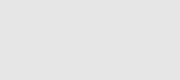
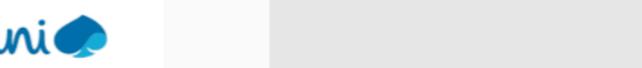


La sécurité peut aussi "Shift Left"

Xavier "Pamplemousse" Maso

2024-10-30



1 / 85

Merci à celleux qui nous permettent de bénéficier d'un super évènement,
accessible.

Xavier Maso

 Secrétaire d'Okiwi

 Ingénieur en Sécurité des Applications, à Oracle NetSuite

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.

 github.com/Pamplemousse git.sr.ht/~pamplemousse¹

 [@Pamplemouss_](https://mamot.fr/@Pamplemouss_)

 www.xaviermaso.com

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

Hypothèses acceptées :

- On veut “shift left” ;
- La sécurité c'est important.

Pré-requis :

- À peu près distinguer sa droite de sa gauche.

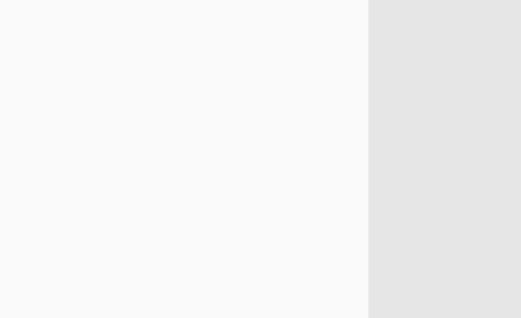
Cette présentation ne couvrira pas le pourquoi on veut “shift left”, ni pourquoi faire attention à la sécurité.

Hypothèses acceptées :

- On veut “shift left” ;
- La sécurité c'est important.

Pré-requis :

- À peu près distinguer sa droite de sa gauche.



Cette présentation ne couvrira pas le pourquoi on veut “shift left”, ni pourquoi faire attention à la sécurité.

Introduction



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.

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

— Threat Modeling Manifesto [1]

Introduction

—

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

“Threat Modeling Manifesto” cité verbatim, mais je trouve que ça s’applique bien à la sécurité en général.

- Compliance = conformité.
- Security or privacy snapshot = Capture de l'état de la sécurité à un instant T.
- Continuous refinement = amélioration continue.

Table des matières

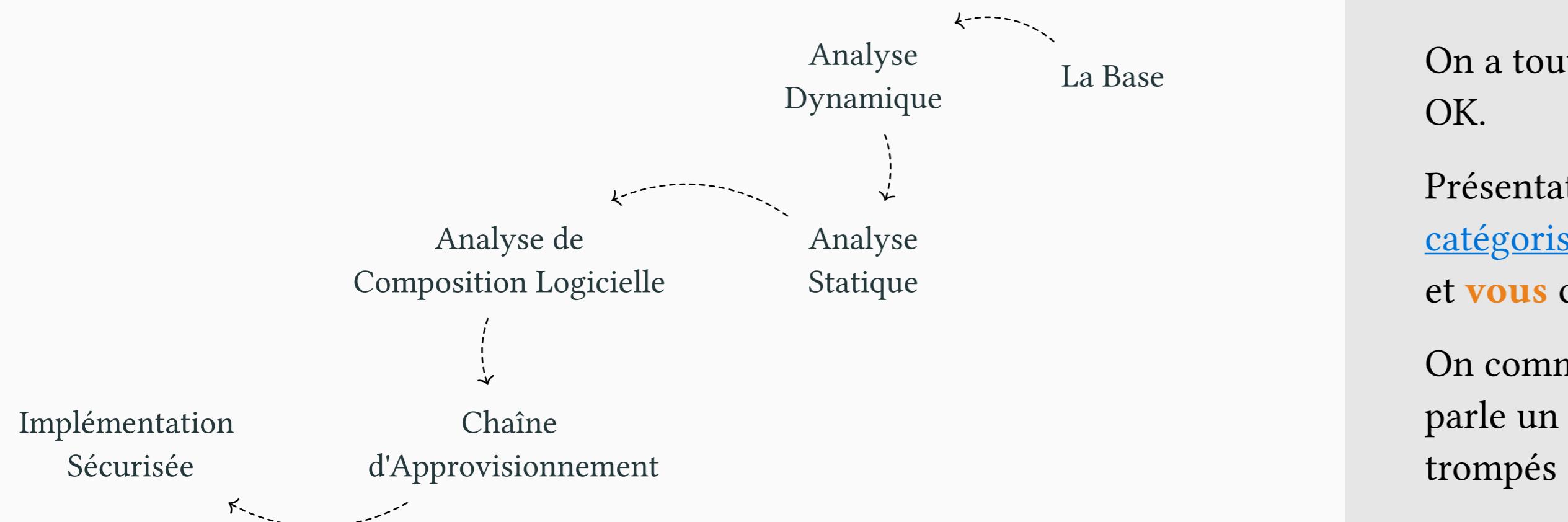


Table des matières

On a tout un espace à explorer, on aura pas le temps de tout faire, et c'est OK.

Présentation dont vous êtes le héros : y'a pleins de sujets, qui sont un peu catégorisés, qui suivent essentiellement un mouvement de droite à gauche, et **vous** choisissez ceux sur lesquels on s'arrêtera.

On commence par "la base", pour s'assurer d'où on met les pieds et que l'on parle un language commun, et aussi permettre à celleux qui se seraient trompés de salle sans s'en rendre compte de partir.

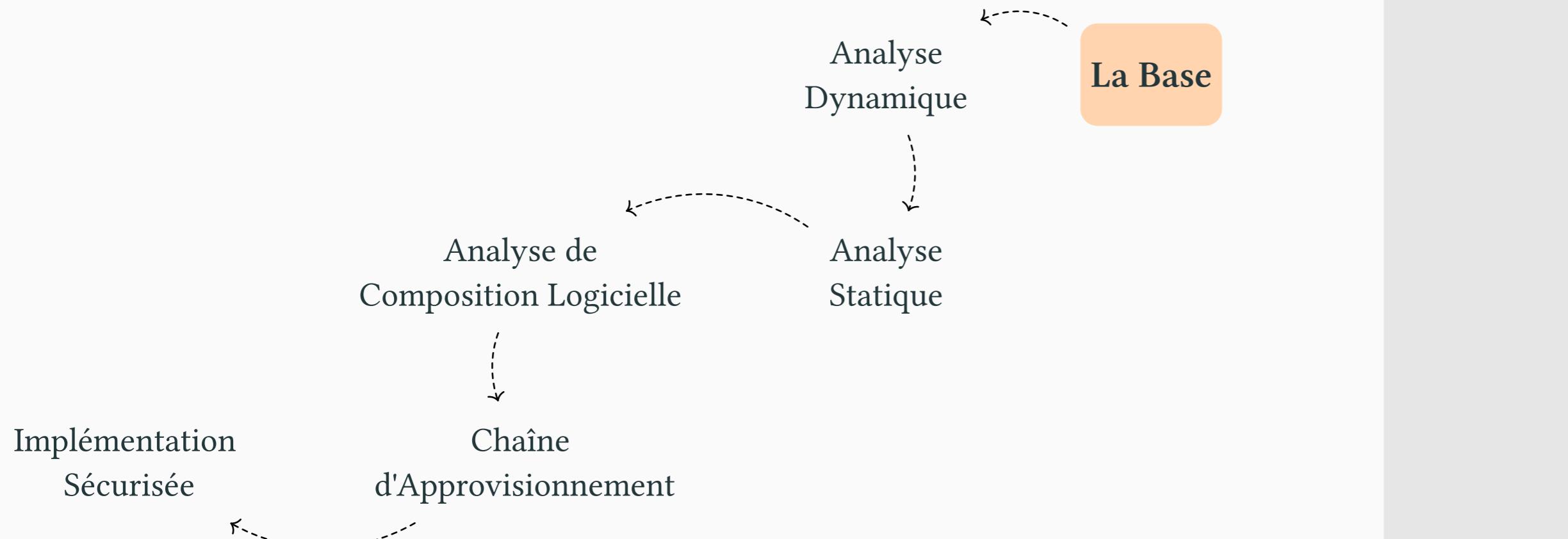
Table des matières

1 La Base	7
1.1 Sécurité	9
1.2 Boucle DevSecOps	10
1.3 Terminologie	12
2 Analyse Dynamique (Dynamic Application Security Testing - DAST)	14
2.1 Présentation	16
2.2 Zed Attack Proxy - ZAP	22
3 Analyse Statique (Static Application Security Testing - SAST)	25
3.1 Présentation	27
3.2 Sous le capot	28
3.3 SemGrep	34
3.4 CodeQL	37
4 Analyse de Composition Logicielle (Software Composition Analysis - SCA)	41
4.1 Présentation	43
4.2 Nomenclature Logicielle (Software Bill of Materials - SBOM)	55
4.3 OWASP dep-scan	59
5 Chaîne d'Approvisionnement	61
5.1 Présentation	63
5.2 Supply chain Levels for Software Artifacts (SLSA)	67
6 Implémentation Sécurisée	70
6.1 Principes de Développement Sécurisé	72
6.2 Modèle de Menaces (Threat Modeling)	76
7 Conclusion	79
8 Crédits	83

Table des matières

La Base

1 La Base



1 La Base

—

- Confidentialité (Confidentiality)
 - Intégrité (Integrity)
 - Disponibilité (Availability)
- ... des données et systèmes.



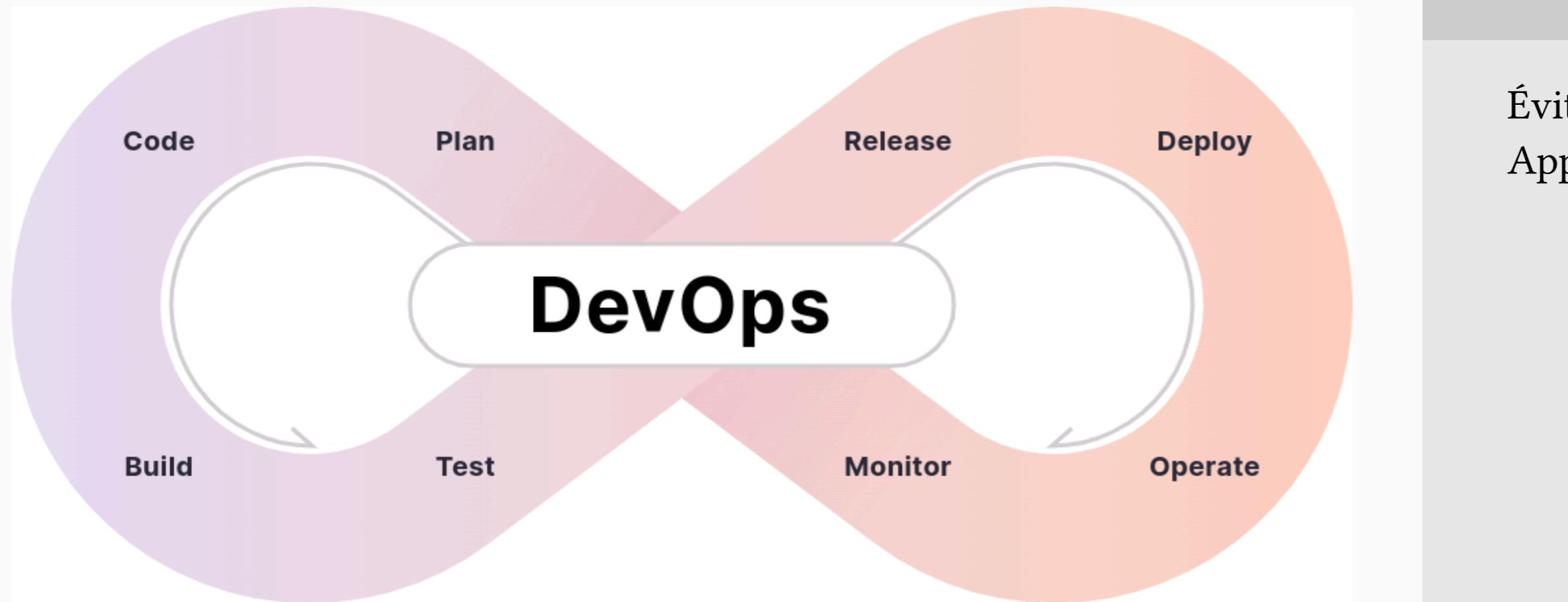


Figure 1: Boucle DevOps “classique”.

