La sécurité peut aussi "Shift Left"

Xavier "Pamplemousse" Maso

2025-03-25

—

Saviez-vous qu'à l'âge de pierre du développement logiciel, nos ancêtres développeureuses avaient pour coutume de tester leur code directement en production ?

Ça peut nous paraitre fou car, de nos jours, la gestion de la qualité au cours développement fait partie de l'état de l'art ; Par exemple, il est devenu "normal" d'écrire des tests automatisés. On peut même aller plus loin, et dire qu'on est d'accord sur le fait que quelqu'un qui ne teste pas n'est pas un.e **bonne** développeureuse.

Et je suis convaincu qu'on ne devrait pas s'arrêter là, nous pouvons faire mieux.

🍊 Xavier Maso

🛠️ ~~Ingénieur en Sécurité des Applications~~, ~~à Oracle NetSuite~~

🛠️ Bientôt Consultant en Développement Logiciel, chez PALO IT

Les points de vue et opinions exprimé.e.s dans cette présentation sont les mien.ne.s et ne reflètent pas celleux de mon employeur (précédent ou futur).

💻 git.sr.ht/~pamplemousse[1]

🫂 mamot.fr/@Pamplemouss_

🏠 www.xaviermaso.com

[1] Vous y trouverez d'ailleurs les slides, et l'environnement utilisé pour les démos : git.sr.ht/~pamplemousse/security_can_also_shift_left.

Présentation

Objectif : **Montrer concrètement qu'il est possible de faire de la sécurité en cours de développement.**

- Montrer aux **QAs** qu'on peut tester avant la fin du développement ;
- Montrer aux **développeur.euse.s** qu'iels peuvent participer à tester ;
- Montrer aux **ingénieur.e.s en sécurité** que certaines validations peuvent être déléguées.

Introduction

—

Chacun.e.s à un rôle à jouer.

Petit sondage démographique de la salle : qui a pour mission principale de faire

- du test ?
- du développement ?
- de la sécurité ?

Hypothèses acceptées :
- On veut “shift left” ;
- La sécurité c’est important.

Pré-requis :
- À peu près distinguer sa droite de sa gauche.

Introduction
—

Cette présentation ne couvrira pas en détails pourquoi on veut “shift left”, ni pourquoi faire attention à la sécurité.
Disons que :
- notre intuition nous dit qu’il vaut mieux prévenir que guérir,
- et que nos fondements moraux (ou un cadre légal pour celleux qui en seraient dénué) nous poussent à respecter nos utilisateurices et leurs données.

Hypothèses acceptées :
- On veut “shift left” ;
- La sécurité c’est important.

Pré-requis :
- À peu près distinguer sa droite de sa gauche.

Introduction
—

Cette présentation ne couvrira pas en détails pourquoi on veut “shift left”, ni pourquoi faire attention à la sécurité.
Disons que :
- notre intuition nous dit qu’il vaut mieux prévenir que guérir,
- et que nos fondements moraux (ou un cadre légal pour celleux qui en seraient dénué) nous poussent à respecter nos utilisateurices et leurs données.

 ~~Avoir tout fini, puis attendre le résultat d'un audit d'expert pour apprendre qu'on doit (presque) tout revoir.~~

 Du feedback sur la sécurité de notre production pour agir sur ce qui pose problème au plus tôt, avant que tout parte en cacahuète.

Introduction

—

Définition “shift left” : tester, évaluer la qualité et la performance au plus tôt dans le processus de développement.

Table des matières

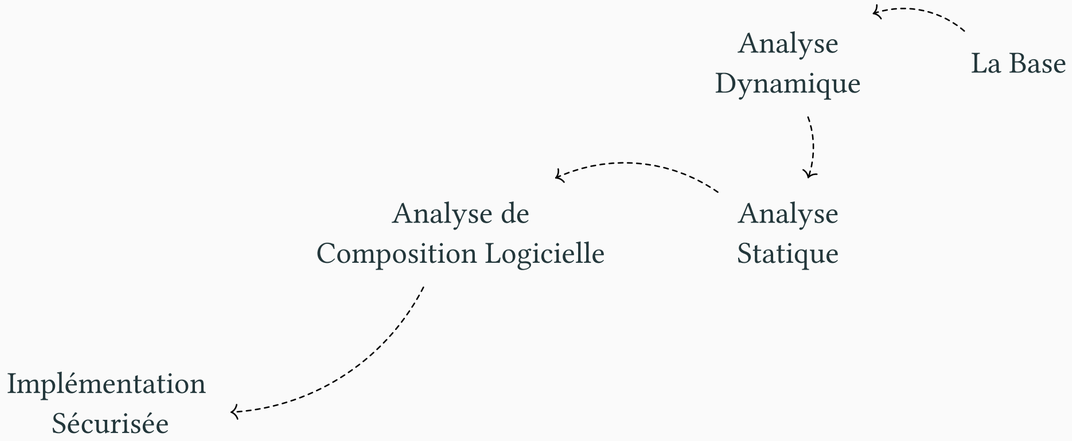
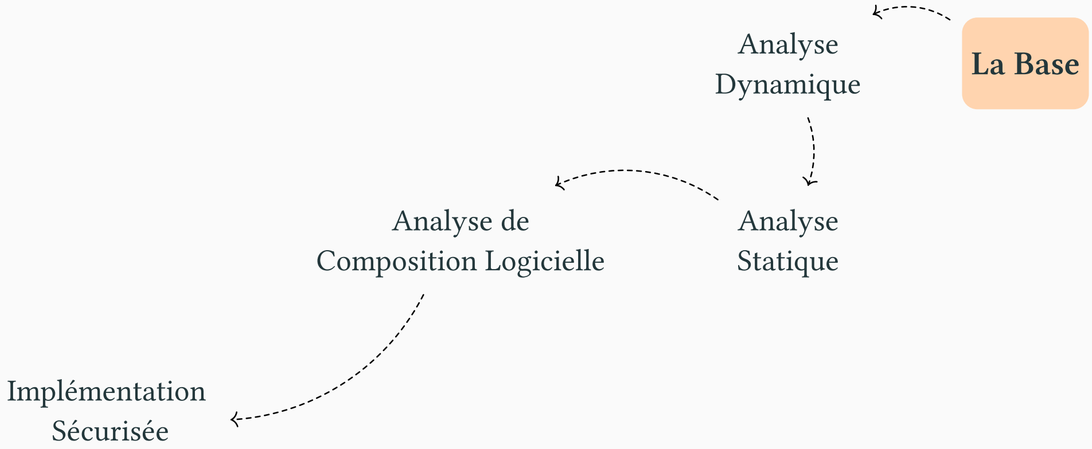


Table des matières

Table des matières

Table des matières

1 La Base


La Base
Analyse Dynamique
Analyse Statique
Analyse de Composition Logicielle
Implémentation Sécurisée

1 La Base

- Confidentialité (Confidentiality)
- Intégrité (Integrity)
- Disponibilité (Availability)

... des données et systèmes.

