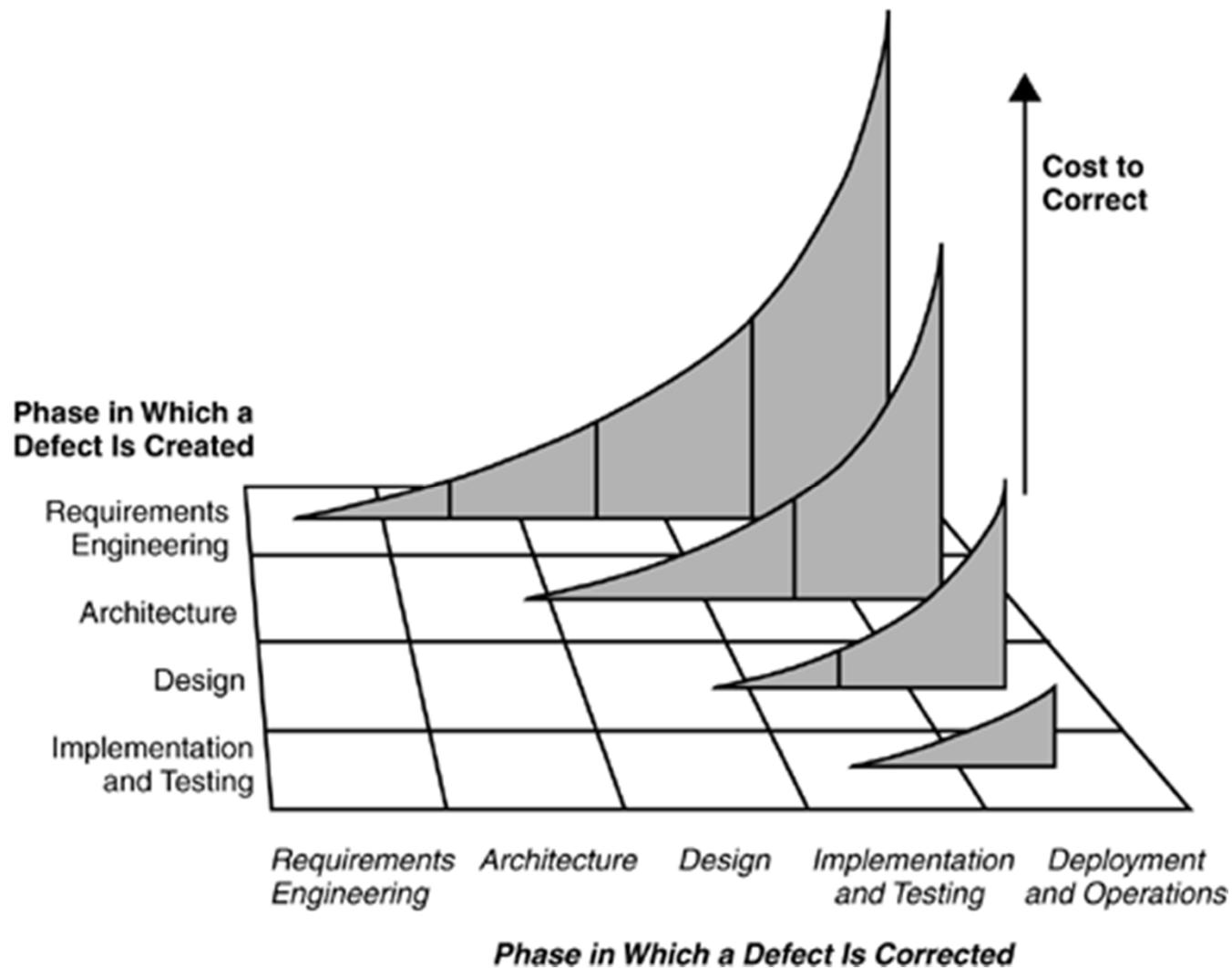
Agenda

- **Méthodologie de revue de code**
- **Les éléments à rechercher**
- **Outils**

L'exposition à une vulnérabilité



Coût de la correction d'une faille



Scope – Approche Black/White/Gray Box

- **Blackbox (aucune connaissance) :**
 - Inefficient et incomplet
 - Ne sert pas à établir un niveau de sécurité
- **GrayBox (connaissance uniquement d'un login par ex) :**
 - Mêmes approche que la BlackBox
- **WhiteBox (connaissance totale des sources) :**
 - Permet d'estimer réellement le risque/niveau de sécurité
 - Nécessité de disposer des documents du projets (de la conception au développement)
 - Pas forcément plus consommateur de temps

La revue de code : Définition

- **La revue de code permet, en mode boîte blanche :**
 - De découvrir l'utilisation faite de fonctions sensibles ou mal sécurisées
 - De corriger avant la mise en production le code vulnérable
 - De mettre en place les frameworks de sécurité nécessaires à toutes les applications