1 La Base

– 1.2 Boucle DevSecOps

Éviter l'effet tunnel, et raccourcir les boucles de feedback.
Approche continue plutôt que discrète.

1.2 Boucle DevSecOps

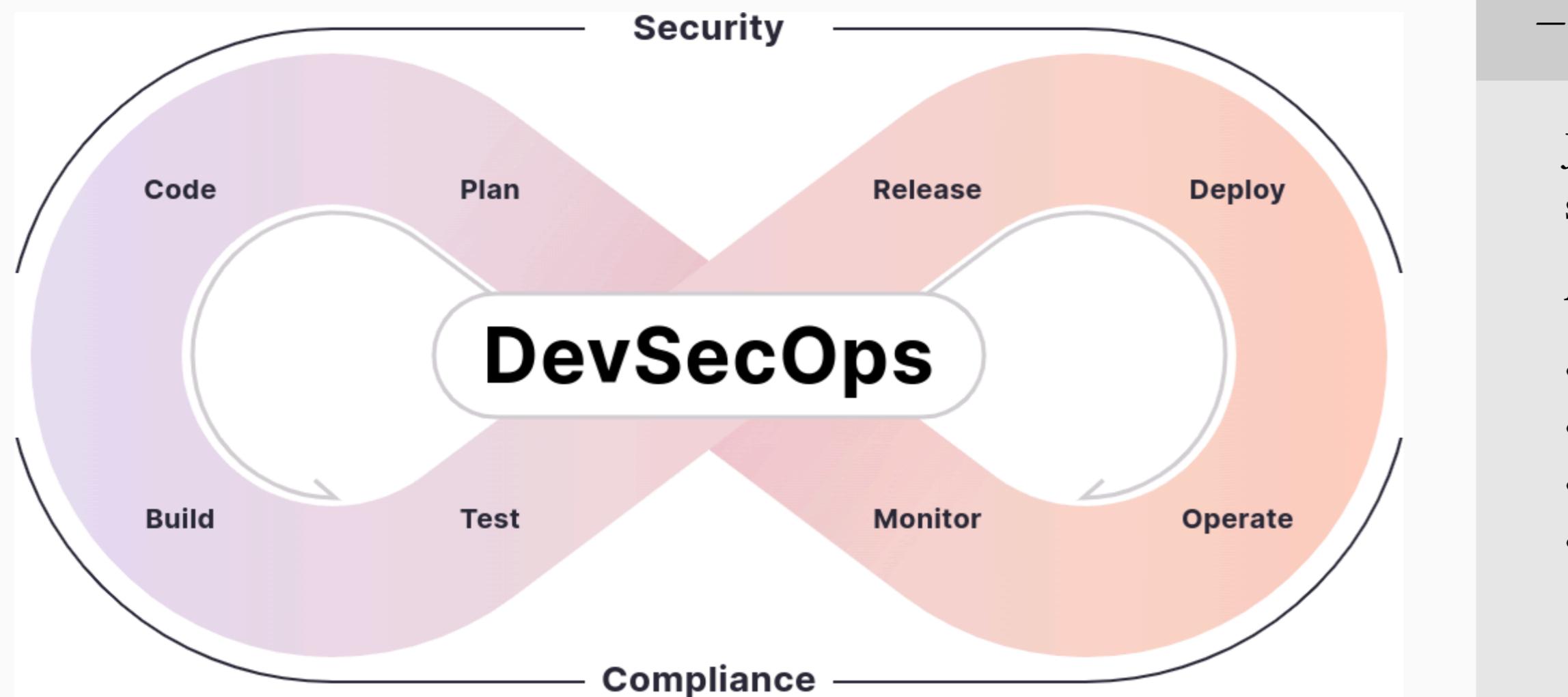


Figure 2: Boucle DevOps, avec un enrobage de sécurité.

1 La Base

– 1.2 Boucle DevSecOps

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: chaîne d'approvisionnement, analyse de composition
- code: revue de code, principes de programmation sécurisée
- plan: modèle de menaces

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 de variables.

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.

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

e.g. Une entrée utilisateurice ajoutant du code SQL arbitraire à exécuter (SQL injection).

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) Une collection publique de vulnérabilités logicielles connues. Un identifiant pour une vulnérabilité reconnue.

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

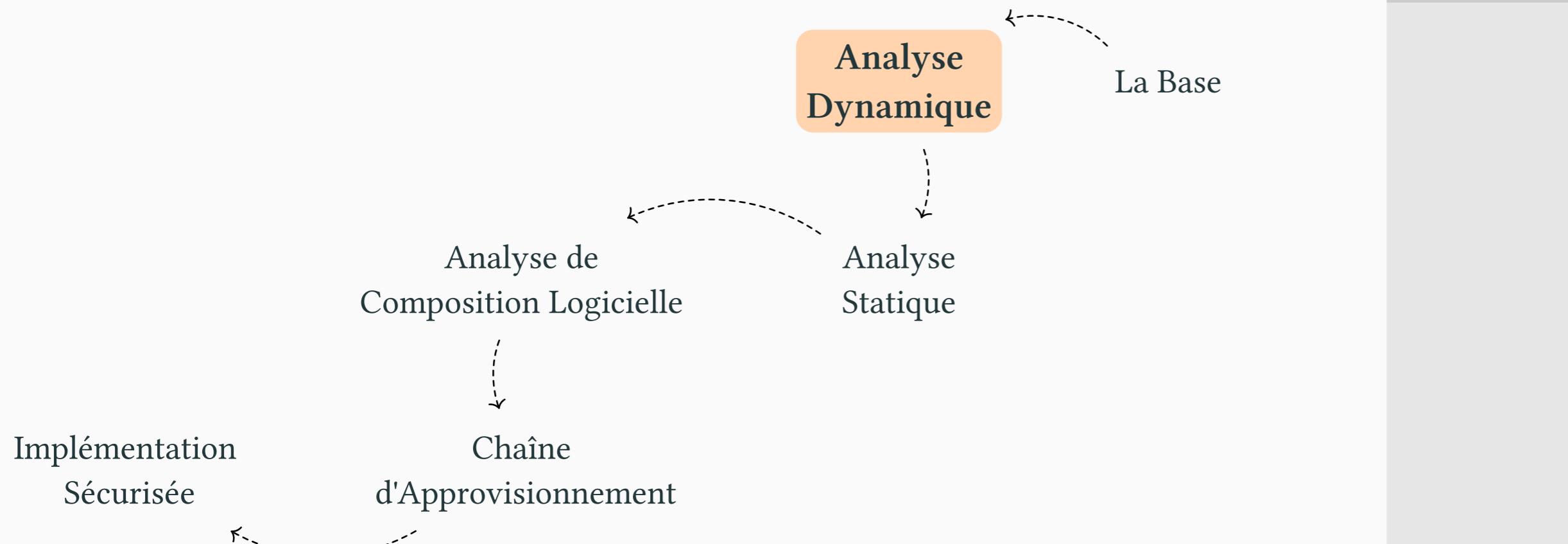
– 1.3 Terminologie

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 Analyse Dynamique (Dynamic Application Security Testing - DAST)



2 Analyse Dynamique (Dynamic Application Security Testing - DAST)

2.1.0.0.1 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 [2]

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

– 2.1 Présentation

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.

2.1.0.0.2 Techniques

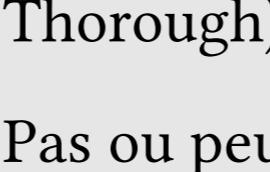
2.1.0.0.2.1 Tests manuels

Utiliser l'artefact 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.



– 2.1 Présentation

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

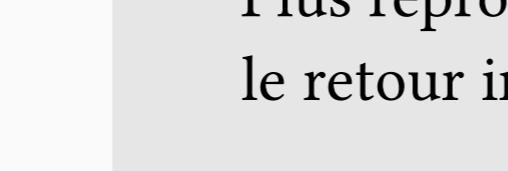
Pas ou peu reproductible ⇒ on investit à chaque fois la même quantité de temps.

2.1.0.0.2.2 Tests automatisés

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

Par exemple : Écrire et exécuter des tests unitaires, d'intégration, bout en bouts¹.

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



¹Par exemple en utilisant ZAP [3] pour des applications web.

– 2.1 Présentation

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

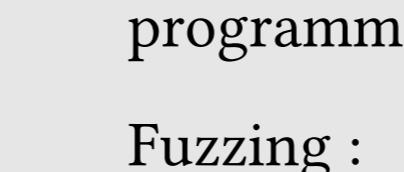
2.1.0.0.2.3 Fuzzing, Tests basés sur les propriétés (PBT)

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

Exemples : AFL [4], QuickCheck [5].

+ Grande couverture en fonction du temps.

- Processus global non déterministes, coûteux à exécuter, plus coûteux à mettre en place.



– 2.1 Présentation

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

Fuzzing :

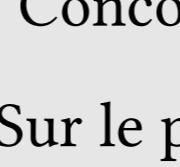
- Génération aléatoire peut être guidée selon des heuristiques (par exemple en mutant des entrées qui ont découvertes de nouvelles portions de code), pour encourager la manifestation de bugs.
- Est-ce que le programme crashe ?

Tests basés sur les propriétés

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

2.1 Présentation

2.1.0.0.2.4 Exécution Symbolique, Concolique



Symbolique Interpréter l'artefact sur une représentation de ses entrées, pour obtenir des expressions et des contraintes à résoudre.

Concolique Utiliser des entrées concrètes, mais suivre la représentation symbolique pour améliorer les générations d'entrées suivantes.

Exemples : `angr` [6], `driller` [7].

+ Très efficace, excellente couverture.

- Coûteux à mettre en place, coûteux en ressources, besoin d'accès au programme analysé.

2 Analyse Dynamique (Dynamic Application Security Testing - DAST)

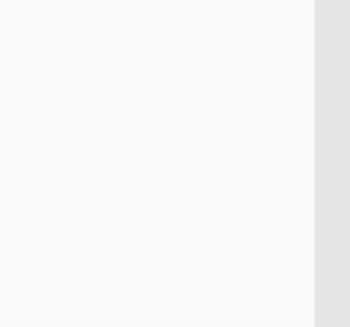
– 2.1 Présentation

“Concolique” est le portemanteau de “concret” et “symbolique”.

Sur le papier, avec suffisamment de temps \Rightarrow couverture parfaite.

2.2.0.0.1 Présentation

- Scanneur d'application web : proxy permettant de trouver des vulnérabilités dans les applications web ;
- Gratuit et open-source ;
- Anciennement OWASP, maintenant Checkmarx¹.
 - <https://www.zaproxy.org/> [3]

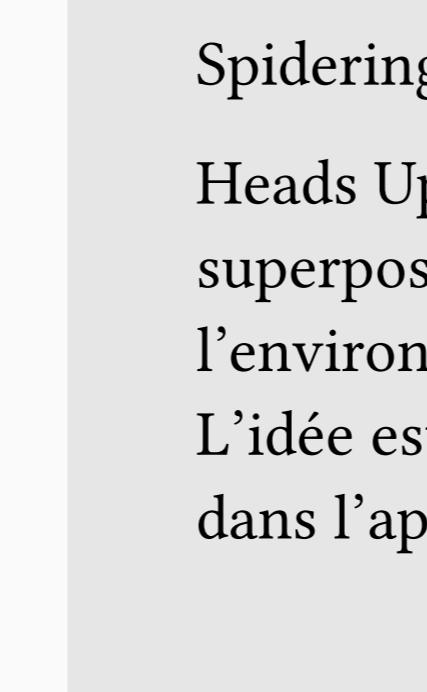


¹Il semblerait sous un modèle “open-core”, c.f. <https://www.zaproxy.org/blog/2024-09-24-zap-has-joined-forces-with-checkmarx/>

– 2.2 Zed Attack Proxy - ZAP

2.2.0.0.2 Fonctionnement

- Interception de traffic HTTP ;
- 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, HUD¹, etc.)² ;
- Et autres fonctionnalités : points d’arrêts, édition et renvoi de requêtes, fuzzing, etc.



¹Interface 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.

– 2.2 Zed Attack Proxy - ZAP

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.