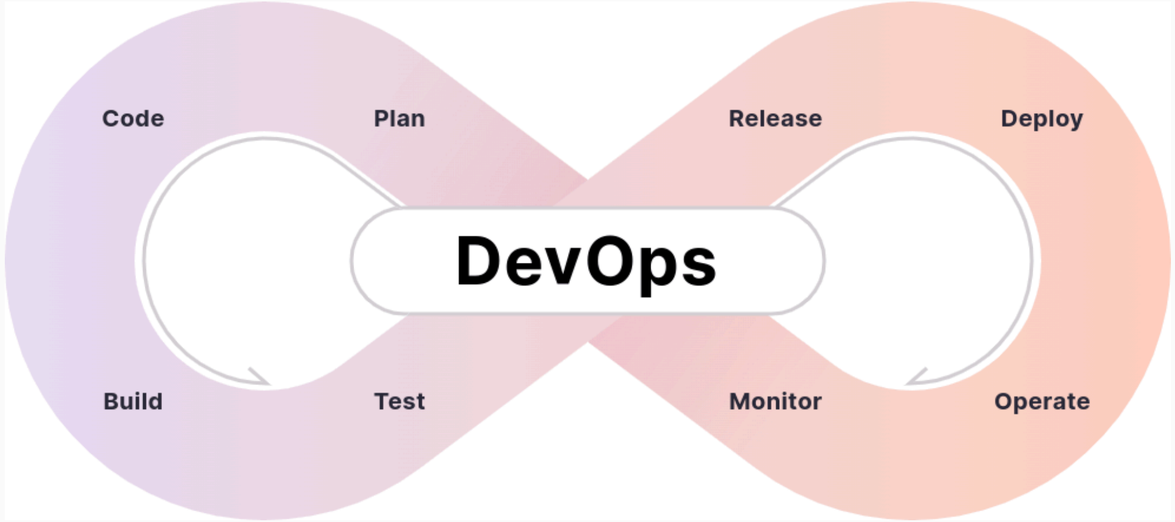


Figure 1: Boucle DevOps “classique”.

Approche “moderne” pour penser et organiser le processus de développement logiciel.

Objectif : accélérer la livraison, améliorer la qualité.
Décloisonner les différentes étapes, et raccourcir les boucles de feedback, pour éviter l’effet tunnel.

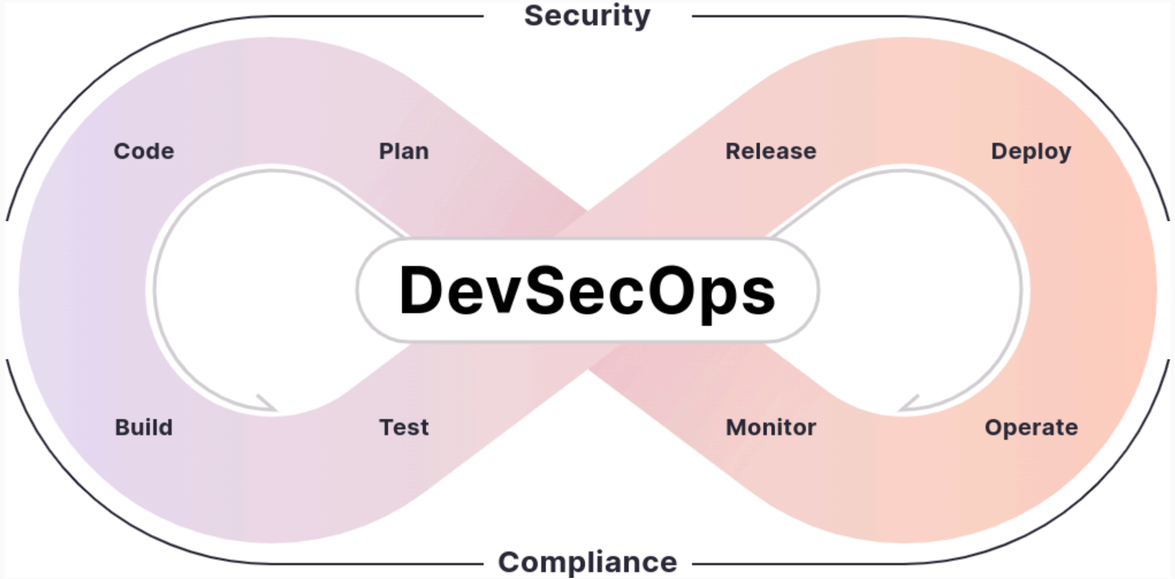


Figure 2: Boucle DevOps, avec un enrobage de sécurité.

Je suis plus familier avec ce qui se trouve sur la gauche (d'où le sujet de la session).

Apporter des pratiques de sécurité sur chacuns des points de la boucle.

- test: analyses statique et dynamique
- build: chaine d'approvisionnement, analyse de composition
- code: revue de code, principes de programmation sécurisée
- plan: modèle de menaces

vulnérabilité, faille Insuffisances, imperfections d'un système, donnant prise à des attaques (exploits).

Qui est familier avec les injections SQL?

vulnérabilité, faille Insuffisances, imperfections d'un système, donnant prise à des attaques (exploits).

e.g. L'exécution d'une requête SQL qui a été composée par la concaténation de variables contenant des entrées utilisateurices non vérifiées.

```
user_id = request.args.get('id')
requete = "SELECT name FROM users WHERE id = '{}';".format(user_id)
connection.execute(requete)
```


Qui est familier avec les injections SQL?

exploit Données et étapes permettant l'exploitation d'une vulnérabilité d'un système, menant à l'obtention des données sensibles qu'il contient, où à l'abus de ses fonctionnalités.

exploit Données et étapes permettant l'exploitation d'une vulnérabilité d'un système, menant à l'obtention des données sensibles qu'il contient, où à l'abus de ses fonctionnalités.

e.g. Une entrée utilisateurice, sous forme de requête HTTP, ajoutant du code SQL arbitraire à exécuter (SQL injection).

```
curl "http://my.app?id=42' OR 1=1; --"
```

```
user_id = request.args.get('id')
requete = "SELECT name FROM users WHERE id = '{}';".format(user_id)
connection.execute(requete)
```

exploit Données et étapes permettant l'exploitation d'une vulnérabilité d'un système, menant à l'obtention des données sensibles qu'il contient, où à l'abus de ses fonctionnalités.

e.g. Une entrée utilisateurice, sous forme de requête HTTP, ajoutant du code SQL arbitraire à exécuter (SQL injection).

```
curl "http://my.app?id=42' OR 1=1; --"
```

```
user_id = "42' OR 1=1; --"
requete = "SELECT name FROM users WHERE id = '{}';".format(user_id)
connection.execute(requete)
```

exploit Données et étapes permettant l'exploitation d'une vulnérabilité d'un système, menant à l'obtention des données sensibles qu'il contient, où à l'abus de ses fonctionnalités.

e.g. Une entrée utilisateurice, sous forme de requête HTTP, ajoutant du code SQL arbitraire à exécuter (SQL injection).

```
curl "http://my.app?id=42' OR 1=1; --"
```

```
user_id = "42' OR 1=1; --"
requete = "SELECT name FROM users WHERE id = '42' OR 1=1; --';"
connection.execute(requete)
```

faiblesse Insuffisances, imperfections d'un système, qui sous certaines circonstances, peuvent laisser le système vulnérable.

C'est le "bout de faille" à corriger.

faiblesse Insuffisances, imperfections d'un système, qui sous certaines circonstances, peuvent laisser le système vulnérable.

e.g. L'exécution d'une requête SQL qui a été composée par la concaténation ou interpolation de variables.

```
requete = "SELECT name FROM users WHERE id = '{}';".format(user_id)
```


C'est le "bout de faille" à corriger.

