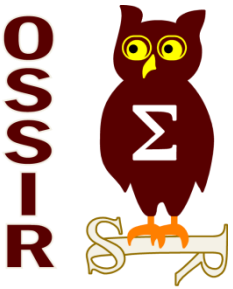
Retour d'expérience sur les outils d'audit d'applications Web en « boite noire »

JSSI 2012 – 13 mars 2012



Laurent Butti – Orange France DMGP/PORTAIL

laurent.butti@orange.com

orange™

À propos

- Expérience en R&D (Orange Labs)
  - > Sécurité des réseaux
  - > Audits de sécurité
  - > Recherche de vulnérabilités
- Rôle opérationnel (Orange Portails)
  - > Amélioration des principes de sécurité au niveau développement
- Orange Portails
  - > Développement et hébergement d'applications Web
  - > Moteur de recherche, annuaire 118712.fr, portail généraliste
  - > Services aux abonnés et sites événementiels

Agenda

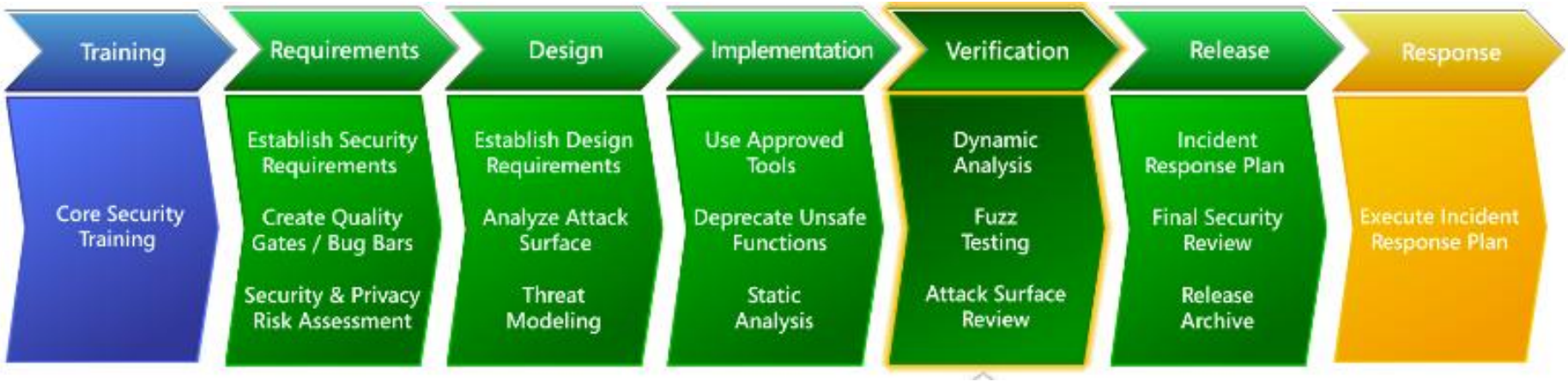
- Contexte
- Intégration des principes de sécurité
- L'audit en « boite noire » : principes et outils
- Retour d'expérience sur ce domaine avec un focus sur les « Web Scanners »
- Conclusions

Contexte

- Les applications Web sont intrinsèquement exposées
  - > Peu ou pas de filtrage applicatif
  - > Accès privilégiés : données sensibles (BDD)
  - > Responsabilité dévolue entièrement aux développeurs
- Attaques à la fois ciblées et non ciblées
  - > Attaques horizontales : *script kiddies*
  - > Attaques coordonnées vers une cible (activisme ou pas…)
- Conséquences parfois lourdes
  - > Récupération ou altération de données
  - > Compromission système