2.2.0.0.3 Exemples d'utilisation

2.2.0.0.3.1 Dans un environnement de développement

Faire passer tout son trafic HTTP dans ZAP

- Obtenir des informations “en direct” sur la sécurité ;
- Outil de débogage : enregistrer, rejouer, modifier des requêtes¹.

2.2.0.0.3.2 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 trafic généré par les tests, et vérifier les alertes.
- Programmé régulièrement :
Utiliser les capacités d'exploration, et tester les “endpoints” découverts.

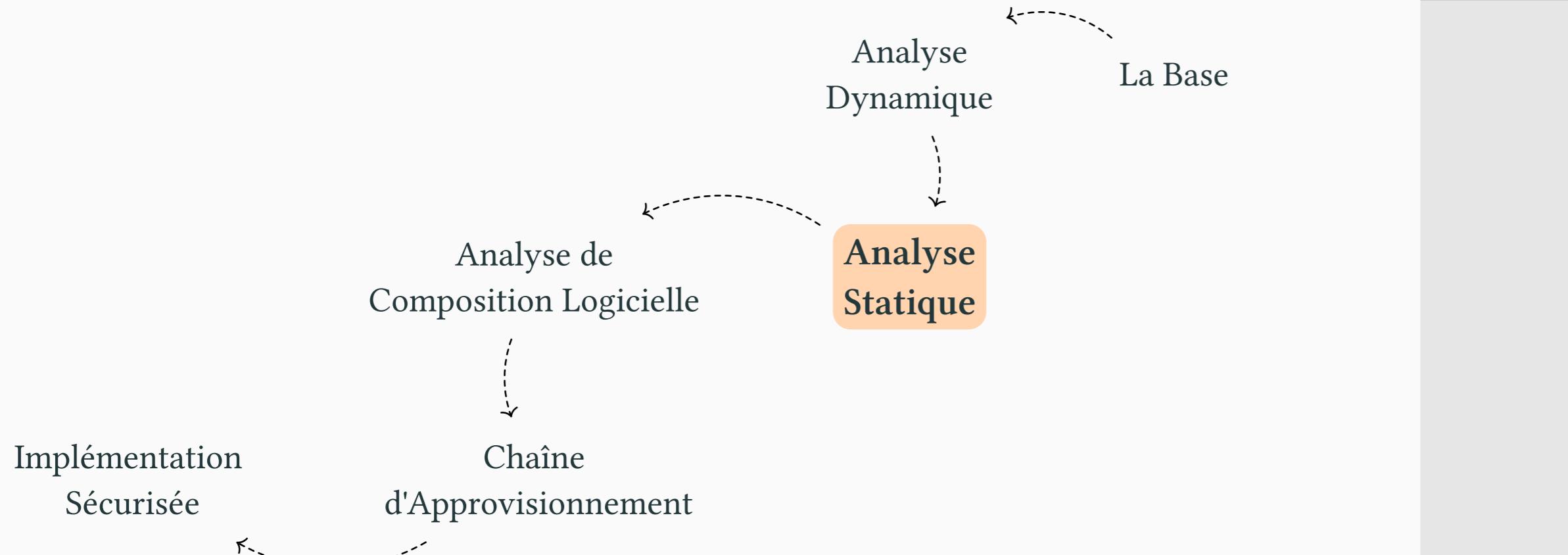
¹À la [MITM](#), [Charles](#), [Postman](#), etc.

– 2.2 Zed Attack Proxy - ZAP

3 Analyse Statique (Static Application Security Testing - SAST)

3 Analyse Statique (Static Application Security Testing - SAST)

3 Analyse Statique (Static Application Security Testing - SAST)



3 Analyse Statique (Static Application Security Testing - SAST)

3.1 Présentation

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 [2]

- Analyser 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](#), ...

3 Analyse Statique (Static Application Security Testing - SAST)

— 3.1 Présentation

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 ?

3.2.0.0.1 Examples de représentations utiles

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

Sources : [8], [9].

– 3.2 Sous le capot

3.2.0.0.1.1 Arbre de la Syntaxe Abstraite (Abstract Syntax Tree - AST)

Arbre¹ représentant la structure d'un programme, sans les détails syntaxiques présents dans le code source (parenthèses, point virgules, etc.).

```
python $EXAMPLES/ast.py
semgrep --lang python --dump-ast $EXAMPLES/ast.py > ast.result.txt

semgrep-core -dump_pfff_ast -lang=python $EXAMPLES/ast.py
semgrep-core -show_ast_json -lang=python $EXAMPLES/ast.py
```

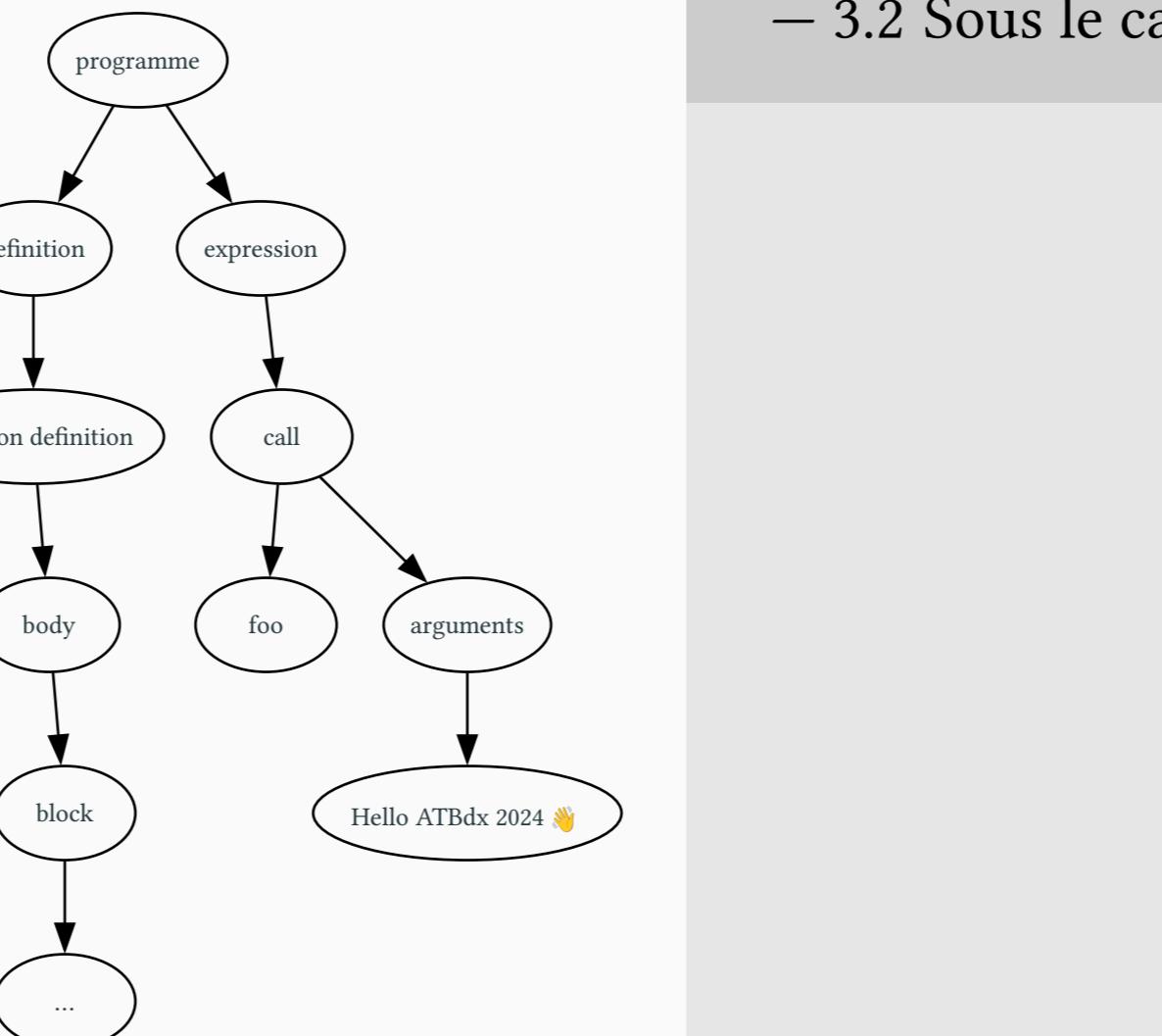
¹La structure de données, pas l'objet botanique.

– 3.2 Sous le capot

- Permet de :
 - Reconnaître des patterns de code ;
 - Générer du code à partir d'une la représentation ;
- Utilisé notamment par :
 - Des outils de reformatage de code ([ESLint](#) for JavaScript, [Black](#) for Python, etc.) ;
 - Mais aussi par des outils d'analyse de sécurité !

3.2 Sous le capot

```
def foo(bar: str):  
    print(bar)  
  
foo("Hello ATBdx 2024 🙋‍♂️")
```



3 Analyse Statique (Static Application Security Testing - SAST)

– 3.2 Sous le capot

3.2.0.0.1.2 Arbre Syntaxique (Parse Tree, or Concrete Syntax Tree)

Produit de l'analyse syntaxique (parsing) du code source selon la grammaire non-contextuelle du language.

Conserve les informations syntaxiques : indentation, parenthèses, crochets, accolades, points virgules, etc.

```
python $EXAMPLES/ast.py  
semgrep-core -dump_tree_sitter_cst -lang=python $EXAMPLES/ast.py >  
cst.result.txt
```

– 3.2 Sous le capot

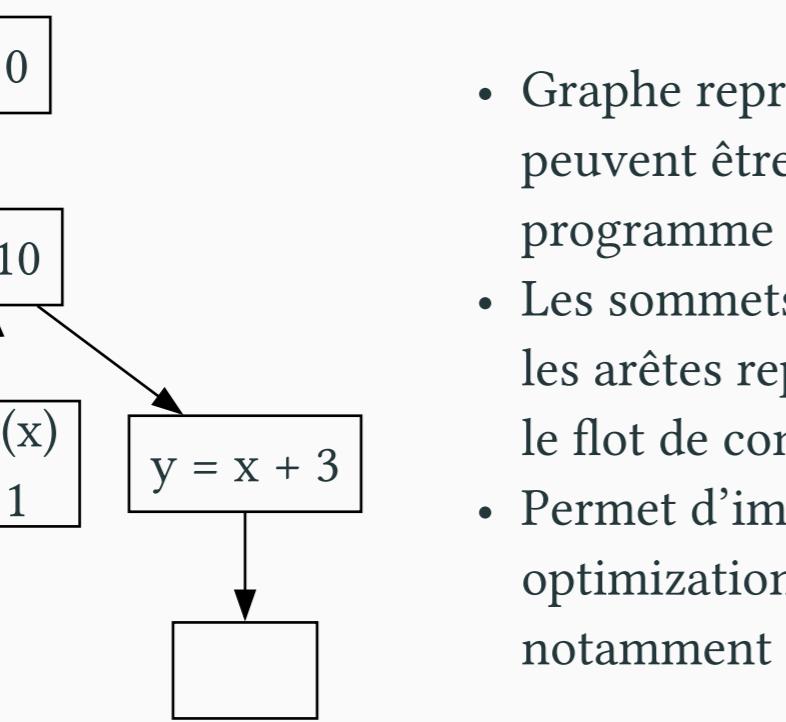
Pas forcément le plus intéressant, ni pratique pour effectuer des analyses, mais certains outils conservent des informations de cet arbre notamment pour pouvoir “respecter” la syntaxe originelle.

```

x = 0
while x < 10:
    print(x)
    x += 1
y = x + 3

```

3.2.0.0.1.3 Graphe de Flot de Contrôle (Control Flow Graph - CFG)



- Graphe représentant les chemins qui peuvent être pris par l'exécution d'un programme ;
- Les sommets sont des **blocs de base**, et les arêtes représentent des **sauts** dans le flot de contrôle ;
- Permet d'implémenter des optimisations, et des analyses, notamment de flux de données.

– 3.2 Sous le capot

Un bloc de base a une seule instruction d'entrée, ET une seule instruction de sortie.

Exemples :

- d'optimisation: élimination de code mort, propagation de constantes, etc.
- d'analyse de flux de données
 - ▶ “live analysis” – la portée d'une variable : l'ensemble des expressions où la variable peut-être utilisée (avant d'être supprimée, ou reécrite) ;
 - ▶ “reaching definitions” – pour une expression : où ont été déclarées les variables utilisées ?
 - ▶ “taint tracking”, pour suivre des valeurs “teintées” le long du flot du programme.