0day Une vulnérabilité qui n'est pas encore connue, ni du public, ni des producteurs (de matériels ou de logiciels) : cela fait "0 jours" qu'un palliatif ou correctif est développé.

Common Vulnerabilities and Exposures (CVE)

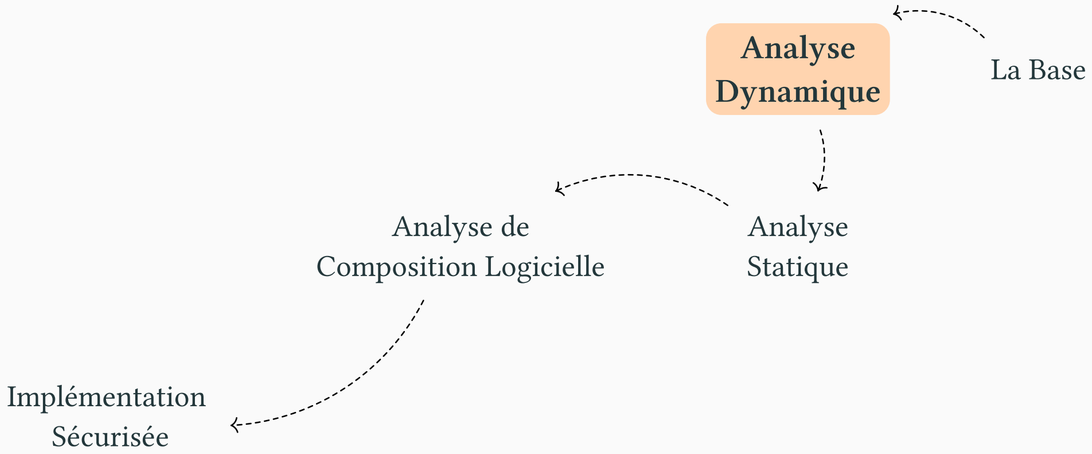
1. Une organisation gestionnaire d'un catalogue public de vulnérabilitées logicielles connues.
2. Un identifiant (numéro de CVE) pour une vulnérabilité reconnue.

CVE Numbering Authority (CNA) Les organisations autorisées à assigner des numéros de CVE concernant les programmes de leur périmètre.

Common Vulnerability Scoring System (CVSS) Un score numérique représentant la sévérité d'une vulnérabilité.

Des sites tous plus moches les uns que les autres.

2 Analyse Dynamique (Dynamic Application Security Testing - DAST)



2 Analyse Dynamique (Dynamic Application Security Testing - DAST)

2.1.1 Définition

> Roughly 63% of applications have flaws in first-party code
> [...]
> 41% of first-party [...] flaws persist beyond the one year mark to become “security debt.”
>
> — State of Software Security, 2024 [1]

- Analyzer des programmes en les exécutant.
- Deux grandes approches :
  - Boite noire - sans aucune connaissance du fonctionnement interne ;
  - Boite blanche - avec connaissance du fonctionnement interne ;
    - Voire de façon intéractive (Interactive Application Security Testing - IAST).

IAST - en modifiant légèrement (le moins disruptif possible) le programme pour obtenir des informations nécessaires aux tests. Par exemple, chemins d’exécutions empruntés (couverture de code), conditions et branches, etc.

2.1.2 Techniques

2.1.2.1 Tests manuels

Utiliser l'artéfact observé, en exerçant ses fonctionnalités.

Par exemple, pour tester un site web ou une application mobile : naviguer sur différentes pages, rentrer des valeurs dans les formulaires, cliquer sur des boutons, etc.

+ Simple à mettre en place.
− Fastidieux, lent, peu reproductible, très localisé.

Pas FIRST (Fast, Isolated/Independent, Repeatable, Self-validating, Thorough).

Pas ou peu reproductible ⇒ on investit à chaque fois la même quantité de temps.

2.1.2.2 Tests automatisés

Faire exercer des fonctionnalités précises de l'artéfact observé, avec des données concrètes fixées a priori, par un agent automatique.

Par exemple : Écrire et exécuter des tests unitaires, d'intégration, bout en bouts[1].

+ Rapide à exécuter, reproductible, plus grande couverture en fonction du temps.
– Plus coûteux à mettre en place, maintenance.

[1] Par exemple en utilisant ZAP [2] pour des applications web.

Plus reproductible ⇒ on peut capitaliser sur l'investissement, et augmenter le retour incrémentalement.

2.1.2.3 Fuzzing, Tests basés sur les propriétés (PBT)

Envoyer des entrées générées pseudo-aléatoirement, et surveiller comment l'artéfact réagit.

Exemples : `AFL` [3], `QuickCheck` [4], `ZAP` [2].

+ Grande couverture en fonction du temps.
− Processus global non déterministes, coûteux à exécuter, plus coûteux à mettre en place.

Les entrées "intéressantes" (qui explore de nouvelles portions du programme testé, qui exerce une vulnérabilité, etc.) sont conservées.

Fuzzing :

- Génération aléatoire peut être guidée selon des heuristiques (par exemple en mutant des entrées qui ont découverts de nouvelles portions de code), pour encourager la manifestation de bugs.
- Est-ce que le programme crashe ?

Tests basés sur les propriétés

- Les entrées sont fabriquées aléatoirement par un générateur qui va respecter des propriétés données (*e.g.* des entiers, des chaines de charactères de taille N et contenant seulement des éléments d'un alphabet

2.1.2.4 Autres

Avant-gardiste, immature, et/ou manque d'intégration à l'outillage usuel.

- Exécution concolique ;
- ~~IA~~ LLMs, ou autres modèles construits par apprentissage profond.

2.2.1 Présentation

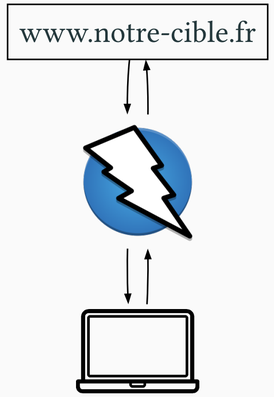
- Scanneur d'application web : proxy permettant d'analyser du traffic ;
- Gratuit et open-source ;
- Anciennement OWASP, maintenant Checkmarx[1].

— https://www.zaproxy.org/ [2]

[1] Il semblerait sous un modèle "open-core", *c.f.* https://www.zaproxy.org/blog/2024-09-24-zap-has-joined-forces-with-checkmarx/

2.2.2 Fonctionnement

- Interception de traffic HTTP, WebSocket ;
- Différents “scans”, actifs et passifs, soulevant des “alertes” ;
- Interface graphique, ou non ;
- Spidering (*a.k.a.* crawling) ;
- Plugins (*e.g.* encodeurs/decodeurs, scanneurs de vulnérabilités, fuzzers, HUD[1], etc.)[2] ;
- Et autres fonctionnalités : points d’arrêts, édition et renvoi de requêtes, etc.



[1] Interface directement intégrée dans le navigateur, *c.f.* https://www.zaproxy.org/docs/desktop/addons/hud/.
[2] *C.f.* https://www.zaproxy.org/addons/ pour la liste des plugins installables.

Spidering pour explorer la surface d’attaque exposée par l’application.

Heads Up Display - HUD (“affichage tête haute” en français) : permet de superposer les informations des instruments de bord à la vision de l’environnement (pensez aux pilotes d’avions de chasse).
L’idée est la même dans ce plugin : injecter une interface ZAP directement dans l’application testée.