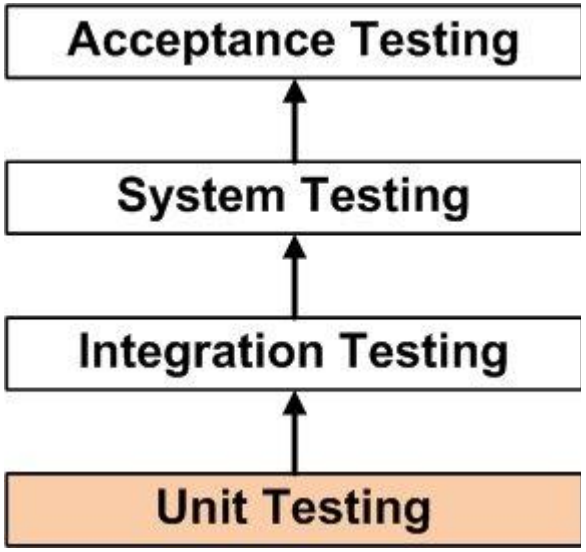
Cycle de développement logiciel (SDLC)

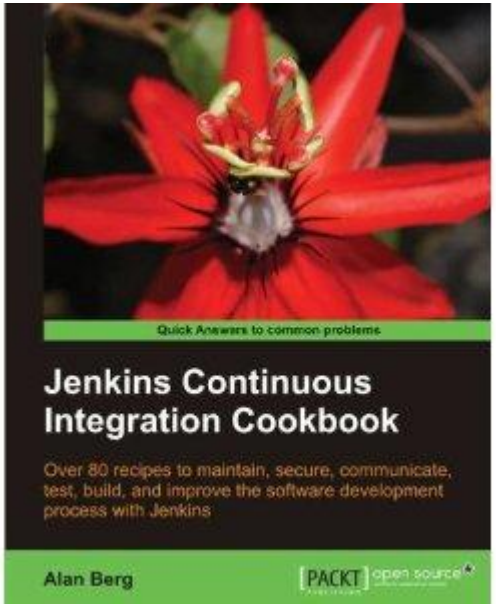
- « Security is a Process, not a Product » (B. Schneier)
  - > Exemple : SDL de Microsoft

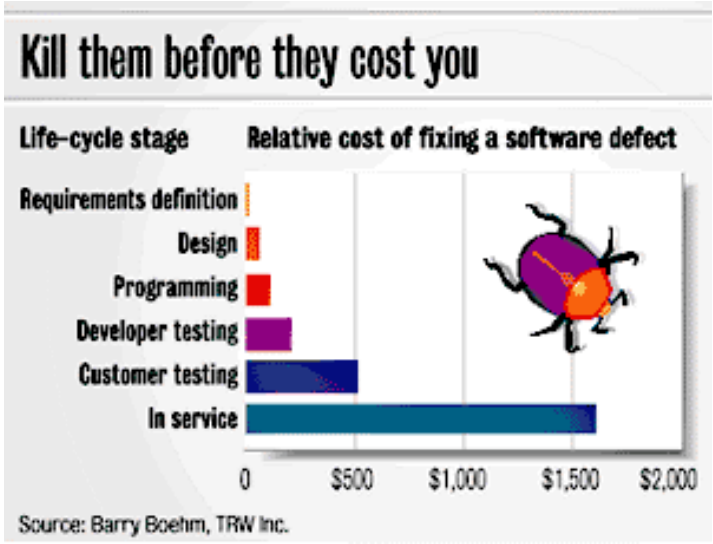


- Les outils sont à la disposition des processus pour fluidifier les rouages

Cycle de développement logiciel (SDLC)







- Influence positive de la qualité de développement sur le niveau de sécurité

La phase de vérification

- Vérification de conformité par rapport à un référentiel
  - > Top 10 de l'OWASP (sécurité Web)
  - > Expérience de l'auditeur
  - > Exigences internes ou contractuelles

- Étape essentielle
  - > Point de passage obligé si l'application suit le SDLC
  - > Positionnée durant la phase de qualification
  - > Phase itérative avec l'équipe de développement (mise en œuvre des correctifs)