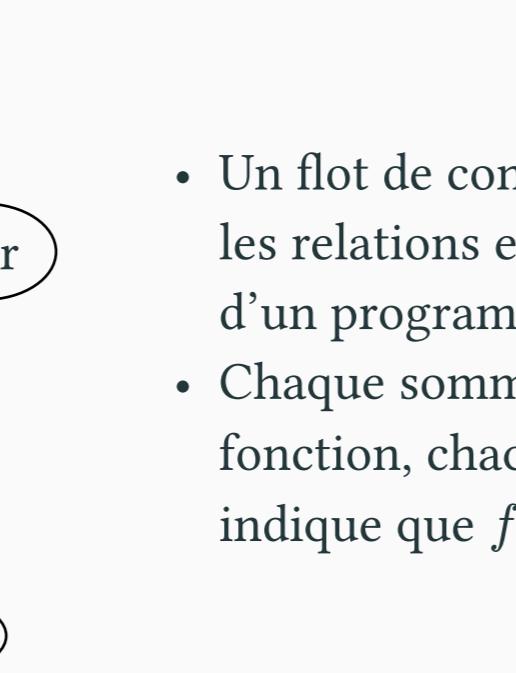
3.2.0.0.1.4 Graphe d'Appels de Fonctions (Function Call Graph)

```
def main():
    foo()
    bar()

def foo():
    baz()

def bar():
    baz()

def baz():
    print("Hello ATBdx 2024
        ")
```



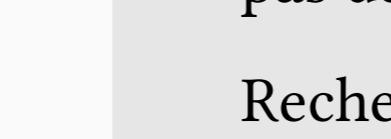
- Un flot de contrôle particulier : les relations entre les fonctions d'un programme ;
- Chaque sommet est une fonction, chaque arête $f \rightarrow g$ indique que f appelle g .

– 3.2 Sous le capot

- Quelles fonctions appellent quelles autres ?
- Où est-ce que ma fonction, ou telle API tierce vulnérable, sont appelées ?

3.3.0.0.1 Présentation

- Semantic Grep ;
- Moteur d'analyse statique, pour chercher des fragments de code selon des règles ;
- Gratuit et open-source¹ ;
- Supporte de nombreux langages² ;
- Fonctionne sur du code "partiel" ;
- Ensemble de règles maintenues par la communauté³.
 - <https://semgrep.dev/products/semgrep-code> [10]



¹Modèle open-core.

²C.f. <https://semgrep.dev/docs/supported-languages>.

³C.f. <https://github.com/semgrep/semgrep-rules>.

– 3.3 SemGrep

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'utilisateur.

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).

3.3.0.0.2 Règle SemGrep

- Un ensemble de clé-valeurs YAML¹ ;
- Représente
 - le(s) pattern(s) de code à chercher,
 - et des méta-information (language, message à afficher, sévérité, et autres.)².

3.3.0.0.3 Exemples d'utilisation

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

– 3.3 SemGrep

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.

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.

3.3.0.0.4 Démo



```
semgrep scan --config $EXAMPLES/rules.yaml $JUICE_SHOP_SOURCES --json >
result.json
< result.json | jq '.results[0]' | less
```

– 3.3 SemGrep

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.

3.4.0.0.1 Présentation

- Chaîne d'outils d'analyse statique¹ ;
- Gratuit pour scanner du code open-source² ;
- Collecte des faits à propos du code à la compilation, dans une base de données ;
- Un langage de requêtes pour extraire des informations de cette base de données ;
- Plusieurs langages supportés³ ;
- Ensemble de bibliothèques, et requêtes maintenues par la communauté⁴.

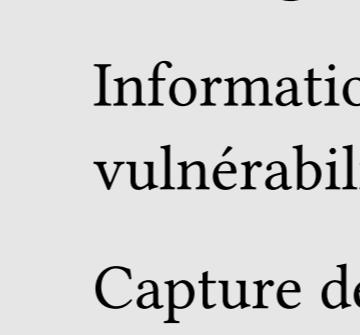
— <https://codeql.github.com> [11]

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

²C.f. <https://github.com/github/codeql-cli-binaries/blob/main/LICENSE.md>.

³C.f. <https://codeql.github.com/docs/codeql-language-guides/>.

⁴C.f. <https://github.com/github/codeql>.



– 3.4 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.

3.4.0.0.2 Comment ça fonctionne ?

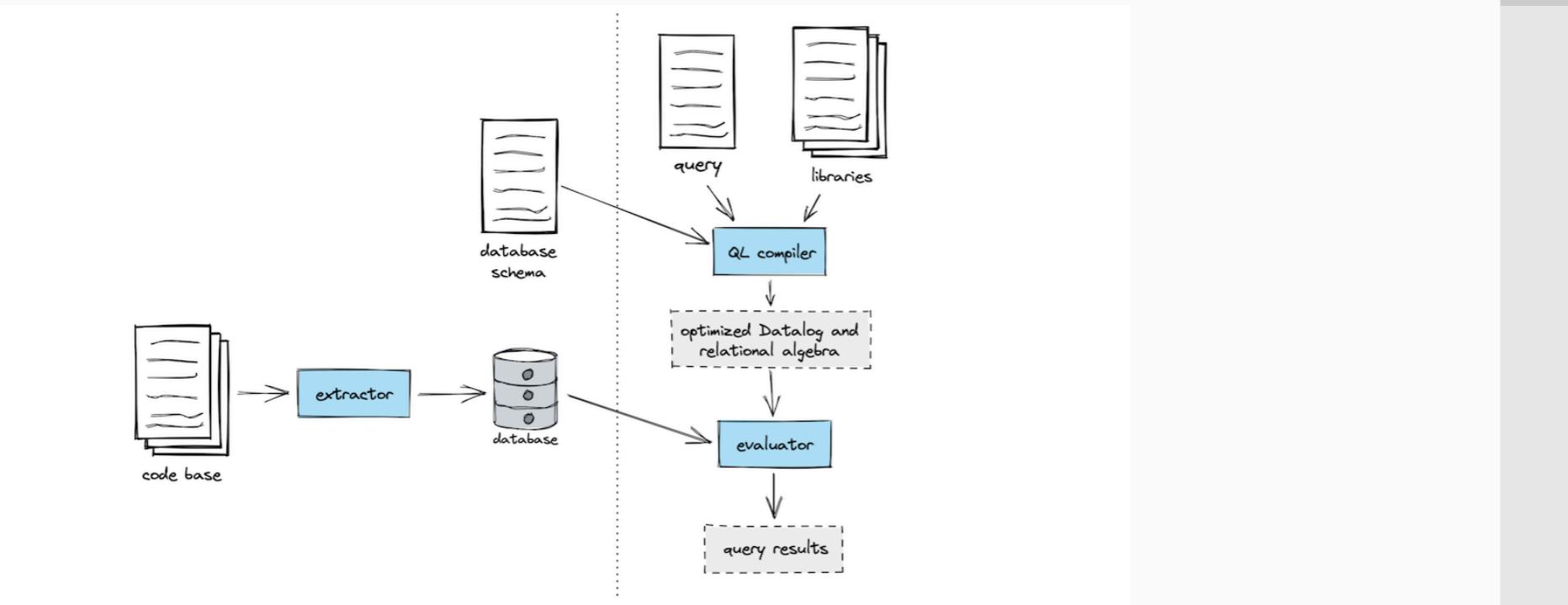


Figure 3: Schéma de fonctionnement global de CodeQL¹.

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

– 3.4 CodeQL

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

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

Les bases 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.

3.4.0.0.3 Exemples d'utilisation

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

3.4.0.0.4 Démo



```
codeql database create ./my.db \
--language typescript \
--source-root $JUICE_SHOP_SOURCES

codeql pack install $EXAMPLES
codeql query run $EXAMPLES/query.ql --database ./my.db --output result.bqrs
codeql bqrs decode result.bqrs --format json \
| jq '.#[#select].tuples | map(.[1].label)'
```

– 3.4 CodeQL

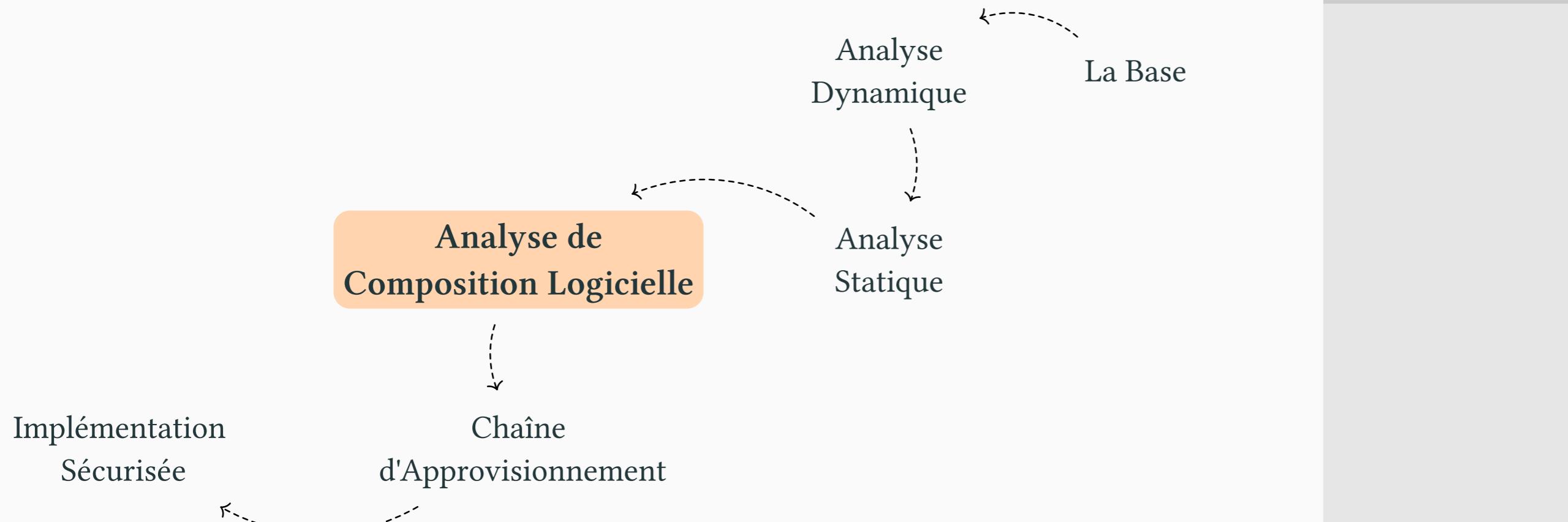
Local dataflow.