2.2.3 Exemples d'utilisation

2.2.3.1 Dans un environnement de développement

Faire passer tout son traffic HTTP dans ZAP

- Obtenir des informations "en direct" sur la sécurité ;
- Outil de débogage : enregistrer, rejouer, modifier des requêtes[1].

2.2.3.2 Dans un environnement d'intégration continue

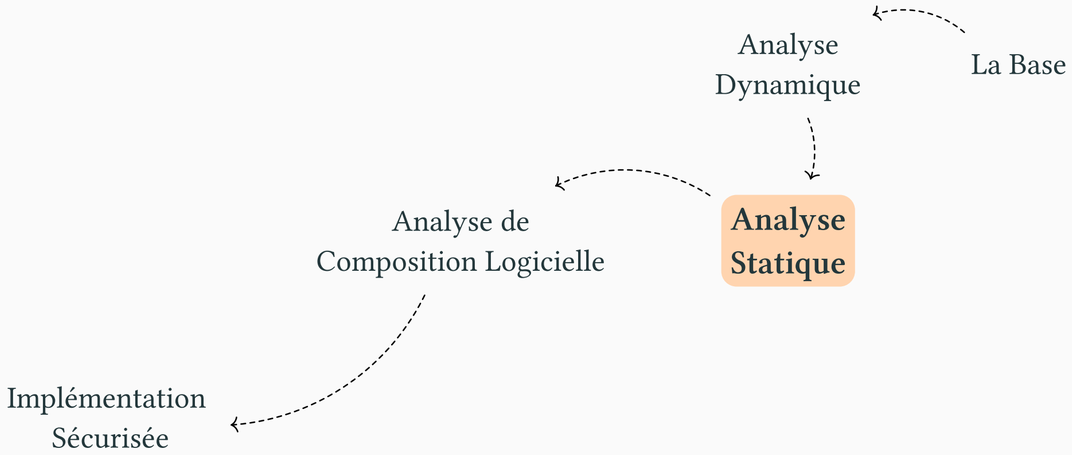
- Tests automatisés exécutés quand des "endpoints" HTTP sont ajoutés, ou modifiés.
  Lancer des requêtes, et vérifier les alertes.
- Intégrer à des tests de bout en bout existants.
  Pour scanner tout le traffic généré par les tests, et vérifier les alertes.
- Programmé régulièrement :
  Utiliser les capacités d'exploration, et tester les "endpoints" découverts.

[1] À la MITM, Charles, Postman, etc.

2.2.4 Démo

On utilise ZAP pour obtenir des informations sur la sécurité de notre cible : l'application OWASP Juice Shop.

3 Analyse Statique (Static Application Security Testing - SAST)



3 Analyse Statique (Static Application Security Testing - SAST)
—

> Roughly 63% of applications have flaws in first-party code
> [...]
> 41% of first-party [...] flaws persist beyond the one year mark to become “security debt.”
>
> — State of Software Security, 2024 [1]

- Analyzer des programmes sans les exécuter ;
- Raisonner sur des représentations du programme pour déduire des propriétés ;
- Exemples d’outils SAST : SemGrep, CodeQL, SonarQube, Fortify, Checkmarx, Trivy, ...

Des propriétés qui nous permettent de dire si des vulnérabilités sont présentes.
Par exemple : est-ce que des requêtes SQL sont composées à partir de des données utilisateurices non vérifiées ?

3.2.1 Examples de représentations utiles

- Arbre de la Syntaxe Abstraite
- Arbre Syntaxique
- Graphe de Flot de Contrôle
- Graphe d'Appels de Fonctions

Sources : [5], [6].

3.3.1 Présentation

- **Sem**antic **Grep** ;
- Moteur d'analyse statique, pour chercher des fragments de code selon des règles ;
- Gratuit et open-source[1] ;
- Supporte de nombreux languages[2] ;
- Fonctionne sur du code "partiel" ;
- Ensemble de règles maintenues par la communauté[3].

> — https://semgrep.dev/products/semgrep-code [7]

[1] Modèle open-core.
[2] *C.f.* https://semgrep.dev/docs/supported-languages.
[3] *C.f.* https://github.com/semgrep/semgrep-rules.

Grep c'est de la recherche textuelle brute, ou avec expressions régulières. On peut pas matcher des éléments de grammaires non contextuelles, ça n'a pas de "compréhension" des structures du code.

Recherche de "morceau(x)" d'AST qui correspondent au(x) pattern(s) spécifiés par l'utilisateurice.

L'entreprise SemGrep propose plusieurs produits, mais on s'intéresse ici à l'outil d'analyse statique dont le moteur est open-source (y'a une version PRO avec des fonctionnalités plus avancées - notamment d'analyses de flux de données).

3.3.2 Règle SemGrep

- Un ensemble de clé-valeurs `YAML`[1] ;
- Représente
  - le(s) pattern(s) de code à chercher,
  - et des méta-informations (language, message à afficher, sévérité, et autres.)[2].

3.3.3 Exemples d'utilisation

- ▸ Scan unique ;
  ▸ Dans un environnement d'intégration continue ;
- ▸ Utiliser les règles communautaires ;
  ▸ Écrire ses propres règles.

[1] 🤢🤢🤢 https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell

[2] *C.f.* https://semgrep.dev/docs/writing-rules/overview pour la syntaxe des règles et des patterns.

`YAML`, c'est affreux :

- Pas modulaire :
  - pas d'import d'autre fichier `YAML` dans le standard,
  - pas de variables, pas de fonctions ;
- En pratique, avec de gros fichiers, la sensibilité à l'indentation devient une vraie galère.

Les gens qui gèrent un grand ensemble de règles peuvent utiliser une représentation plus haut niveau, et "compilent" leur règles vers du `YAML`.

3.3.4 Démo

```
semgrep scan --json \
  --config "${DEMO_EXAMPLES}/rules.yaml" \
  "${DEMO_JUICE_SHOP_SOURCES}" > result.semgrep.json
< result.semgrep.json jq '.results'
```


On cherche des injections SQL dans l'application web OWASP Juice Shop. Lire les sources, c'est tricher (*c.f.* https://pwning.owasp-juice.shop/companion-guide/latest/part1/rules.html#_source_code), mais je fais ce que je veux parce que c'est moi qui présente.

3.3.5 Démo

```
yq --slurp --yaml-output 'map(.rules) | flatten | { rules: . }' \
  $(find "${DEMO_SEMGREP_RULES}" -wholename '*script/*.yaml') \
  > rules.javascript.yaml
semgrep scan --json \
  --config rules.javascript.yaml \
  "${DEMO_JUICE_SHOP_SOURCES}" > result.community.semgrep.json
< resul.communityt.semgrep.json \
  jq '.results | map(select(.check_id == "eval-detected") | .path)'
```


On peut aussi utiliser des règles existantes.
Lire les sources, c'est encore tricher (*c.f.* https://pwning.owasp-juice.shop/companion-guide/latest/part1/rules.html#_source_code), mais je fais ce que je veux parce que c'est moi qui présente.

3.4.1 Présentation

- Chaîne d'outils d'analyse statique[1] ;
- Gratuit pour scanner du code open-source[2] ;
- Collecte des faits à propos du code à la compilation, dans une base de données ;
- Un language de requêtes pour extraire des informations de cette base de données ;
- Plusieurs languages supportés[3] ;
- Ensemble de librairies, et requêtes maintenues par la communauté[4].