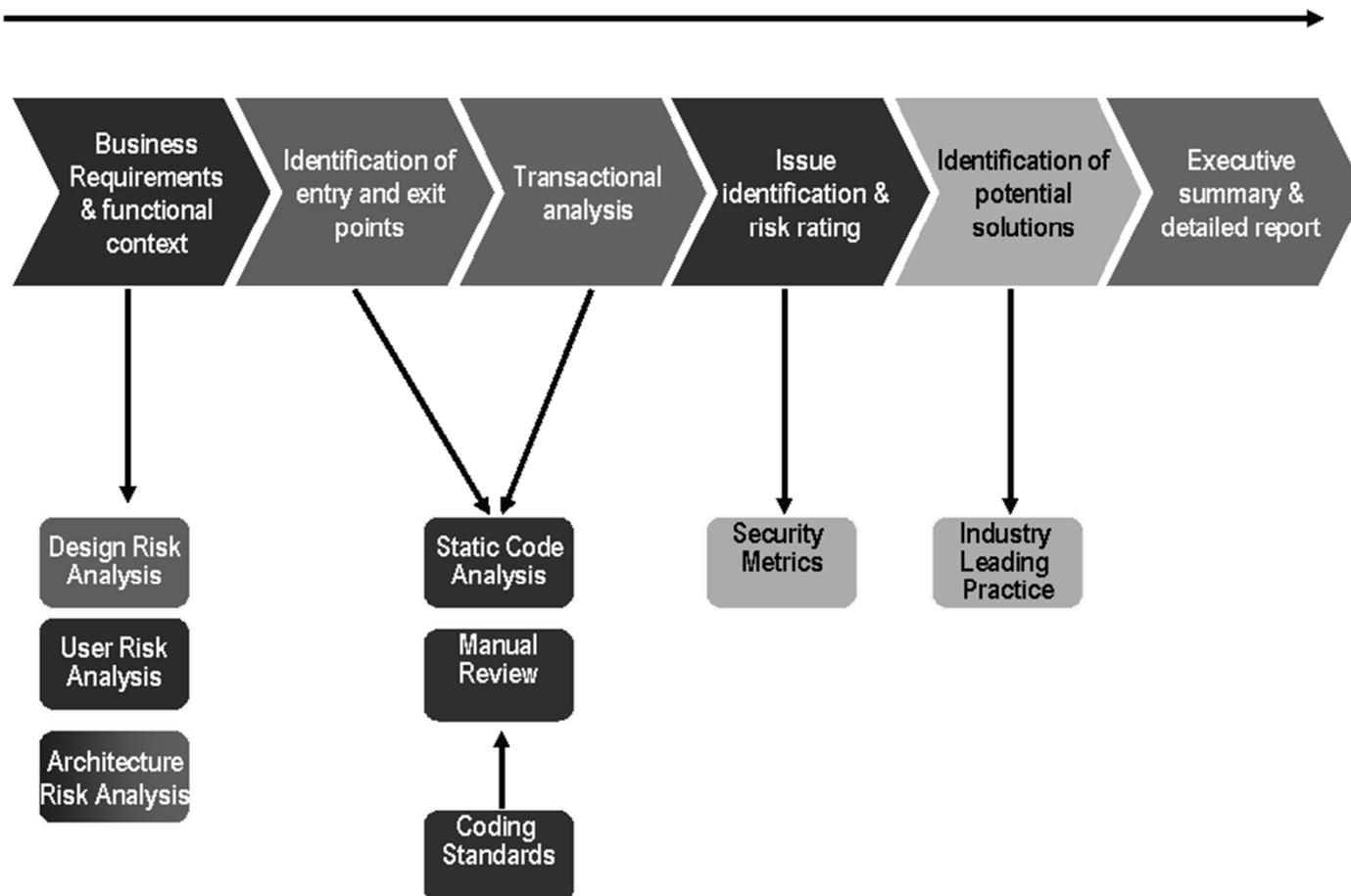
La revue de code

► Méthodologie d'analyse de code

Quelques fonctions Java à auditer

Secure Code review process

Secure Code review process – Operational process



Phases

- **Scope :**
 - Définir l'étendue de la revue
- **Planification:**
 - Déterminer les priorités
- **Exécution :**
 - Procéder à la revue
- **Reporting :**
 - Générer un rapport