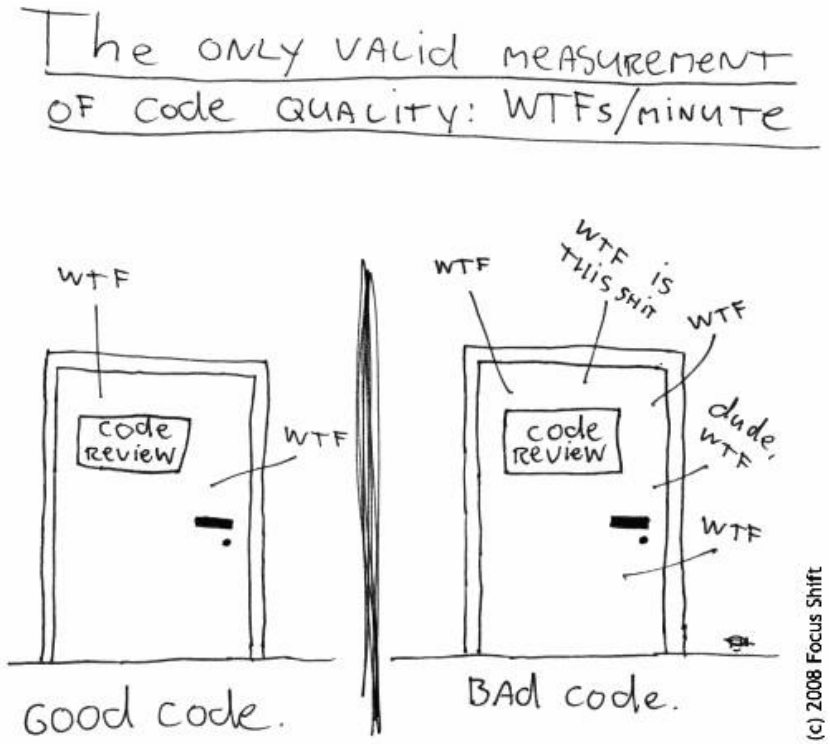
La phase de vérification : comment ?

Audit “classique”	Audit “pentest”
Découverte des faiblesses et vulnérabilités	Trouver un ou plusieurs chemins d’exploitation
Nécessité d’être exhaustif	Pas de nécessité d’être exhaustif
Prouver la faiblesse	Prouver l’exploitabilité

- Approche SDLC : identifier **tous** les problèmes **avant** mise en production

La phase de vérification : comment ?

Audit “boite blanche”	Audit “boite noire”
Connaissance complète : spécifications, architecture, application (code source), hébergement (configuration)	Pas ou peu de connaissance sur la cible
Audit code source, configuration	Audit en aveugle des points d’entrée



Les outils : pourquoi faire ?

- Objectifs
  - > Améliorer l'efficacité des audits manuels (assister l'auditeur)
  - > Industrialiser l'approche audit
  - > Couvrir des applications non auditées manuellement (première passe) ?

- Les outils sont à la disposition de l'auditeur
  - > L'auditeur interprète les résultats de l'outil
  - > L'auditeur se focalise alors sur les failles fonctionnelles

- Connaître les avantages et inconvénients des outils : les utiliser à bon escient !

Les outils « boite noire »

Web Scanners	• Crawling, brique d'injection, algorithmes de découverte de failles, reporting
Proxies « intrusifs »	• Interception et altération des flux (audit manuel ou semi-automatique)
Modules navigateurs	• Mêmes principes que précédemment mais avec les primitives des navigateurs
Outils orientés exploitation	• Si faille avérée, outils d'aide à l'exploitation

Les outils « boite noire »

Web Scanners	• Généralistes Open Source : w3af, arachni, skipfish, wapiti… • Dédiés Open Source : sqlmap, xsser… • Commerciaux : IBM AppScan, Acunetix WVS, NetXpose, NTOSpider, NetSparker, HailStorm…
Proxies « intrusifs »	• Paros Proxy, Burp Proxy (Free)…
Modules navigateurs	• XSS-Me, SQL Inject-Me, DOMinator…
Outils orientés exploitation	• modules Metasploit, SQLmap, BeEF, XSSF…