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)



4.1.0.0.1 Sur le papier

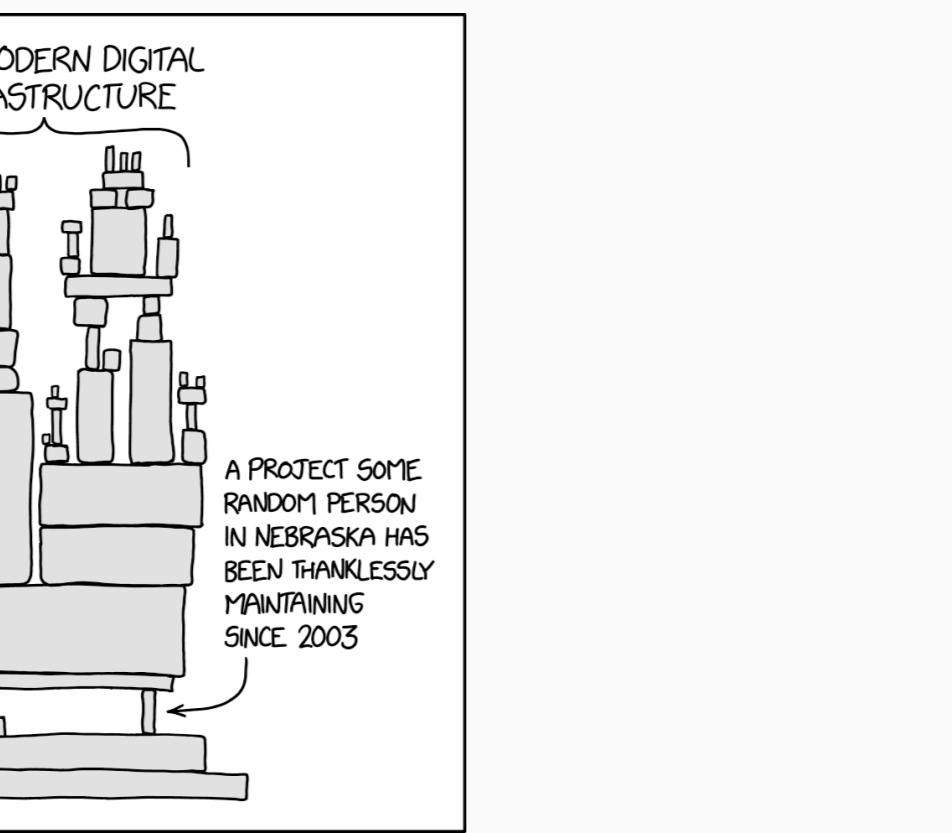


Figure 4: “Dependency” de xkcd, <https://xkcd.com/2347/>

– 4.1 Présentation

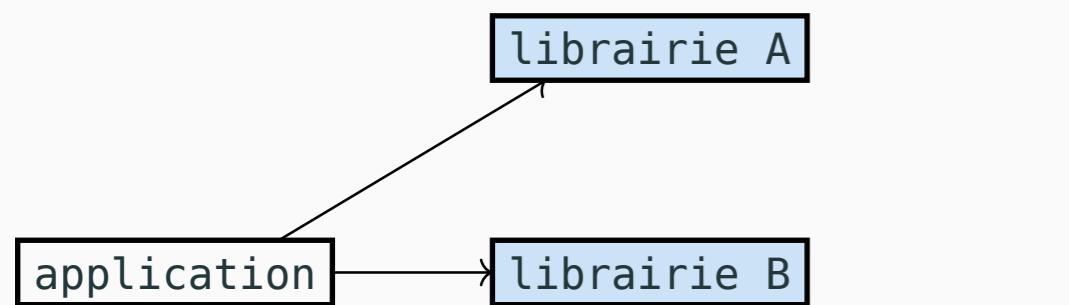
application

– 4.1 Présentation

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.



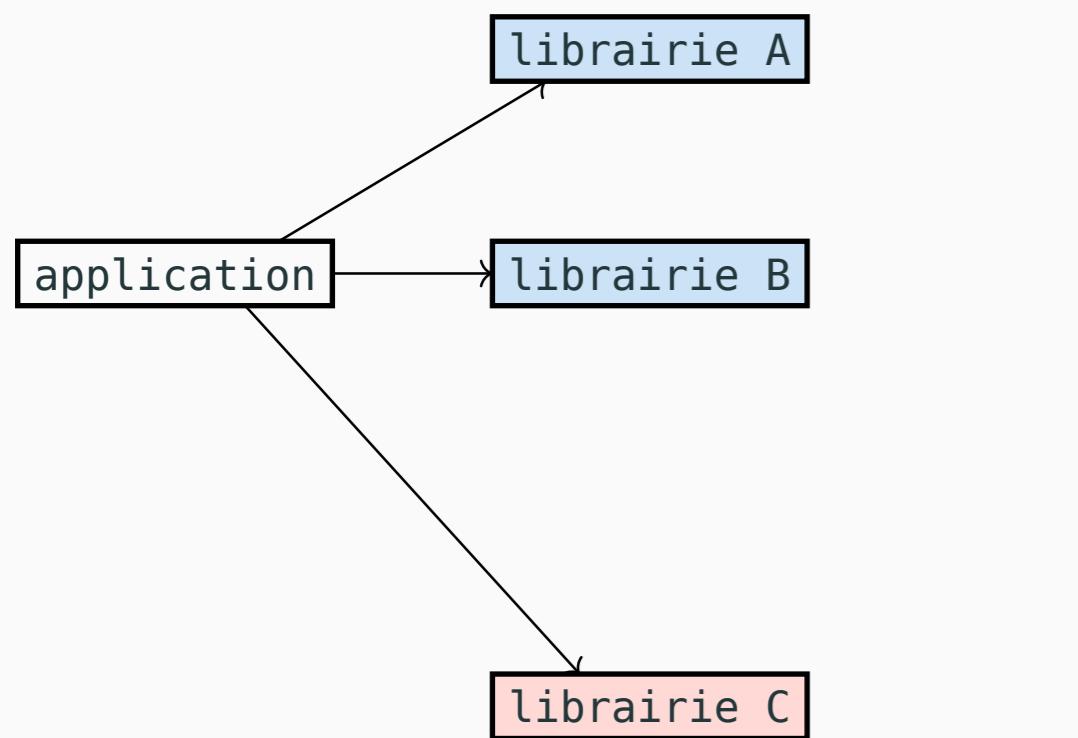
4 Analyse de Composition Logicielle (Software Composition Analysis - SCA)

– 4.1 Présentation

Imaginons que l'on développe une application.

Elle peut avoir des librairies qui :

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



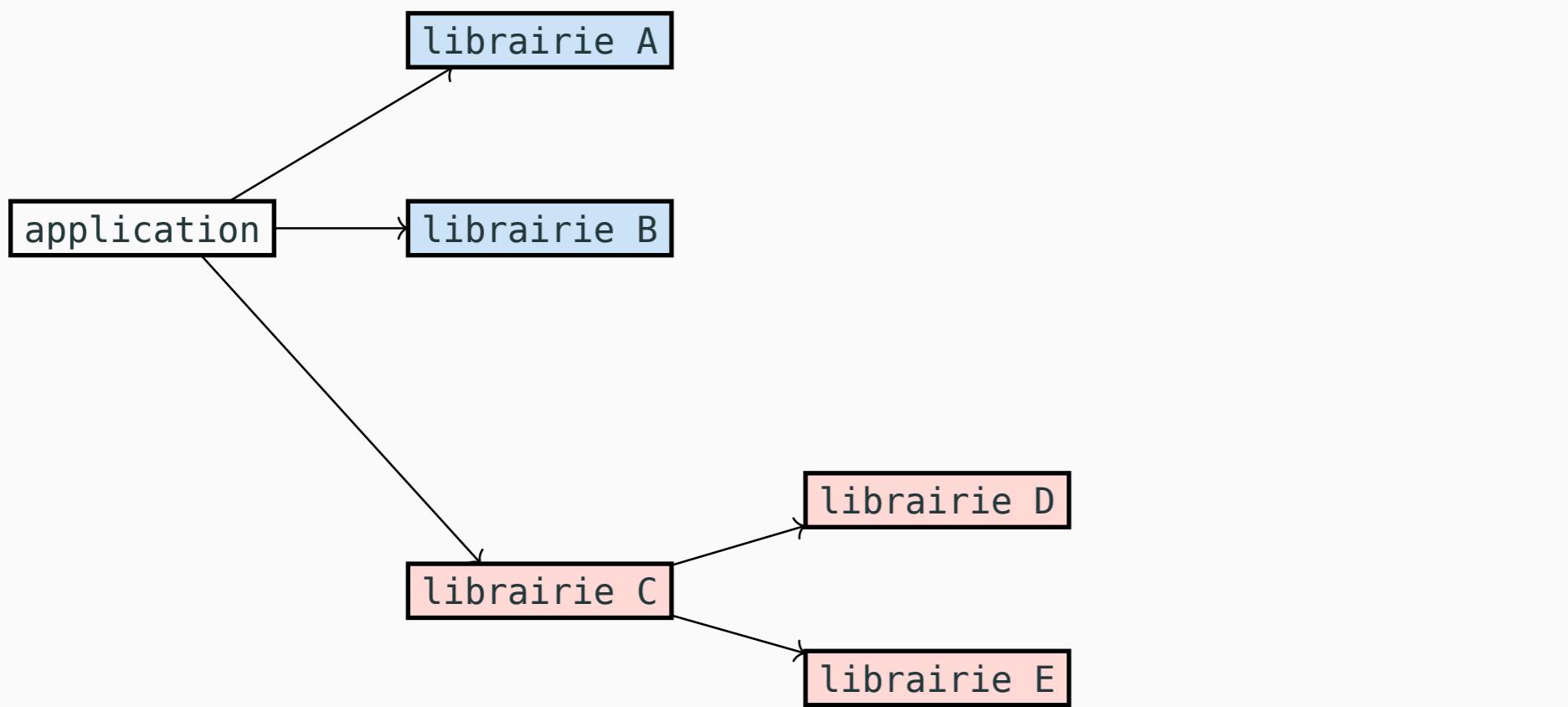
4 Analyse de Composition Logicielle (Software Composition Analysis - SCA)

– 4.1 Présentation

Imaginons que l'on développe une application.

Elle peut avoir des librairies qui :

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

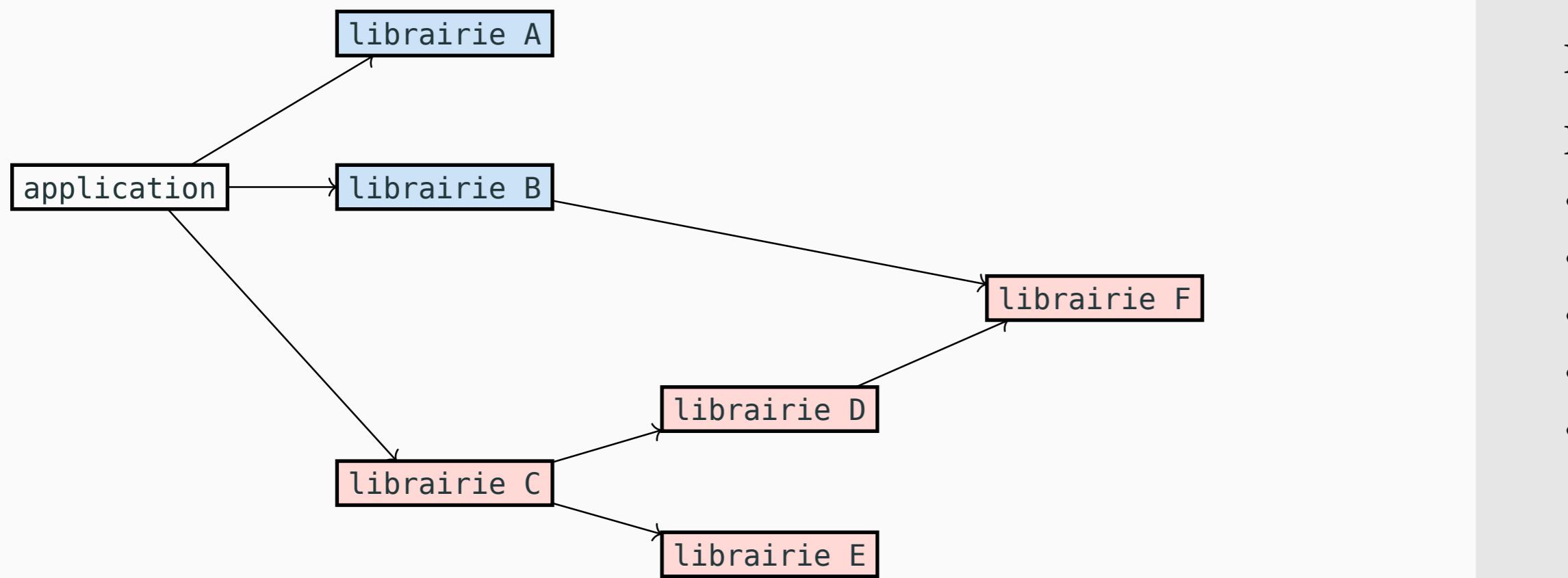


– 4.1 Présentation

Imaginons que l'on développe une application.