— https://codeql.github.com [8]

[1] Initié par Semmle, racheté par GitHub, *c.f.* https://github.blog/news-insights/company-news/github-welcomes-semmle/.

[2] *C.f.* https://github.com/github/codeql-cli-binaries/blob/main/LICENSE.md.

[3] *C.f.* https://codeql.github.com/docs/codeql-language-guides/.

[4] *C.f.* https://github.com/github/codeql.

Code Query Language, basé sur DataLog.

Informations à extraire sont typiquement où sont les faiblesses voire les vulnérabilités.

Capture de multiples représentations utiles dans sa base de données.
Analyse de flux de données.

3.4.2 Comment ça fonctionne ?

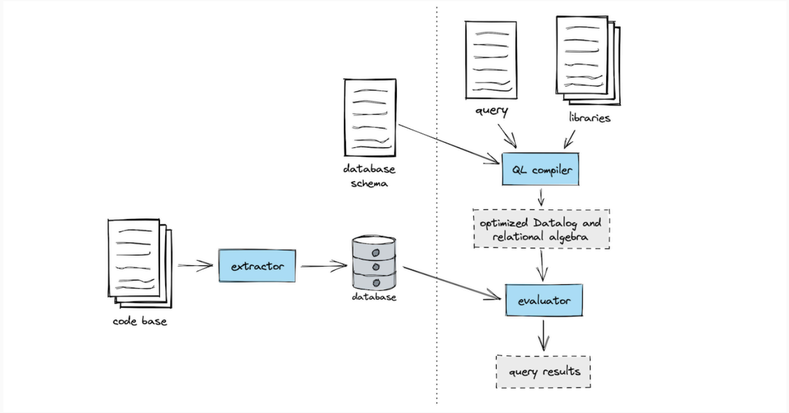


Figure 3: Schéma de fonctionnement global de CodeQL[1].

[1] *C.f.* https://github.blog/security/web-application-security/code-scanning-and-ruby-turning-source-code-into-a-queryable-database/.

Quand je disais que plusieurs languages étaient “supportés” ça voulait dire que :

- y’a un extracteur pour lire des sources écrites dans de tels languages et y collecter des données ;
- y’a une librairie de requêtes pour manipuler les représentations de code spécifiques à ces languages.

Les base de données d’informations collectées par language sont assez “bas niveau” (arbre de syntaxe abstraite, etc.).
Pour chercher des patterns un peu abstraits dans le code, il faut construire des requêtes “de base” sur lesquelles s’appuyer.

3.4.3 Exemples d'utilisation

- ▸ Scan unique ;
  - ▸ Dans un environnement d'intégration continue ;
- ▸ Utiliser les requêtes communautaires ;
  - ▸ Écrire ses propres requêtes, en bénéficiant des librairies communautaires.

3.4.4 Démo

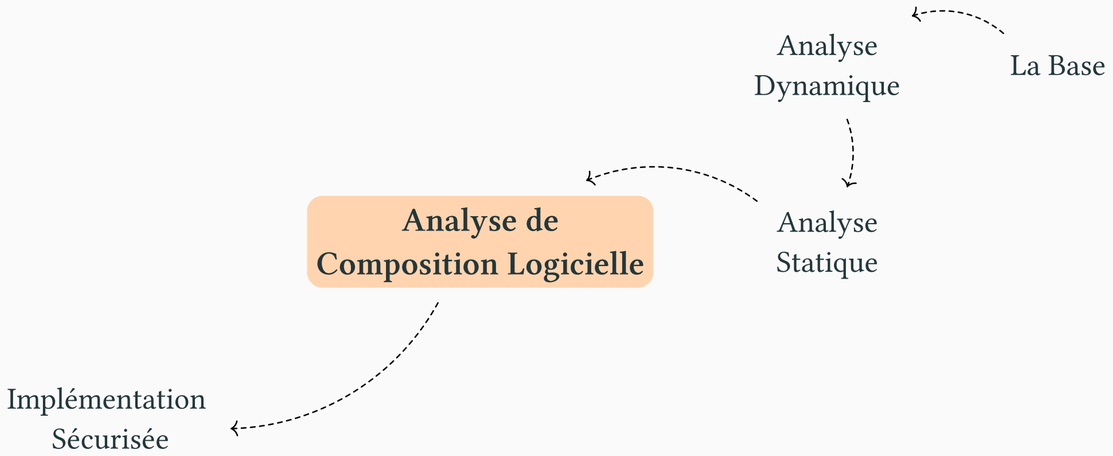
```
codeql database create ./my.db \
  --language typescript \
  --source-root "${DEMO_JUICE_SHOP_SOURCES}"
codeql pack install "${DEMO_EXAMPLES}/codeql-pack"
codeql query run --database ./my.db --output result.bqrs \
  "${DEMO_EXAMPLES}/codeql-pack/query.ql"
codeql bqrs decode result.bqrs --format json > result.codeql.json
< result.codeql.json jq '.["#select"].tuples | map(.[0].label)'
```


Local dataflow.

La marche est clairement plus haute qu'avec SemGrep, mais la puissance de l'outil est bien plus grande :

- en particulier grâce au QL
  - permet d'exprimer des recherches beaucoup plus complexes
  - est plus modulaire

4 Analyse de Composition Logicielle (Software Composition Analysis - SCA)



4 Analyse de Composition Logicielle (Software Composition Analysis - SCA)

4.1.1 Sur le papier

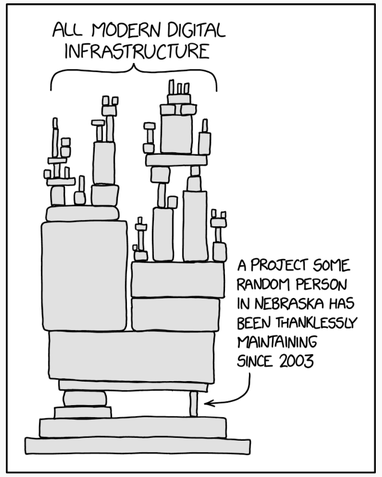


Figure 4: “Dependency” de xkcd, https://xkcd.com/2347/

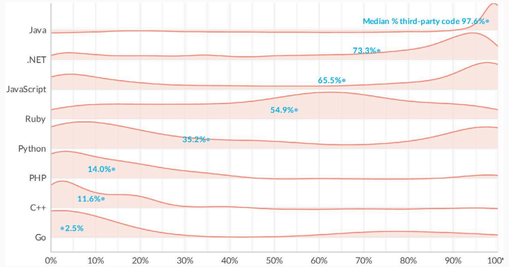
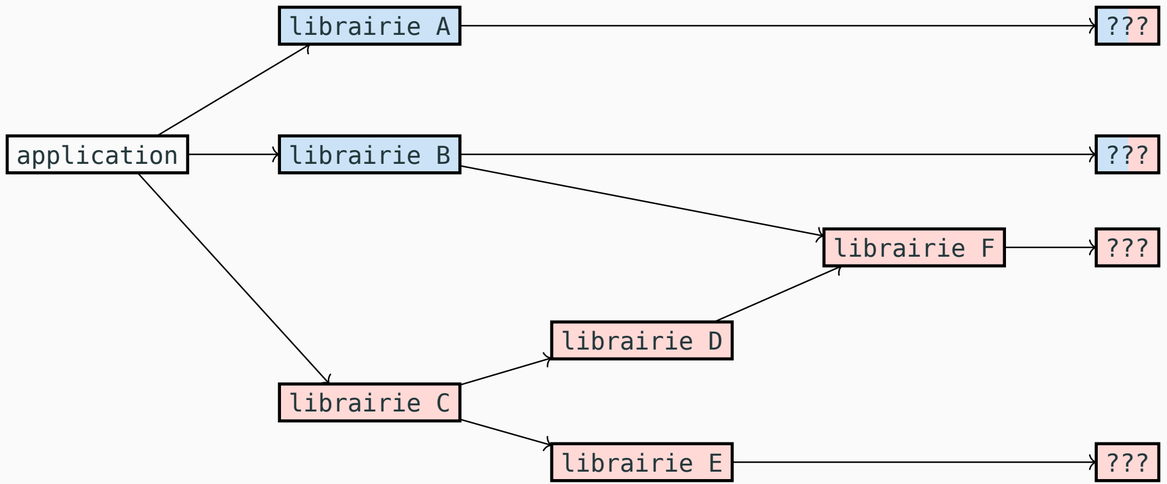