Principes de fonctionnement d'un « web scanner »

Crawler	• Recherche des points d'entrée de l'application • Fonctionnalités de gestion de session (cookie, authentification…)
Brique d'injection	• Support des vecteurs d'injection (GET/POST/COOKIES/HEADERS/URI)
Algorithmes de découverte de failles	• Support des failles Web (XSS, SQLi, LFI/RFI, CSRF…) • Défaut de configuration
Reporting	• GUI • Formats de rapports

Crawling : à propos

- “Crawling is the term used to describe the action taken by a program as it browses from page to page on a website. The crawler will visit a starting page and parse the provided links, crawling to those pages.” (source : Web Application Security Consortium)

- Dans un mode « automatique », l’URL principale est renseignée à l’outil
  - > Découverte des pages présentes sur le site Web
  - > Découverte des points d’injection

- Le crawler doit
  - > Être capable d’interpréter des pages malformées
  - > Supporter les redirections (JavaScript, Meta Refresh, HTTP Redirect)
  - > Supporter les notions de session (cookies, formulaires authentification…)

Crawling : Web Input Vector Extractor Teaser

- “WIVET is a benchmarking project that aims to statistically analyze web link extractors. In general, web application vulnerability scanners fall into this category.” (source : WIVET)

Outil	Score WIVET
w3af (SVN 4658)	50%
Arachni 0.4.0.2	14%
Wapiti 2.2.1	NOK
Skipfish 2.03b	NOK
Wget 1.12	10%

- Selon [http://code.google.com/p/wivet/wiki/CurrentResults], les scanners commerciaux ont de bien meilleures notes (> 85%)

Crawling : puits de crawling

- Certaines parties peuvent être problématiques
  - > e.g. application de type calendrier (infini)

Outil	Puits de crawling
w3af (SVN 4658)	OK
Arachni 0.4.0.2	NOK, l'exception doit être explicitement renseignée
Wapiti 2.2.1	NOK, l'exception doit être explicitement renseignée
Skipfish 2.03b	NOK, mal crawlé
Wget 1.12	NOK, crawling infini

Crawling : conclusions

- Ne pas faire (aveuglement) confiance au crawler de l'outil
  - > Support JavaScript très limité
  - > Puits de crawling à renseigner manuellement
  - > Incapacité à simuler un parcours client
- Contrôler son mode de fonctionnement
  - > Lui signifier de ne pas scanner en dehors de l'hôte à scanner
  - > Éviter toute page de déconnexion (effacement cookies de session)

Brique d'injection : fonctionnalités

Outil	Injection GET	Injection POST	Injection URI	Injection HEADERS	Injection COOKIES
w3af (SVN 4658)	OK	OK	OK	NOK, partiel *	NOK, partiel*
Arachni 0.4.0.2	OK	OK	OK	OK	OK
Wapiti 2.2.1	OK	OK	OK	NOK	NOK
Skipfish 2.03b	OK	OK	OK	NOK, partiel **	NOK

(*) uniquement en CLI et très limité

(**) en dur

Algorithmes de découverte de failles : que faut-il découvrir ?

- Ne pas oublier : découverte ≠ exploitation !

Failles d'implémentation	Failles de configuration
Injections (SQL, LDAP…)	Messages d'erreur
XSS (Reflected, Stored, DOM-based)	Bannières verbeuses
File Include, File Disclosure	Configuration SSL/TLS
Command Execution	…
…	

XSS : à propos

- “The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.” (source : CWE-79)

- Failles présentes dans de nombreuses applications
  - > Peu d’assistance par les outils de développement (e.g. moteurs de template)

- Conséquences
  - > Tout ce qui est faisable via du JavaScript !