Elle peut avoir des librairies qui :

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



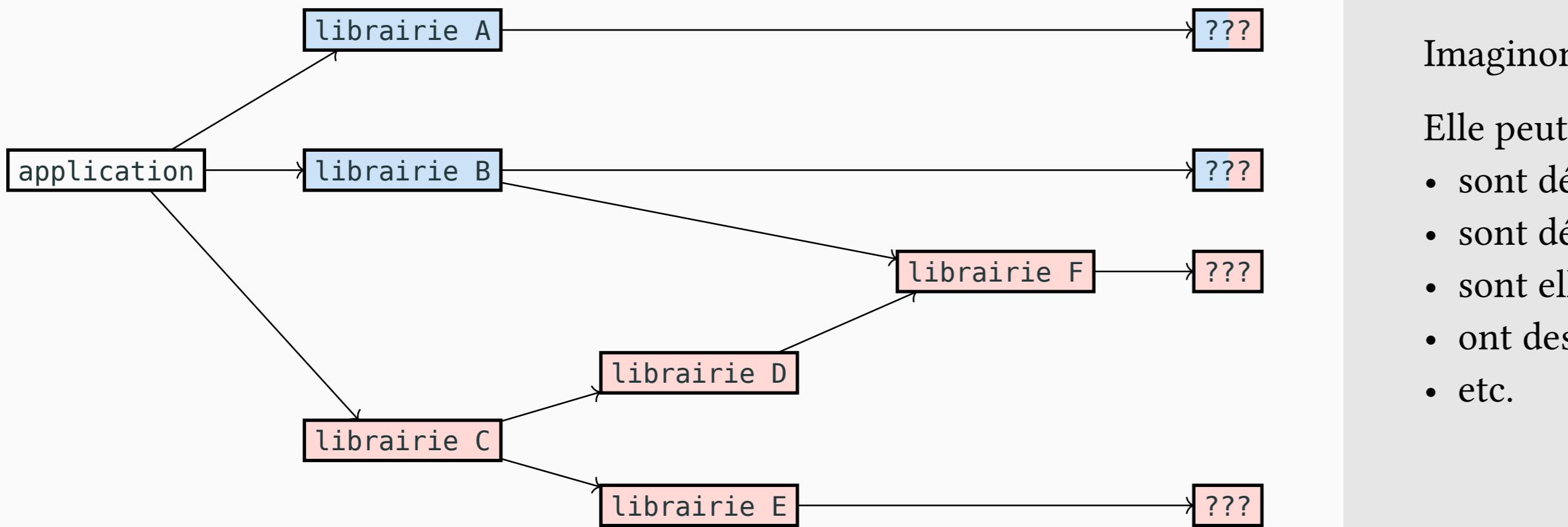
4 Analyse de Composition Logicielle (Software Composition Analysis - SCA)

– 4.1 Présentation

Imaginons que l'on développe une application.

Elle peut avoir des librairies qui :

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



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

– 4.1 Présentation

Imaginons que l'on développe une application.

Elle peut avoir des librairies qui :

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

4.1 Présentation

77% of all code in the total codebases originated from open source

— State of Software Security, 2024 [2]

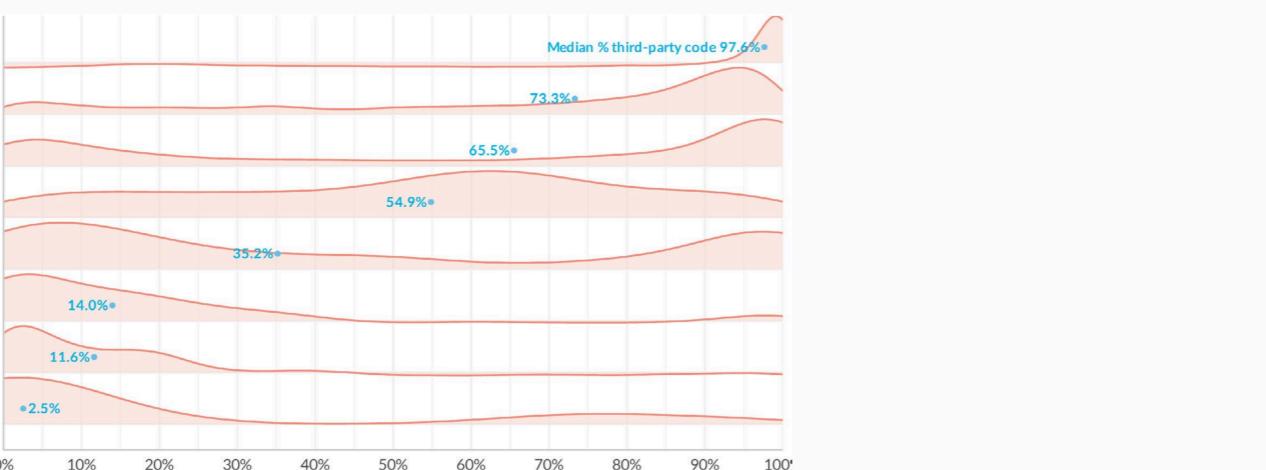
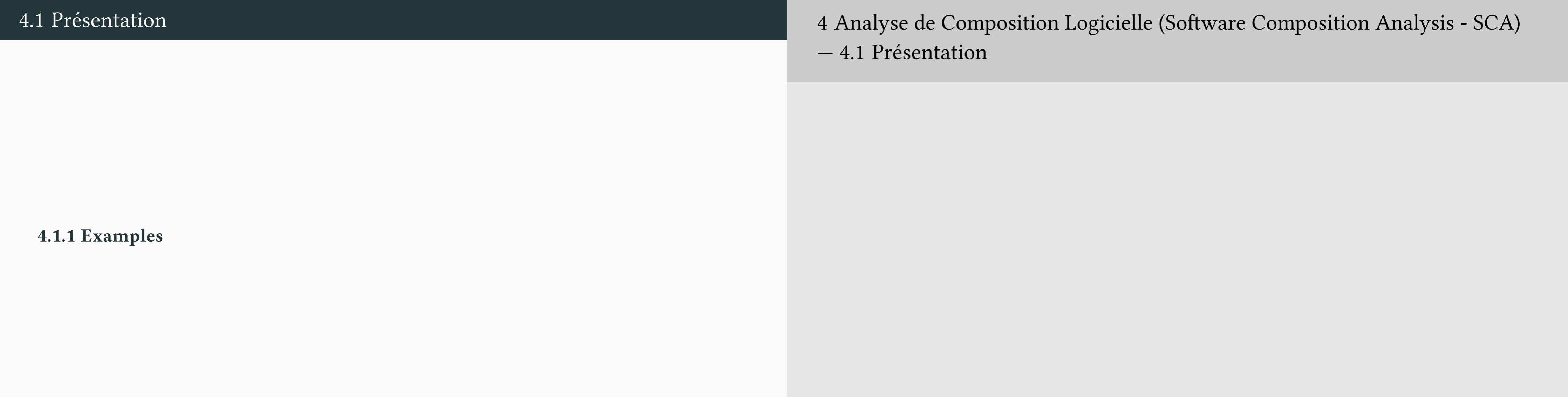


Figure 5: Percentage of third-party code in applications using third-party code in most recent scan

— State of Software Security, 2024 [2]

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

— 4.1 Présentation



4.1.1 Examples

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

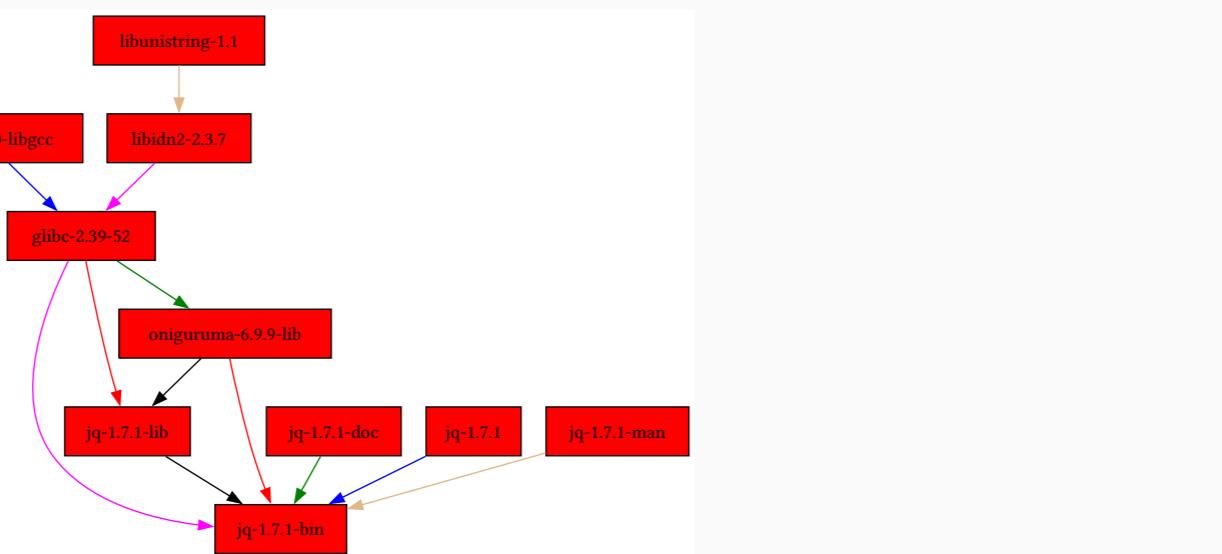


Figure 6: Clôture transitive des dépendances du programme [jq](#).

