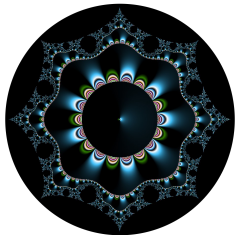
Sécurité des applications

Cycles de développement

Thibaut et Corinne HENIN



www.arsouyes.org
@arsouyes

INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
CENTRE VAL DE LOIRE

Pourquoi on code des vulnérabilités ?

Par Négligence

« Tant que ça marche, on ne touche à rien »

Par conservatisme

« On a toujours codé comme ça ici ! »

Par dette technique

« C’est trop long de corriger proprement »

Par Incompétence

« Je ne pouvais pas le savoir »

Par Paresse

« C’est trop chiant »

La perfection est impossible

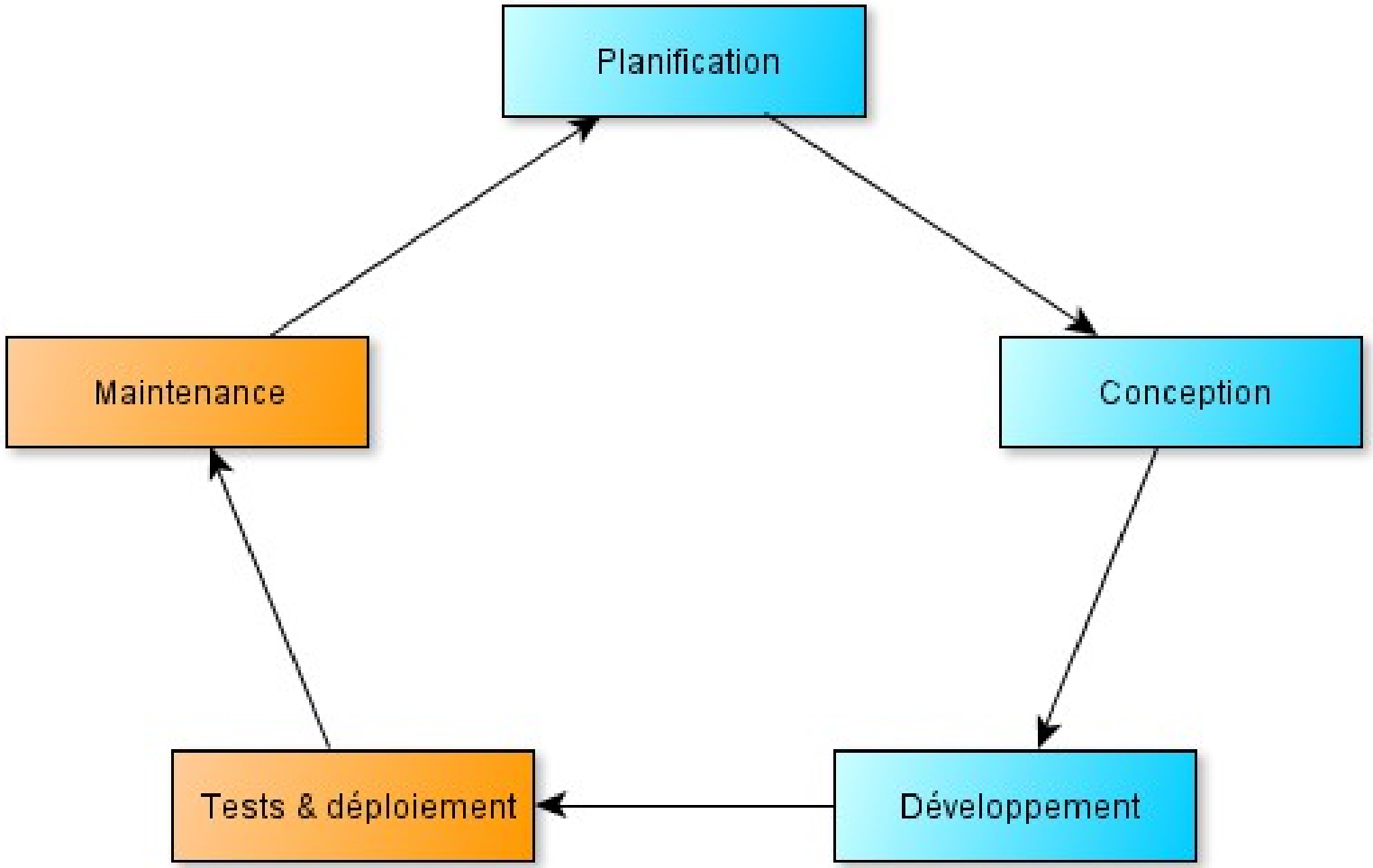
« Mince, je l'avais pas vu »