- Prévention
  - > « Output Encoding » en fonction du contexte (HTML, JS, CSS, URL…)

Reflected XSS : découverte

- La découverte des failles XSS doit être contextuelle
  - > Mais les outils ne font généralement pas de découverte contextuelle
- Les outils ont souvent une approche « brutale »
  - > Injection de charges XSS prédéfinies
  - > Recherche dans la page retour si la charge est retournée intacte
- **`arachni`** est le seul à faire de la découverte contextuelle
- Failles qui devraient être faciles à découvrir automatiquement
  - > Peu de faux négatifs / faux positifs

Stored XSS : découverte

- XSS dont la source est issue d'un stockage (e.g. base de données)

- Pénible à tester en boite noire
  - > Injection d'une charge XSS
  - > Parcours des pages « connues » à la recherche de la charge XSS
    - – Peut être très long
    - – Inefficace sur les failles issues de stockage commun à plusieurs services

- Tout comme les Reflected XSS : dangereux à tester en boite noire !

- Nécessité de connaître le service audité avant de scanner

DOM-based XSS : découverte

- XSS côté client : la requête n'arrive pas au serveur

- Plusieurs techniques de découverte des DOM-based XSS
  - > Analyse de code source statique
  - > Analyse dynamique via moteur JavaScript

- **`w3af`** a un module de découverte des DOM-based XSS
  - > Recherche des points d'entrée (sources)
  - > Recherche des points de sortie (sinks)
  - > Identification dans le code source de la présence des ces sources et sinks

- Technique très faillible : sujette à des faux positifs et faux négatifs

SQL Injection : à propos

- “The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.” (source : CWE-89)

- Conséquences lourdes
  - > Récupération et/ou altération de données, compromission système

- Prévention
  - > Validation des entrées non de confiance **ET**
  - > Protection lors de l’utilisation des entrées non de confiance
    - – Échappement de caractères spéciaux
    - – Utilisation de requêtes préparées avec « bind variables »

In-band SQL Injection : découverte

- « Error-based » SQL injection
  - > Nécessite d'avoir des retours d'erreur : ne devrait plus exister !
- « UNION-based » SQL injection
  - > Les résultats sont affichés dans la page retour

Blind SQL Injection : découverte

- « Time-based » Blind SQL injection
  - > Injection d'une condition avec une commande SQL impactant le temps de réponse et comparaison du temps de réponse de la page retour entre deux conditions vrais/fausses
- « Diff-based » Blind SQL injection
  - > Injection d'une condition et comparaison de la page retour entre deux conditions vrais/fausses
- L'exploitation est bien plus longue que la découverte !

Blind SQL Injection : découverte

- Méthodologie
  - > Identifier si la page est stable en temps de réponse ou en contenu
  - > Injecter un vecteur d'injection SQL parmi un ensemble de vecteurs prédéfinis
  - > Détection d'un temps de réponse élevé (time-based) ou de différences de contenu dans la page retournée (diff-based)

Blind SQL Injection : découverte

- Méthode diff-based très sujette aux faux positifs !
- Méthode time-based plus efficace mais très lente !

- Axes d'amélioration des outils
  - > Enrichir les vecteurs possibles : réduction des faux négatifs
  - > Adapter les vecteurs à la cible (MySQL/MSSQL/Oracle) : meilleure rapidité

- Les faux négatifs sont liés à l'incapacité de l'outil à
  - > Construire une requête valide (time-based et diff-based)
  - > Identifier les changements dans la page retournée (diff-based)

Classement (benchmark)

	w3af (full)	arachni (default)	arachni (full)	wapiti (default)	skipfish (default)
Temps	3 minutes	90 secs	4 minutes	2 minutes	10 secs
Nb requêtes	3300	14000	75000	1400	5500
Vecteurs	2/5	3/5	4/5	3/5	2/5
CSRF	1/1	1/1	1/1	0/1	1/1
XSS	1/3	2/3	2/3	1/3	2/3
LFI	1/1	1/1	1/1	1/1	1/1
SQLi	7/17	16/17	16/17	16/17	2/17
Redirections	3/3	2/3	2/3	1/3	0/3
Note globale	15/30	25/30	26/30	22/30	7/30