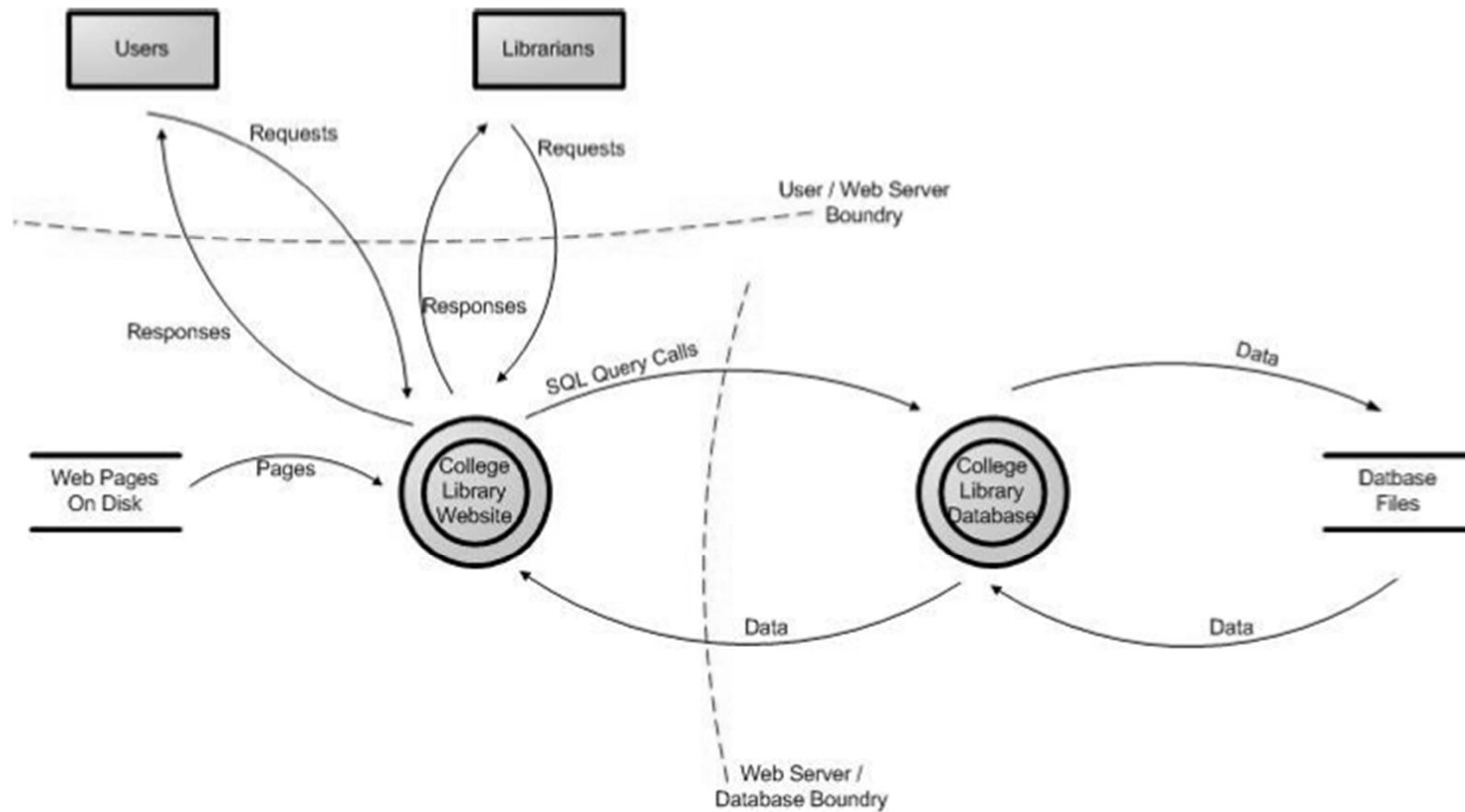
Scope

- **Comprendre la surface d'attaque**
 - Navigateur,
 - Cookies
 - Properties
 - Fichiers
 -