Sommaire

- **Organiser la sécurité**
  - Cycle de développement
  - Planification
  - Conception
  - Développement
  - Test et déploiement
  - Maintenance
  - Ressources Humaines
- **Projets existants**
  - Inventorier et découper
  - Durcir les interfaces
  - Nettoyer les responsabilités

Cycles de développement

Phases dans la vie d’un logiciel



Planification

Analyse et spécifications

PSSI - Inventaire

- Biens à protéger
    - Quelles données ?
    - Quelles fonctionnalités ?
- Surface d'attaque
    - Entrées utilisateur,
    - Environnement d'exécution
- Propriétés de sécurités
    - Confidentialité, Intégrité, Disponibilité,
    - Traçabilité,
    - …

PSSI : analyse de risques

- Quelles menaces ?
    - Quels compromissions ?
- Quelles vraisemblance ?
    - Réaliste ou fantaisiste ?
- Quel impact ?
    - Pour les clients ?
    - Pour l’entreprise ?

Exemple d’erreur

- **Idée commerciale :**
    - « Une réservation est remboursée si annulée »

- **Exploitation :**
    - Ola vs Uber vs Lyft
    - Réservation en masse puis annulations
    - Déni de Service du concurrent

Conception

Vulns ici : slowloris, crypto, …

Architecture logicielle

- **Découpage en composants**
  - Définition des frontières / interfaces
  - Couplage afférent / efférent

- **Deux types :**
  - Fournisseur de sécurité
  - Utilisateur de sécurité
  - **Ne pas mélanger !**

Développement

Code propre

- Écriture vs Relecture

- Faciliter la maintenance :
    - Revues
    - Améliorations
    - Corrections

- Réduire les risques
    - Propre et Simple est synonyme de produit sûr

Coding style guideline

- Niveau 0
    - Prérequis

- Exemples :
    - C : Kernighan et Ritchie (K&R)
    - PHP : PSR-2 (K&R)

- Important :
    - Consensus et adoption par les développeurs
    - Utiliser un standard

Intérêt d’un standard

- Conventions graphiques communes
    - Cf. typographie, orthographe, grammaire, …
- Guide l’œil vers les points d’intérêt
    - Cf. ergonomie, web design
- Lecture plus rapide et plus facile

Simplicité : taille

- Taille du code :
    - Largeur : 80 caractère maximum
    - Longueur : 20 lignes

- Taille Mémoire Humaine de Travail :
    - G. A. Miller, **« The magical number 7 plus or minus two »**, 1956

Simplicité : complexité

- **Complexité cyclomatique**
  - McCabe, 1976
  - Nombre de choix dans un algorithme
  - Max : 10

- **Complexité Npath**
  - Nejmeh, 1988
  - Nombre de chemins non cycliques
  - Max : 200

Secure Coding Guidelines

- Niveau 1
  - Nécessaire
- Exemples :
  - SEI CERT Coding Standards
- Vérification manuelle
  - Core review, Pair programming, …

POO – Forever

- Encapsulation
    - Regrouper les données et les fonctions
    - Une seule responsabilité
- Masquarade
    - Cacher les détails
    - Empêcher les contournements
- Boite noire
    - Sécurité par construction

POO - exemple

```
class ShellExec {

    private $cmd ;

    public function __construct($cmd) {
        $this->cmd = escapeshellargs($cmd) ;
    }

    public function args($arg) {
        $this->cmd .= " " . escapeshellargs($arg) ;
    }

    public function exec() {
        return shell_exec($this->cmd) ;
    }
}
```

POO – RAII Pattern

- **Principe :**
  - Allocation : constructeur
  - Libération : destructeur
  - Erreur : exceptions
  - Objet toujours valide

- **Limitations :**
  - Libération des objets lorsqu'on quitte la portée
    - **OK :** C++, PHP, …
    - **KO :** Java, Python, NodeJS, …