– 4.1 Présentation

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
```

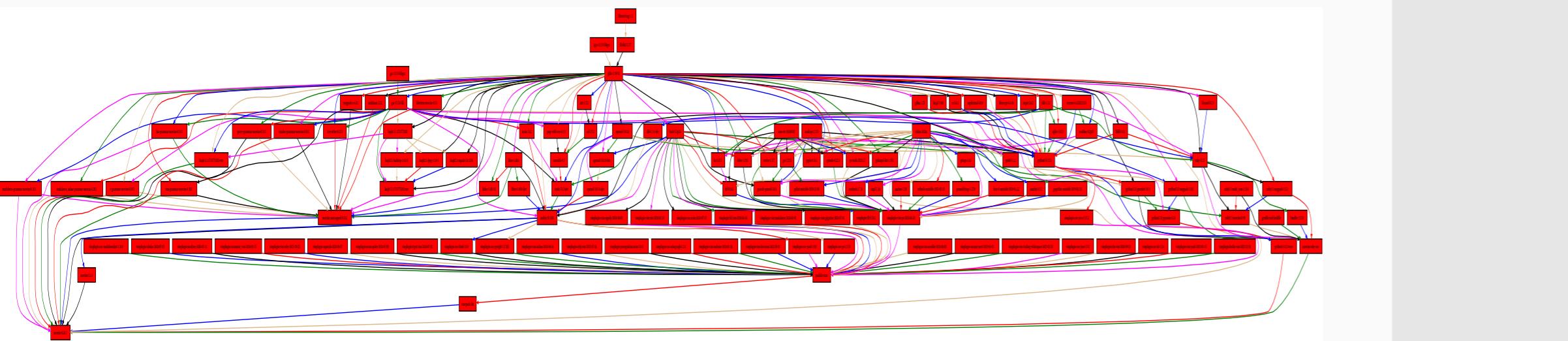


Figure 7: Clôture transitive des dépendances du programme [neovim](#).

– 4.1 Présentation

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.

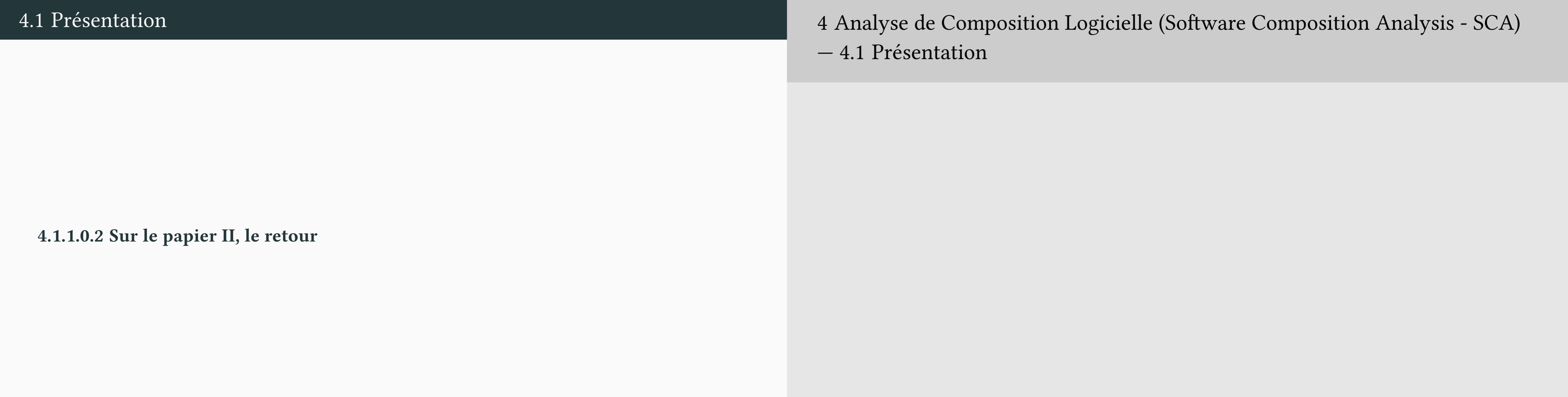
– 4.1 Présentation

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 gérer 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 couramment démarré.

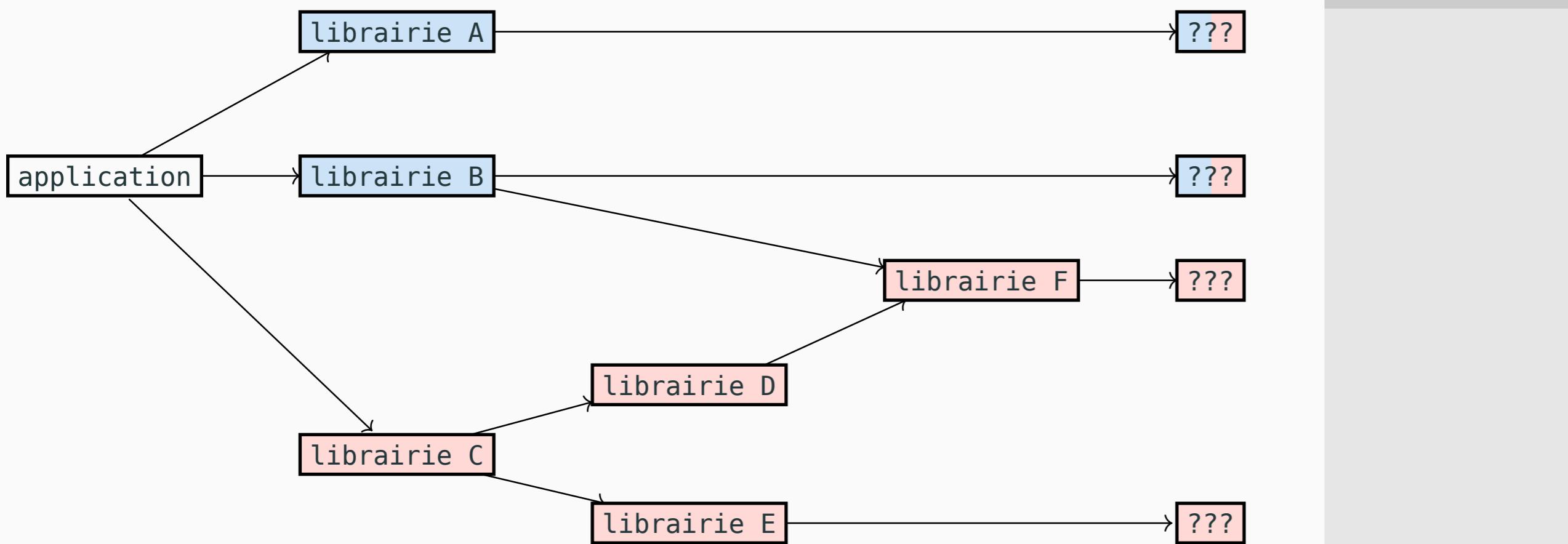
4.1.1.0.1 Les risques

- 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 licences incompatibles avec l'usage que l'on en fait ?

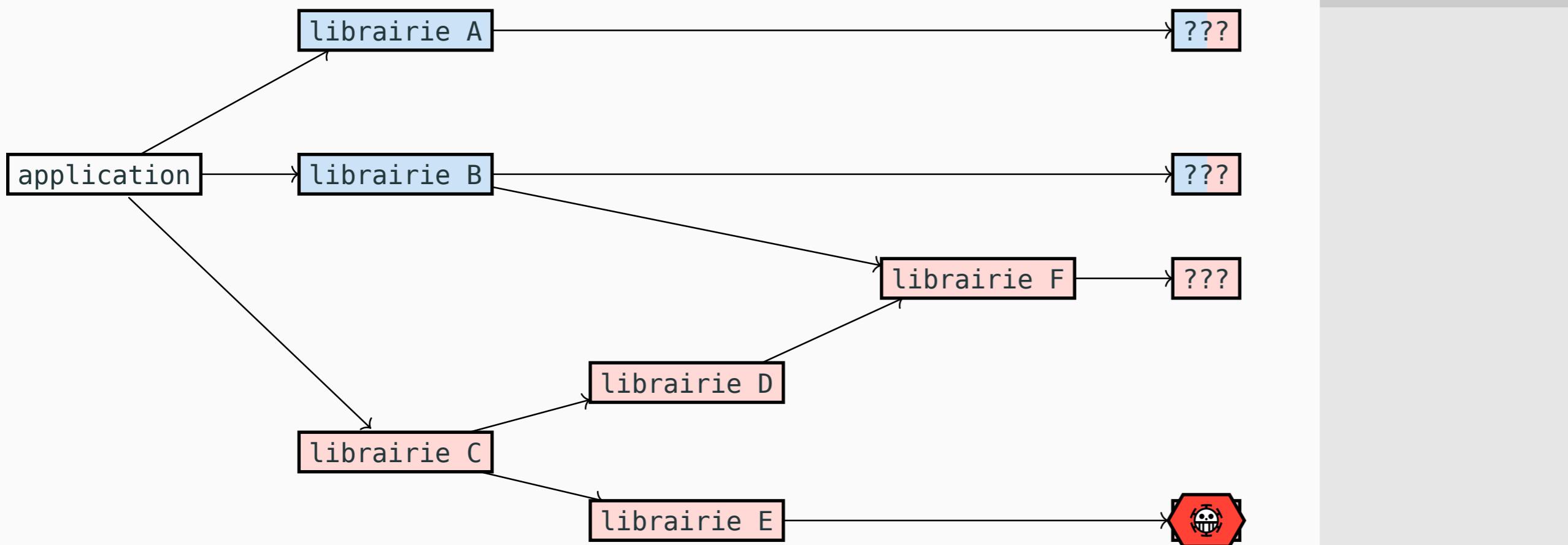


4.1.1.0.2 Sur le papier II, le retour

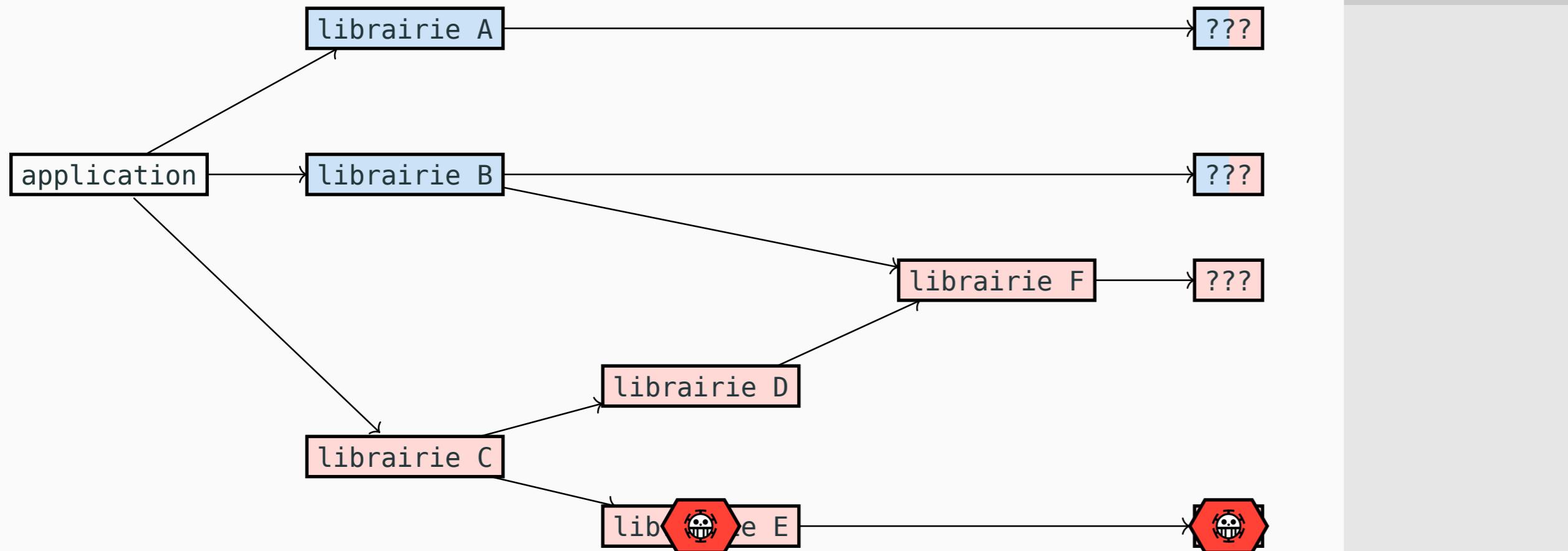
– 4.1 Présentation



– 4.1 Présentation



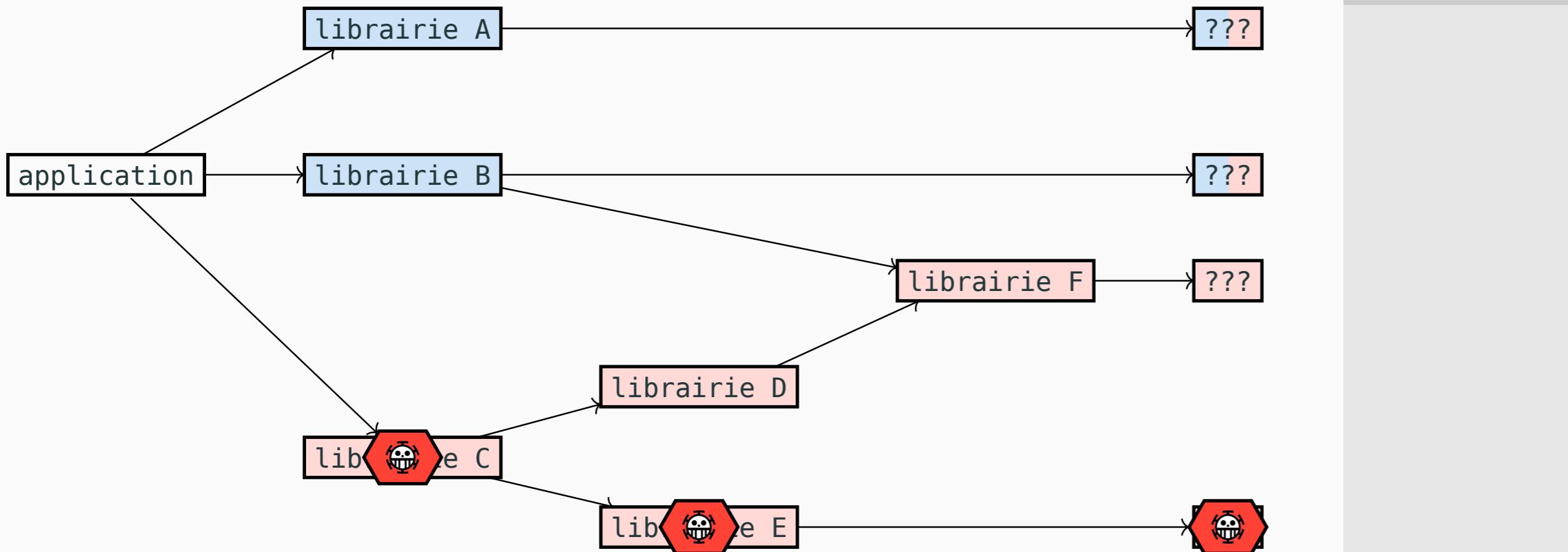
4.1 Présentation



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

– 4.1 Présentation

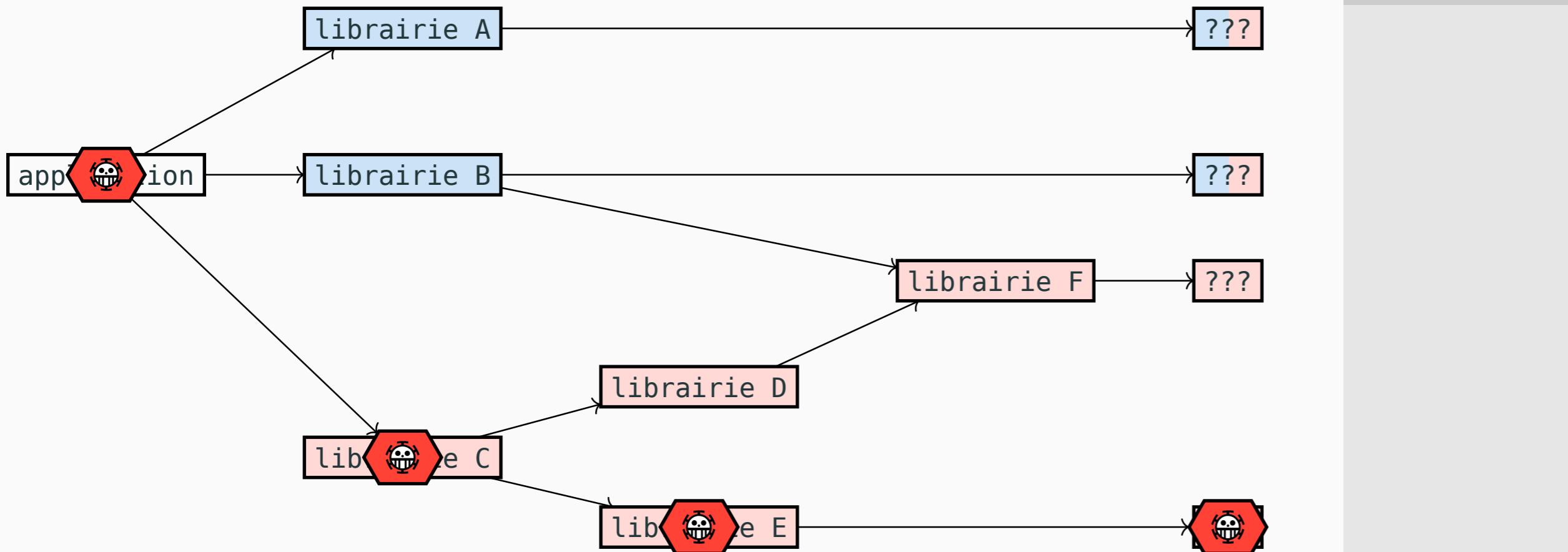
4.1 Présentation



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

– 4.1 Présentation

4.1 Présentation



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

– 4.1 Présentation

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).



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

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

– 4.1 Présentation

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).



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



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

– 4.1 Présentation

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.

– 4.1 Présentation

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

– 4.2 Nomenclature Logicielle (Software Bill of Materials - SBoM)

Nomenclature Inventaire hiérarchisé 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.

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

Deux spécifications majeures : [SPDX](#) (Linux Foundation) [12], [CycloneDX](#) (OWASP) [13].

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

4.2.0.0.1 CycloneDX



Figure 1: Modèle de nomenclature dans la spécification CycloneDX [13]

– 4.2 Nomenclature Logicielle (Software Bill of Materials - SBoM)

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.

4.2.0.0.2 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 [13] : github.com/CycloneDX/specification/blob/1.6/schema/bom-1.6.schema.json ;
- Exemples d'utilisation : github.com/CycloneDX/bom-examples.

– 4.2 Nomenclature Logicielle (Software Bill of Materials - SBoM)

4.2 Nomenclature Logicielle (Software Bill of Materials - SBoM)