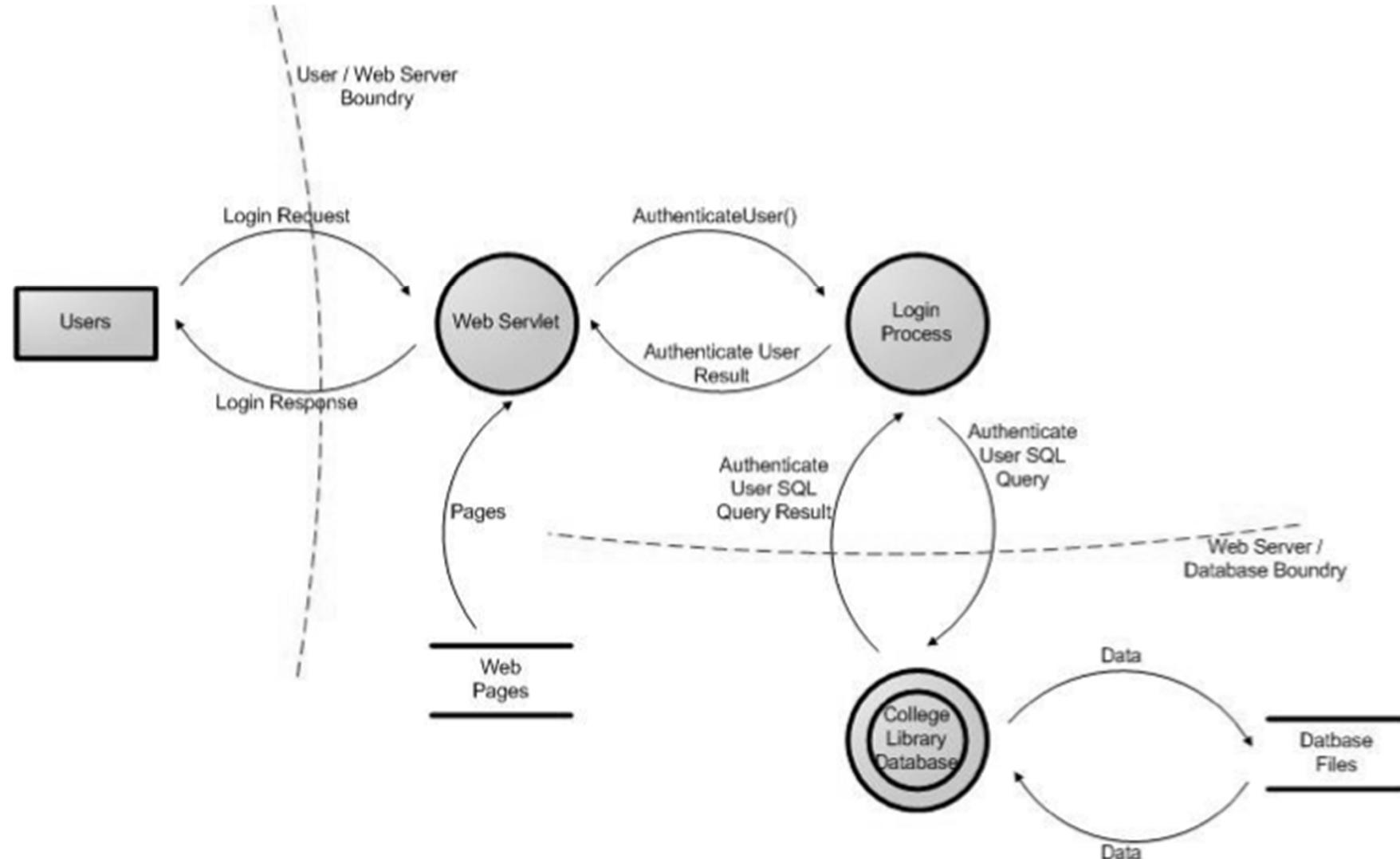
Modéliser les processus

- **Décomposer l'application**
 - Modéliser les traitements (fonctions métiers, ...)
 - Modéliser les dépendances externes (SGBD, ...)
 - Définir les points d'entrée et de sortie
 - Définir les liens entre les entrées et les sorties
 - Définir les actifs utilisés
 - Définir les niveaux de sécurité

Exemple 1



Exemple 2



Modéliser les menaces

- Penser malveillant !
- Détailler l'architecture de sécurité en terme de composants, connexions et contrôles de sécurité
- Modéliser les menaces, les attaques, les vulnérabilités, les impacts techniques et business
- => Un risque existe lorsqu'une menace utilisant une attaque sur une vulnérabilité génère un impact business

Deux types de menaces

- **Les menaces externes :**
 - Utilisateur anonyme, authentifié, privilégié
 - Gouvernements
 - Partenaires, ISP, etc.
- **Les menaces internes :**
 - Administrateur système, applicatif, etc.
 - Utilisateur interne
 - Manager, directeur

Les menaces internes représentent encore plus de 50 %

Catégoriser les attaques

- **Via la méthode STRIDE (par exemple) :**
 - Spoofing
 - Tampering
 - Repudiation
 - Information disclosure
 - Denial of Service
 - Elevation of Privilege

Exemples d'attaques catégorisées

STRIDE Threat List		
Type	Examples	Security Control
Spoofing	Threat action aimed to illegally access and use another user's credentials, such as username and password	Authentication
Tampering	Threat action aimed to maliciously change/modify persistent data, such as persistent data in a database, and the alteration of data in transit between two computers over an open network, such as the Internet	Integrity
Repudiation	Threat action aimed to perform illegal operation in a system that lacks the ability to trace the prohibited operations.	Non-Repudiation
Information disclosure.	Threat action to read a file that they were not granted access to, or to read data in transit.	Confidentiality
Denial of service.	Threat aimed to deny access to valid users such as by making a web server temporarily unavailable or unusable.	Availability
Elevation of privilege.	Threat aimed to gain privileged access to resources for gaining unauthorized access to information or to compromise a system.	Authorization