POO – RAII Pattern 1/4

```
interface Lockable {
    public function lock() ;
    public function unlock() ;
}
```

POO – RAII Pattern 2/3

```
class FileLock implements Lockable {

  private $handler = null;

  public function __construct($name)
  {
    $handler = fopen("/tmp/$name", "c") ;
    if (! $handler) {
      throw new \Exception(…) ;
    }
    $this->handler = handler ;
  }
```

```
  public function __destruct()
  {
    fclose($this->handler) ;
  }

  public function lock()
  {
    flock($this->handler, LOCK_EXCL) ;
  }

  public function unlock()
  {
    flock($this->handler, LOCK_UN) ;
  }
}
```

POO – RAII Pattern 3/3

```
class LockGuard {

    private $lock;

    public function __construct(Lockable $lock)
    {
        $this->lock = $lock ;
        $this->lock->lock() ;
    }

    public function __destruct()
    {
        $this->lock->unlock() ;
    }

}
```

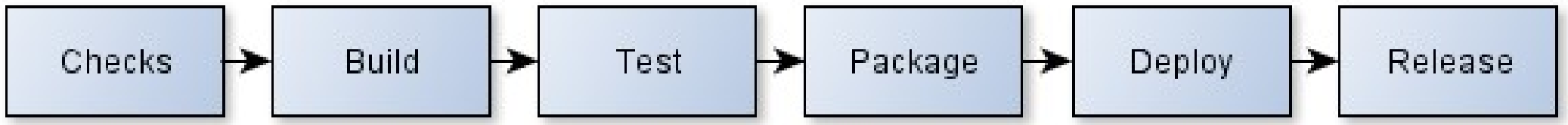
```
function someCriticalStuff() {
    $lockguard = new LockGuard(
        new FileLock("lock_name")) ;
    // DO Stuffs
}
```

POO – RAII Pattern

- Gestion mémoire
- Gestion des fichiers
- Mutexes
- Connexions aux bases de données

Test et déploiement

Déploiement continu



Analyse statique de code

- Convention Syntaxiques :
    - i.e. Indent, phpcs/phpcbf, …
- Métrologie de code :
    - i.e. phpmetrics
- Mauvaises pratiques
    - Option de compilation (i.e. gcc –Wall)
    - PMD (et clones : phpmd)

`https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis`

Audit Externe

- Principe de l’audit :
    - Chercher des vulnérabilités

- Deux types :
    - Boite blanche (accès au code source et à la documentation)
    - Boite noire (accès au logiciel seul)

- Externe > Interne
    - Regard neuf
    - Compétences

Mises à jours des dépendances

- Inventaire des dépendances
    - Logicielles / matérielles
- Veille de mise à jours
    - Amélioration
    - Sécurité
- Si possible :
    - Automatique
    - Intégrée au système de « build »

Maintenance

Mises à jours du produit

- Automatiques vs Manuelles

- Sécurisée ?

    - **RSSI : Intégrité**
        - version « officielle »

    - **Client : Disponibilité**
        - Disponible et facile à installer

    - **Éditeur : Confidentialité**
        - Teneur des correctifs

Un 0day sur notre produit !?

- No Disclosure
- Responsible Disclosure
- Full Disclosure

Ressources humaines

Mise à jours des compétences

- Formation continue
    - Centres agréés
    - Certifications
- Auto-formation
    - Apprentissage (blogs, conférences, …)
    - Entraînement (challenges)

Pair Programming

- Programmation en binôme
  - *Driver* qui code
  - *Observer* qui assiste
- Extrême programming

Pair Programming

Avantages

- Plus rapide
  - 80 % à 57 %
- Meilleure qualité
- Aspect Social

Warnings

- Sentiment de gaspillage
- Binômes disfonctionnels
  - Silence, désintérêt, effacement
- Télétravail

Culture de la sécurité

« Do I Implements »

vs.

« Do I Understand »

Code existant

Inventorier et découper

- Définir les frontières
  - Interfaces, packages, ...
- Biens à protéger
  - Ensembles de données
  - Groupes de fonctionnalités

Durcir les interfaces

- Filtrer les entrées/sorties
  - *i.e.* Données utilisateurs

- Sécuriser l'API
  - Fonctionnellement

Nettoyer les responsabilités

- Refactoriser
  - « Responsabilité unique »
  - « Ensapsulation et Masquage »
  - « KISS »