Figure 5: Percentage of third-party code in applications using third-party code in most recent scan

— State of Software Security, 2024 [1]



Imaginons que l'on développe une application.

Elle peut avoir des librairies qui :

- sont développées en interne
- sont développées par des tiers
- sont elles-même dépendantes d'autres librairies
- ont des dépendances en commun
- etc.

4.1.2 Examples

```
nix-store --query --graph $(readlink $(which jq)) \
  | nix shell nixpkgs#graphviz --command dot -Tsvg
```

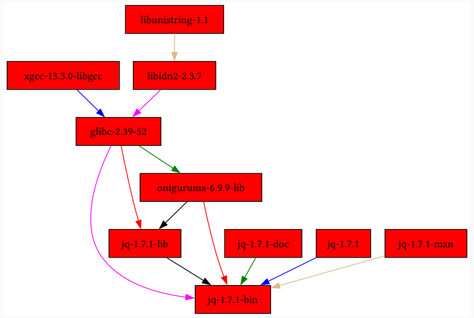


Figure 6: Clôture transitive des dépendances du programme `jq`.

Le gestionnaire de paquets `nix` force chaque paquet à déclarer ses dépendances. Comme chaque dépendance est elle-même un paquet, elle doit déclarer ses dépendances, etc.

Donc, quand un paquet est installé par `nix`, on peut, par transitivité, requêter toute ses dépendances (clôture transitive) sous forme de graphe.

Là, on construit le graphe des dépendances du programme `jq`.

```
nix-store --query --graph $(readlink $(which nvim)) \
  | nix shell nixpkgs#graphviz --command dot -Tsvg
```

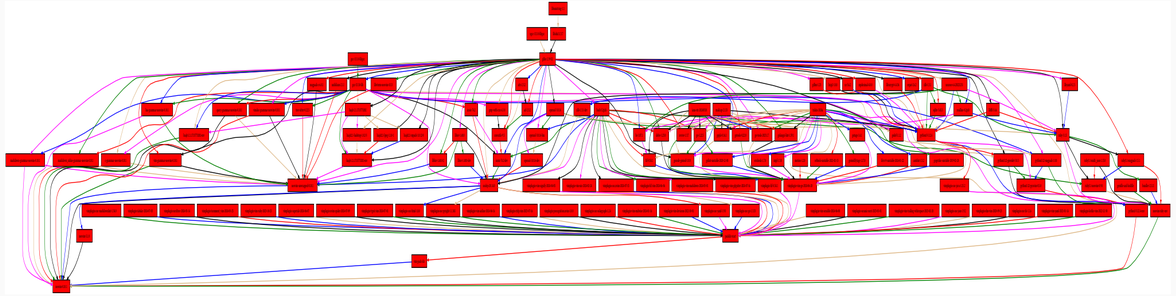


Figure 7: Clôture transitive des dépendances du programme neovim.

Là, on construit le graphe des dépendances du programme `neovim`.

```
nix-store --query --graph $(readlink /run/booted-system) \
  | nix shell nixpkgs#graphviz --command dot -Tsvg
```

Figure 8: Un plat de spaghettis.

`NixOS` est un système d'exploitation construit "par dessus" `nix`. Il part de l'idée qu'un OS est ensemble de programmes, et de configurations, et tout ça peut se gèrer avec `nix`. Donc on peut aussi requêter la clôture transitive des dépendances de tout un système.

Là, on construit le graphe des dépendances du système courrament démarré.

Le SVG fait 101000+ lignes…

4.1.3 Les risques

> [Roughly] 70% [of applications] contain flaws in third-party code
> […]
> 48% of third-party flaws persist beyond the one year mark to become “security debt.”
>
> — State of Software Security, 2024 [1]

Le code écrit par les autres, qu’on importe dans nos projets, participe à l’augmentation de notre dette de sécurité.

4.1 Présentation

- A-t-on un inventaire complet de nos dépendances logicielles ?
- Utilise-t-on des dépendances obsolètes ? Avec des faiblesses et/ou des vulnérabilités connues ?
- Utilise-t-on des dépendances avec des licenses incompatibles avec l'usage que l'on en fait ?

4.1 Présentation

La plupart des vulnérabilités dans les programmes et librairies ne concernent qu'un sous-ensemble de fonctionnalité(s) et d'API(s).

$$\Longrightarrow$$

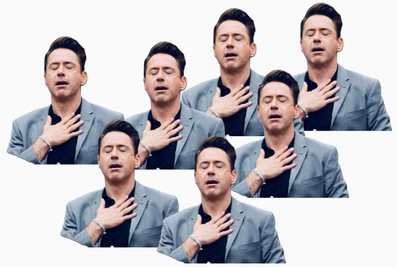
Utiliser une librairie qui a une vulnérabilité connue ne rend pas forcément vulnérable.

4.1 Présentation

La plupart des vulnérabilités dans les programmes et librairies ne concernent qu'un sous-ensemble de fonctionnalité(s) et d'API(s).

$$\Longrightarrow$$

Utiliser une librairie qui a une vulnérabilité connue ne rend pas forcément vulnérable.



Bref, c'est compliqué.

- Les graphes de dépendances sont (très) complexes ;
- Des vulnérabilités sont découvertes et publiées tous les jours ;
- Il peut y avoir des contraintes de compatibilité
  - au niveau de l'application et de ses dépendances
  - et entre les dépendances elle-mêmes ;

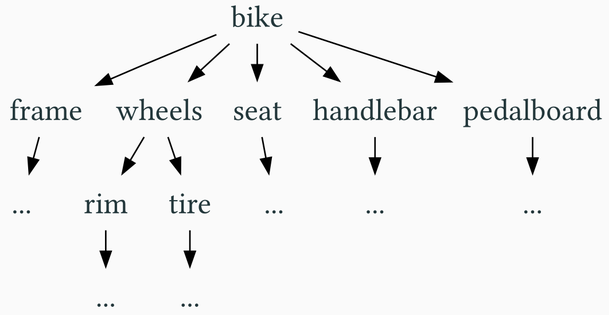
Automatiser, prioriser ce que l'on fixe, et garder un contrôle sur ce que l'on ne fixe pas.

Un graphe de dépendances sain hier peut ne plus l'être aujourd'hui.