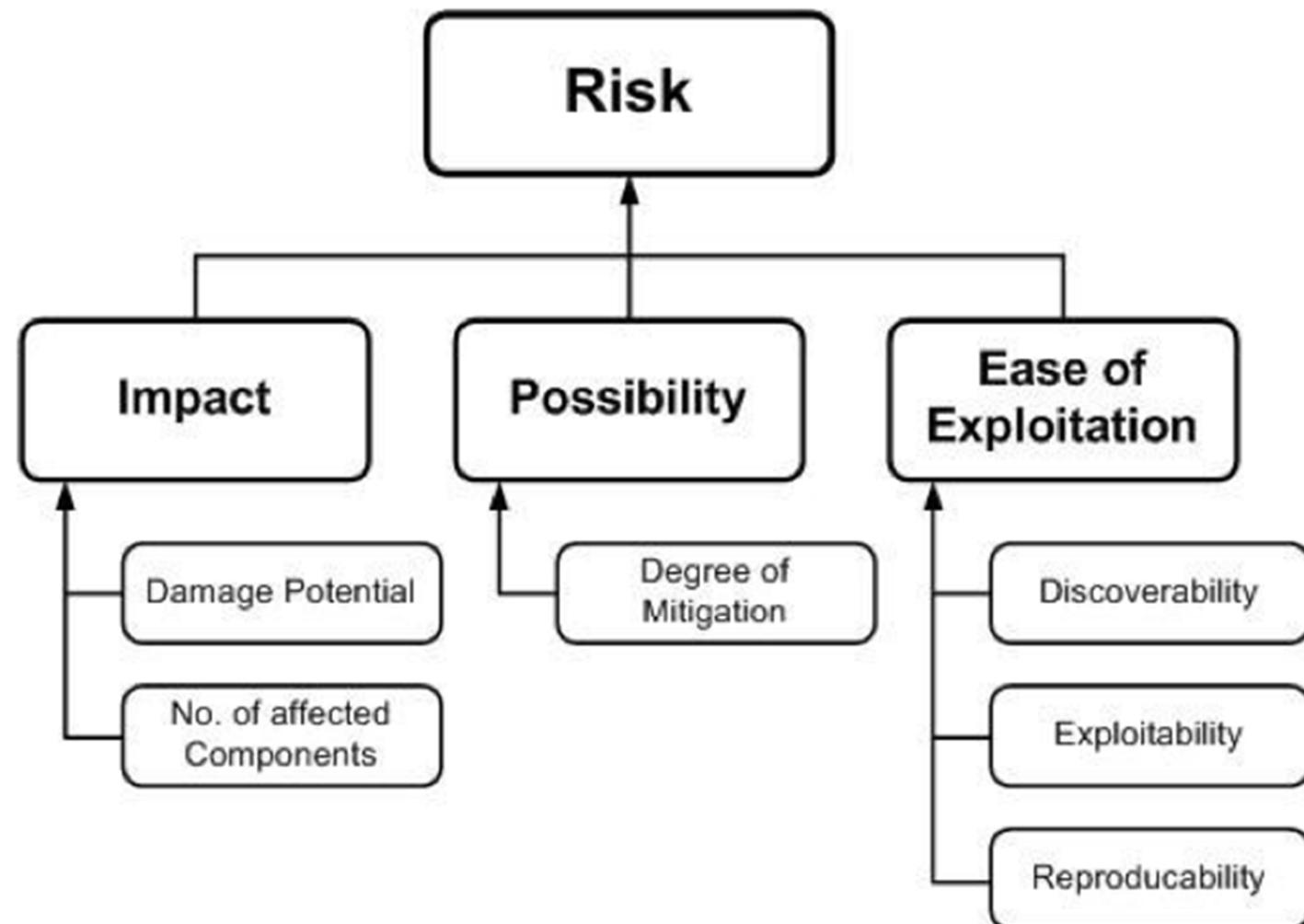
Contrôles et menaces

- **Identifier les contrôles limitant les attaques**
 - Vérifier qu'ils existent.
 - Vérifier qu'ils sont correctement écrits.
 - Vérifier qu'ils sont correctement utilisés.
- **Analyser les attaques dans le contexte**
 - Vérifier l'impact après contrôles

Prioriser / calculer le risque

- Il est impossible de disposer d'assez de temps pour TOUT tester
- Il est nécessaire de tester en priorité les risques les plus importants (et non pas les menaces les plus fortes)
- Il est nécessaire de redéfinir une priorité régulièrement en fonction des données découvertes
- Les checklists sont utiles, mais uniquement comme base de départ