Faux négatifs et faux positifs

Faux négatifs	Faux positifs
Faux sentiment de sécurité	Mauvais ressenti
Manques du crawler, brique d'injection et algorithmes de découverte de failles	Problème… surtout si non « acquittables » entre plusieurs scans

- La complémentarité des outils n'est pas évidente

Problématiques opérationnelles

- Auditer des plates-formes en production : oui, mais…
  - > Base de données en écriture
  - > Configurer finement l'outil
  - > Déclenchera les mécanismes de protection (les attaques sont issues du scanner)
  - > Perturbera les métriques (audience, charge réseau/système…)
  - > Communiquer vers les équipes de supervision

- En pratique, délicat à mettre en œuvre
  - > Plus réaliste sur les plates-formes de qualification (intégration continue)

Limitations des « Web Scanners » : résumé

Limites intrinsèques	Limites techniques	Limites opérationnelles (si tests en production)
Dédiés à la découverte de failles d'implémentation ou de configuration	Sites difficiles à crawler	Impacts potentiels sur le fonctionnement de l'application
Ne se focalisent que sur ce qui est visible de leur part (boite noire)	Technologies particulières (JSON, Flash, Action Message Format…)	Impacts sur les métriques : communication à réaliser
	Temps de scan pouvant exploser	Impacts sur les mécanismes de sécurité

Recommandations (pour une meilleure efficacité)

- Assister la partie crawling
  - > Décrire les points d'entrée « à la main »
  - > Exclure les pages de « logout »
  - > Exclure les cookies de session/authentification
  - > Identifier les puits de crawling ou mettre un verrou par défaut
- Faire un scan « léger » en première intention pour réduire le temps de scan

Conclusions

- Utilisation des « Web Scanners » délicate sur les architectures en production
- À considérer comme de l'assistance à l'auditeur (outillage)
- Si le site est « simple » à crawler alors
  - > **`arachni`** est très efficace sur les failles d'implémentation (XSS, SQLi)
  - > **`w3af`** apporte la meilleure exhaustivité de tests
- Si le site est « complexe » à crawler alors
  - > À la main et s'aider de proxies intrusifs (e.g. Burp)
- SDLC : cette démarche peut faire partie des « tests de régression »

Liens

- Web Application Security Scanner Evaluation Criteria
  - > http://projects.webappsec.org
- Louis Nyffenegger & Luke Jahnke – Harder, Better, Faster, Stronger…
  - > http://www.ruxcon.org.au/assets/Presentations/2011-2/LNLJ-Harder_Better_Faster_Stronger_V1.0.pdf
- The Scanning Legion: Web Application Scanners Accuracy Assessment…
  - > http://sectooladdict.blogspot.com/2011/08/commercial-web-application-scanner.html
- Why Johnny can't pentest?
  - > http://cs.ucsb.edu/~adoupe/static/black-box-scanners-dimva2010.pdf

Annexe : Application « benchmark »

- Objectif
  - > Valider l'efficacité des outils sur des failles simples et classiques
  - > Uniquement vis-à-vis des faux négatifs
  - > Pas de piège particulier : toutes les failles devraient être découvertes
- Partie « Vecteurs » : identifier les outils ayant toutes les fonctions d'injection
- Partie « Failles » : accessibles toutes en GET, donc évaluation de la qualité des algorithmes de découverte des failles (et leur configuration par défaut)
- Attention…
  - > Toute application de benchmark peut être biaisée pour favoriser un outil

Annexe : Classement (briques)

- Attention… Reflète mon point de vue personnel

	Crawling	Injection	Découverte	Reporting	Exhaustivité
w3af (SVN 4658)	1er	2ème	2ème	3ème	1er
arachni 0.4.0.2	2ème	1er	1er	1er ex.	2ème
wapiti 2.2.1	3ème	3ème ex.	3ème	4ème	3ème ex.
skipfish 2.03b	4ème	3ème ex.	4ème	1er ex.	3ème ex.