4.2 Nomenclature Logicielle (Software Bill of Materials - SBoM)

Nomenclature Inventaire hierarchisé de tous les différent.e.s composants et matières premières constitutives d'un produit.
Représentée sous forme d'un arbre où la racine décrit le produit fini, et où chaque noeud symbolise un composant de son noeud parent.

e.g.



Nomenclature Logicielle (SBoM)[1] Nomenclature des composants logiciels impliqués dans le développement et la publication d'une application (*e.g.* bibliothèques, chaine de compilation, etc.).
Pour chaque composant listé dans le document : on peut notamment trouver les informations suivantes : version, license, auteur.ice.s, vulnérabilités connues, etc.

Deux spécifications majeures : SPDX (Linux Foundation) [9], CycloneDX (OWASP) [10].

[1] Il existe d'autres types de nomenclatures, telles que : Software-as-a-Service BoM, Hardware BoM, Machine Learning BoM, Cryptography BoM, etc.

4.2.1 CycloneDX

Figure 9: Modèle de nomenclature dans la spécification CycloneDX [10]

J'ai uniquement (et un peu) utilisé CycloneDX, donc je ne peux parler que de ce format…

Les entrées qui nous intéressent le plus dans ce modèle :

- Components : Ensemble des informations sur les composants (directs ou indirects, interne ou tiers) ;
- Dépendances : Relations entre les différents composants ;
- Vulnérabilités : Ensemble des failles trouvées dans les composants. On va détailler ça un peu.

4.2.2 Vulnerability Exploitability eXchange (VEX)

Pour communiquer des failles de sécurité s'appliquant à des composants.

- Au moins deux utilisations différentes :
  - Nomenclature intégrant des données au format VEX ;
  - Dans un document VEX indépendant ;
- Schéma des informations VEX dans CycloneDX [10] : github.com/CycloneDX/specification/blob/1.6/schema/bom-1.6.schema.json ;
- Exemples d'utilisation : github.com/CycloneDX/bom-examples.

```
< "${DEMO_CYCLONEDX_SPECIFICATION}/schema/bom-1.6.schema.json" \
  jq -r '.definitions.vulnerability.properties
    | to_entries[]
    | .key as $k | select(any((["advisories", "affects", "description", "id", "source",
"ratings" ] | index($k))))
    | [.key, .value.description]
    | @csv' > vex.schema.light.csv
```

id	The identifier that uniquely identifies the vulnerability.
source	The source that published the vulnerability.
ratings	List of vulnerability ratings
description	A description of the vulnerability as provided by the source.
advisories	Published advisories of the vulnerability if provided.
affects	The components or services that are affected by the vulnerability.
…	…

Voilà des exemples d'informations pouvant être communiquées par VEX pour chaque vulnérabilité.

- Ratings = notes, scores (*e.g.* CVSS)
- Advisories = notifications, annonces

4.3.1 Présentation

- Scan d'applications pour détecter des vulnérabilités connues dans les dépendances logicielles ;
- Analyses complémentaires pour valider l'applicabilité des vulnérabilités ;
- Différentes sources de vulnérabilités connues, notamment :
  - OSV : Base de données distribuée pour l'Open Source ;
  - National Vulnerability Database - NVD : Base de données maintenues par le NIST 🇺🇸;
  - Vulnérabilités rapportées dans les distributions Linux.

— https://owasp.org/www-project-dep-scan/

Librairies, images de conteneurs (logiciels, et configuration), etc.

OSV = Open Source Vulnerabilities

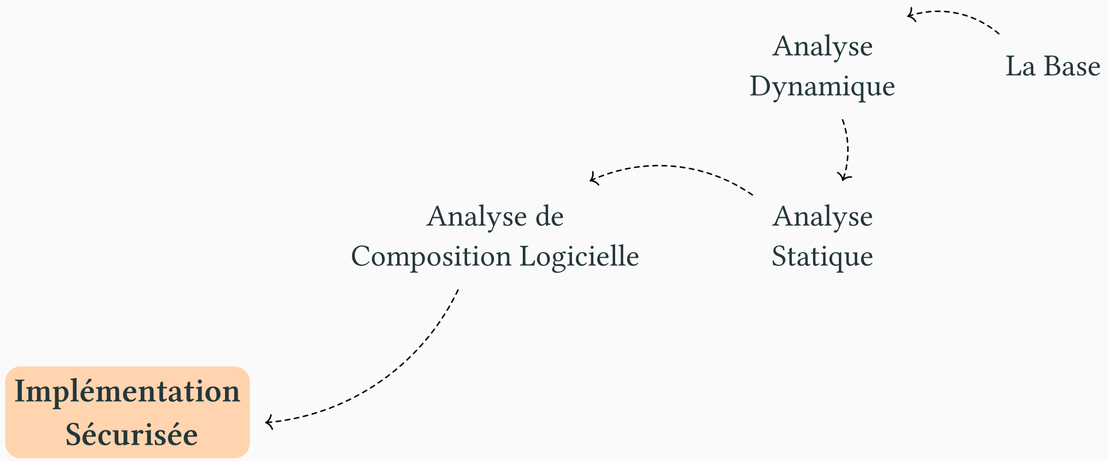
4.3.2 Démo

```
depscan --src "${DEMO_JUICE_SHOP_BUNDLE}" --bom "${DEMO_JUICE_SHOP_BUNDLE}/
bom.json" --profile research -t javascript
< reports/depscan-javascript.json jq -s 'map(select(.reachable_flows != 0))'
< reports/depscan-javascript.json jq -s 'map(select(.severity ==
"CRITICAL"))'
```


Le JuiceShop n'a pas de `package-lock.json` (ils packagent le SBOM généré lors des builds https://github.com/juice-shop/juice-shop/issues/2268).

À la première exécution, il commence par récupérer une base de données de vulnérabilités connues.

5 Implémentation Sécurisée



5 Implémentation Sécurisée

💡 La meilleure façon de gérer les bugs, c’est de ne pas en écrire.

C’est une approche d’extrême gauche.

5.1.1 Quelques ressources pour se former, s'entrainer

5.1.1.1 Des plateformes d'entraînement

Avec des challenges : des exercices où l'objectif est d'exploiter une faille de sécurité pour obtenir un secret ou un accès caché.

- https://cryptohack.org/ – Orientée cryptographie ;
- https://pwn.college/ – Beaucoup de challenges de bonne qualité, assez velu, mais très progressif ;
- https://www.hackthebox.eu – Des environnements réalistes ;
- Et bien d'autres !

5.1.1.2 OWASP [11]

- https://cheatsheetseries.owasp.org : Des ensembles d'informations concises sur différent.e.s principes, pratiques, outils, etc. ;
- Top 10 : Enquête sur les risques les plus critiques trouvés dans les applications web ;

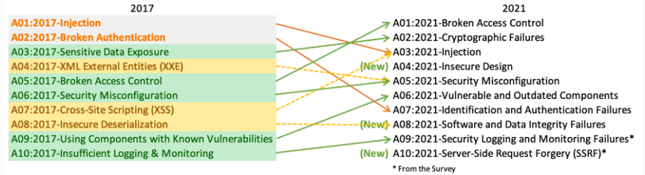


Figure 10: Le Top 10 des vulnérabilités dans les applications web en 2021

- Vulnerable Web Applications Directory : Liste d'applications vulnérables à dessein. Juice Shop est réaliste et utilise des technologies modernes ;
- Mais aussi : des librairies, des guides, des évènements, etc.