Calculer le risque



Identifier les contre-mesures

- **En fonction des menaces**
- **En fonction du risque**

Réduire l'impact

- **Plusieurs solutions :**
 - Ne rien faire
 - Informer sur le risque
 - Réduire le risque
 - Accepter le risque
 - Transférer le risque

Quelle est la meilleure approche

- **Top down (chercher pour) :**
 - Passer outre les mécanismes de contrôles
 - Découvrir de mauvaises configurations

- **Bottom up (noter) :**
 - L'usage de fonctions telle que document.write();
 - Des actions Struts disposant de validate=false();

La revue de code

Méthodologie d'analyse de code

► **Quelques fonctions Java à auditer**

Entrées/Sorties

- **Java.io**
- **java.util.zip**
- **java.util.jar**
- **FileInputStream**
- **ObjectInputStream**
- **FilterInputStream**
- **PipedInputStream**
- **SequenceInputStream**
- **StringBufferInputStream**
- **BufferedReader**
- **ByteArrayInputStream**
- **CharArrayReader**
- **File**
- **ObjectOutputStream**
- **PipedOutputStream**
- **StreamTokenizer**
- **getResourceAsStream**
- **java.io.FileReader**
- **java.io.FileWriter**
- **java.io.RandomAccessFile**
- **java.io.File**
- **java.io.FileOutputStream**
- **mkdir**
- **renameTo**

Servlets

- javax.servlet.*
- getParameterNames
- getParameterValues
- getParameter
- getParameterMap
- getScheme
- getProtocol
- getContentType
- getServerName
- getRemoteAddr
- getRemoteHost
- getRealPath
- getLocalName
- getAttribute
- getAttributeNames
- getLocalAddr
- getAuthType
- getRemoteUser
- getCookies
- isSecure
- HttpServletRequest
- getQueryString
- getHeaderNames
- getHeaders
- getPrincipal
- getUserPrincipal
- isUserInRole
- getInputStream
- getOutputStream
- getWriter
- addCookie
- addHeader
- setHeader
- setAttribute
- putValue
- javax.servlet.http.Cookie
- getName
- getPath
- getDomain
- getComment
- getMethod
- getPath
- getReader
- getRealPath
- getRequestURI
- getRequestURL
- getServerName
- getValue
- getValueNames
- getRequestedSessionId

XSS & Response Splitting

- `javax.servlet.ServletOutputStream.print`
- `javax.servlet.jsp.JspWriter.print`
- `java.io.PrintWriter.print`
- `javax.servlet.http.HttpServletResponse.sendRedirect`
- `addHeader, setHeader`

Redirection

- **sendRedirect**
- **setStatus**
- **addHeader, setHeader**

SQL

- **jdbc**
- **executeQuery**
- **select**
- **insert**
- **update**
- **delete**
- **execute**
- **executestatement**
- **createStatement**
- **java.sql.ResultSet.getString**
- **java.sql.ResultSet.getObject**
- **java.sql.Statement.executeUpdate**
- **java.sql.Statement.executeQuery**
- **java.sql.Statement.execute**
- **java.sql.Statement.addBatch**
- **java.sql.Connection.prepareStatement**
- **java.sql.Connection.prepareCall**

Les Outils

- **OpenSource :**
 - OWASP LAPSE+
 - OpenSource
 - Plugins ECLIPSE
 - Permet de vérifier sur la base du Top10
 - Google CodePro Analytix
 - Permet aussi d'avoir des informations orienté qualimétrie de code
 - SonarSource :
 - Intégration des règles de PM, findbugs
 - Plugins Top10 OWASP Commerciaux
 - HP Fortify
 - CheckMarx
 - IBM AppScanSource
 - Veracode
 - Coverity

Chapitre 12

Résumé du cours

Résumé du cours

Dans ce cours, nous avons :

- Présenté les bonnes pratiques permettant de sécuriser le développement d'une application
- Présenté les erreurs d'utilisation de code
- Fourni les éléments nécessaire à une revue de code sécurité