5.1.1.3 Autres

- tl;dr sec – Infolettre hebdomadaire sur des outils, des pratiques, des retours d'expérience.
- The Linux Foundation, Developing Secure Software - Un cours gratuit en ligne pour apprendre les bases de la programmation sécurisée.

OWASP est une association à but non lucratif dont les membres sont des praticiens dans le domaine de la sécurité.

- cheatsheets exemples : des conseils pour gérer l'authentification, les autorisations, etc. ; des conseils de sécurité dans Docker, Django, etc.

Motiver et éclairer des décisions sur le design, l'implémentation, les tests, le déploiement, etc.
Reconnaitres des faiblesses, et trouver des menaces.

1. What are we working on?
2. What can go wrong?
3. What are we going to do about it?
4. Did we do a good enough job?

— Threat Modeling Manifesto [12]

5.2.1 Comment procéder ?

- Entrée : Représentation du système (par exemple, un diagramme de flux de données) ;
- Sortie : Ensemble de menaces, et les protections à mettre en place.

5.2.2 Diagramme de flux de données (Data Flow Diagram)

Représenter les acteur.ice.s, les services et applications, les stockages de données, et les périmètres de confiance du système.

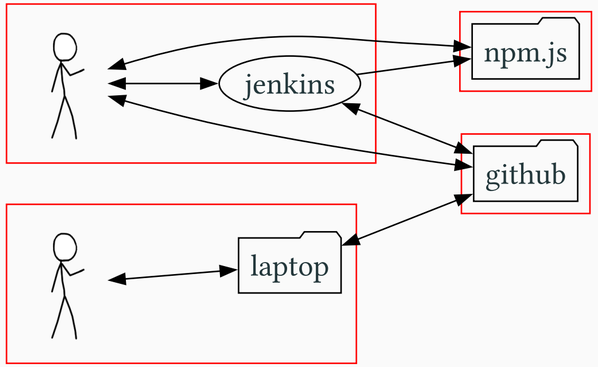


Figure 11: Un exemple de diagramme de flux de données.

5.2.3 Identifier des menaces

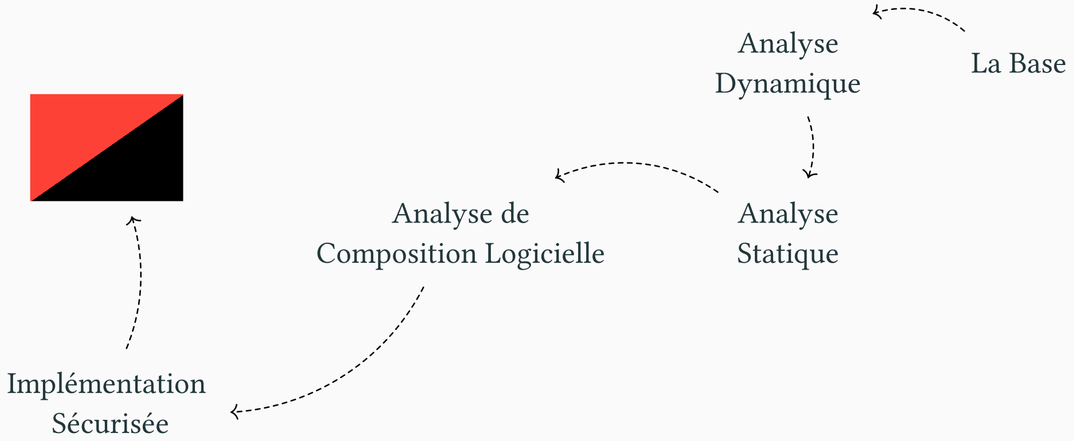
Regarder le diagramme en se grattant la tête, et en plissant les yeux.

Figure 12: Vous, quand vous voyez un truc qui cloche.

- LINDDUUN (Linkability, Identifiability, Non-repudiation, Detectability, Data disclosure, Unawareness/Undetectability, Non-compliance) ;
- STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege) ;
- Et autres.

6 Conclusion

6 Conclusion



6 Conclusion

—

On est arrivés ? Au bout de la présentation.

- Responding to change over following a plan

— Agile Manifesto [13]

- A culture of finding and fixing [security] issues over checkbox compliance.
- People and collaboration over processes, methodologies, and tools.
- A journey of understanding over a security or privacy snapshot.
- Doing [security] over talking about it.
- Continuous refinement over a single delivery.

— (à peu près) Threat Modeling Manifesto [12]

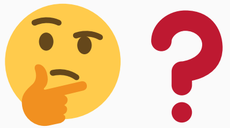
Rappel : la boucle DevSecOps est un continuum ; Les étapes précédentes peuvent/doivent nourrir notre vision et compréhension du système, et guider nos prochains pas vers un ensemble plus sécurisé.

C'est notre responsabilité de conserver la sécurité de nos utilisateurices et de leurs données malgré les changements.

Et s'adapter aux changements, c'est être agile. Le "Threat Modeling Manifesto" offre un aperçu de ce à quoi une approche de sécurité agile peut ressembler.

- Compliance = conformité.
- Security or privacy snapshot = Capture de l'état de la sécurité à un

Questions ?



Merci de m'avoir écouté.
Est-ce que vous avez des questions?

7 Crédits

7 Crédits

7 Crédits

- typst - A new markup-based typesetting system that is powerful and easy to learn.
- typix - Deterministic Typst compilation with Nix.
- touying - Creating slides in Typst.
- Nix/NixOS - A declarative package and system configuration manager.

7 Crédits

Bibliography

[1] Veracode, “State of Software Security.” Accessed: Oct. 01, 2024. [Online]. Available: https://github.com/jacobdjwilson/awesome-annual-security-reports/blob/main/Annual%20Security%20Reports/2024/Veracode-State-of-Software-Security-Report-2024.pdf?raw=true

[2] “Zed Attack Proxy.” Accessed: Oct. 01, 2024. [Online]. Available: https://www.zaproxy.org/

[3] “American Fuzzy Lop.” Accessed: Oct. 01, 2024. [Online]. Available: https://lcamtuf.coredump.cx/afl/

[4] K. Claessen and J. Hughes, “QuickCheck: a lightweight tool for random testing of Haskell programs,” in *Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, 2000, pp. 268–279.

[5] V. A. Alfred, S. L. Monica, and D. U. Jeffrey, *Compilers principles, techniques & tools*. pearson Education, 2007.

[6] S. Muchnick, *Advanced compiler design implementation*. Morgan kaufmann, 1997.

[7] “SemGrep.” Accessed: Oct. 01, 2024. [Online]. Available: https://semgrep.dev/

[8] “CodeQL.” Accessed: Oct. 01, 2024. [Online]. Available: https://codeql.github.com/

[9] “System Package Data Exchange (SPDX®).” Accessed: Oct. 01, 2024. [Online]. Available: https://spdx.dev/

[10] “CycloneDX Specification Overview.” Accessed: Oct. 01, 2024. [Online]. Available: https://cyclonedx.org/specification/overview/

[11] “Open Worldwide Application Security Project (OWASP).” Accessed: Oct. 01, 2024. [Online]. Available: https://owasp.org/

[12] “Threat Modeling Manifesto.” Accessed: Oct. 01, 2024. [Online]. Available: https://www.threatmodelingmanifesto.org/

[13] “Agile Manifesto.” Accessed: Oct. 01, 2024. [Online]. Available: https://agilemanifesto.org/

Bibliography

—