```
cat $CYCLONEDX_SPECIFICATION/share/schema/bom-1.6.schema.json \
| jq -r '.definitions.vulnerability.properties
| to_entries[]
| .key as $k | select(any(([["advisories", "affects", "cves", "description", "id",
"source", "ratings"] | index($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
cves	List of Common Weaknesses Enumerations (CWEs) codes that describes this vulnerability.
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.
...	...

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

– 4.2 Nomenclature Logicielle (Software Bill of Materials - SBoM)

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

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.

4.3.1.0.1 Démo



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

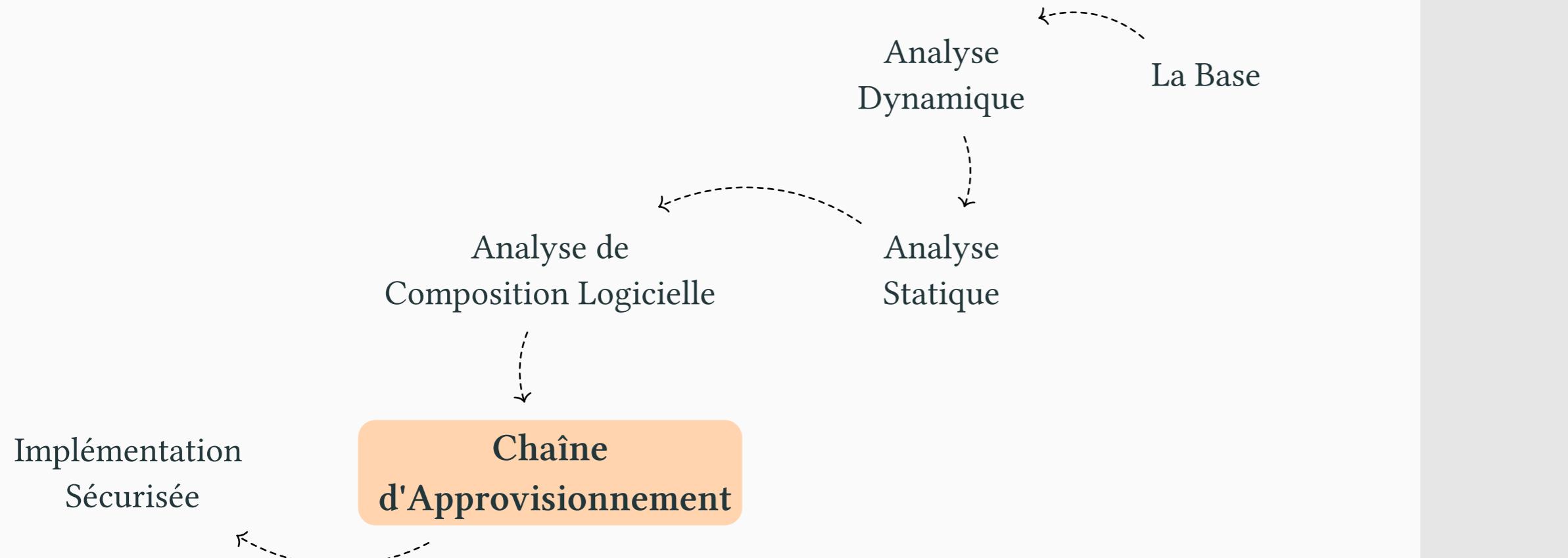
– 4.3 OWASP dep-scan

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 Chaîne d'Approvisionnement

5 Chaîne d'Approvisionnement



5.1 Présentation

5.1.1 Menaces

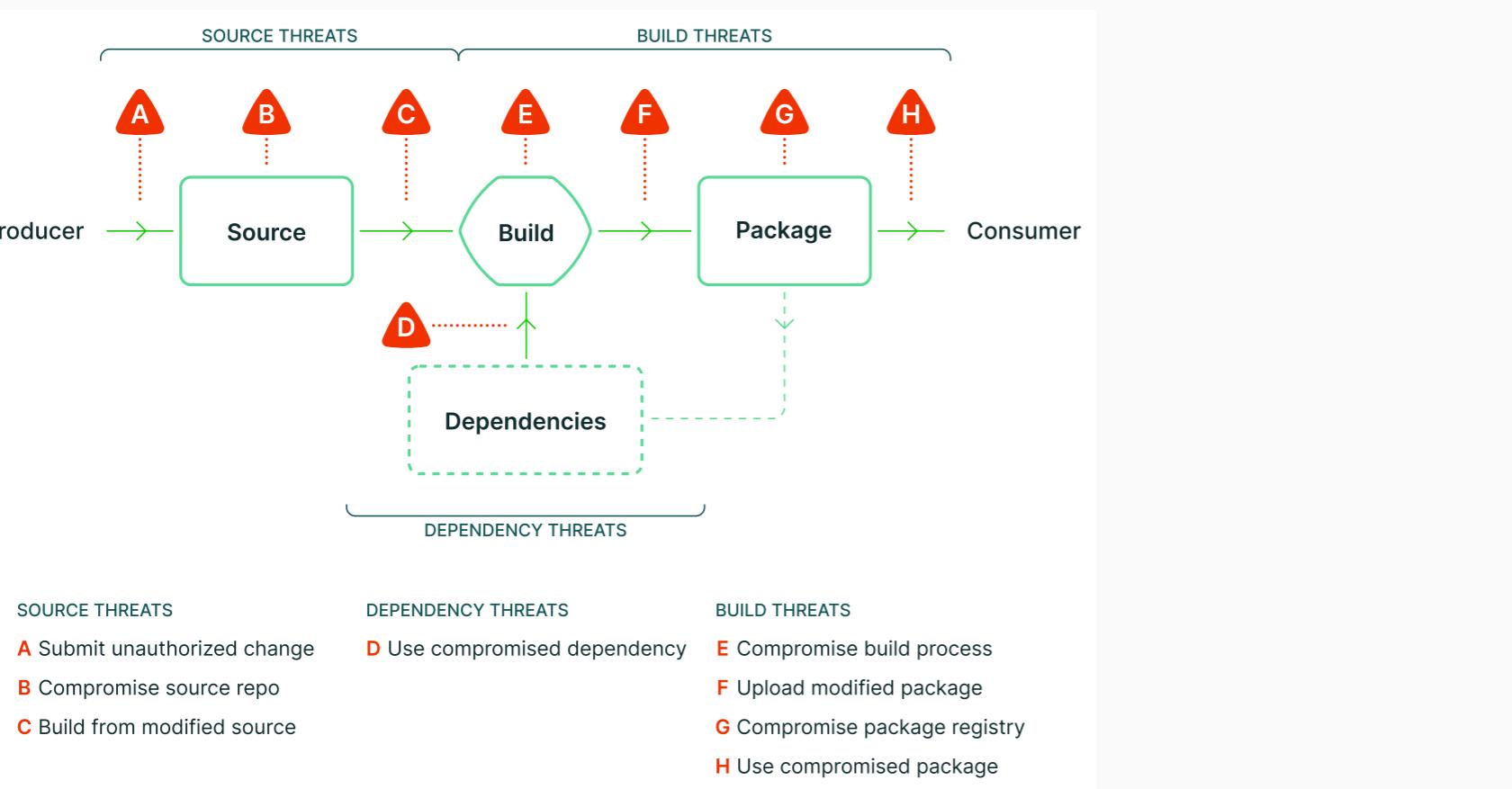


Figure 2: Les menaces sur la chaîne d'approvisionnement selon SLSA [14].

5 Chaîne d'Approvisionnement

– 5.1 Présentation

5.1 Présentation

Les attaques de la chaîne d'approvisionnement (supply chain attack) abusent des faiblesses listées précédemment.

Exemples :

- slsa.dev/spec/v1.0/threats-overview
 - ▶ Solarwinds (2020) :
 - logiciel malveillant russe (SUNSPOT)
 - ajoutant une porte dérobée (SUNBURST)
 - en remplaçant des fichiers source à la compilation
 - d'un logiciel utilisé par Microsoft, FireEye, les Départements du Trésor, du Commerce, de la Sécurité Intérieure des États-Unis, etc.¹
 - ▶ ...

¹C.f. <https://www.crowdstrike.com/en-us/blog/sunspot-malware-technical-analysis/>, <https://www.reuters.com/article/us-global-cyber-usa-dhs-idUSKBN28O2LY/>.

5 Chaîne d'Approvisionnement

– 5.1 Présentation

5.1.1.0.1 Quid du matériel ?

Exemples :

- Attaque (terroriste¹) d'Israël sur les pagers et talkie-walkies utilisés par le Hezbollah au Liban (2024) ;
- Allégations :
 - Le producteur de co-processeurs Cavium aurait implémenté des portes dérobées dans les opérations cryptographiques de certains de ses produits² ;
 - Des puces produites en Chine pourraient contenir des portes dérobées³.

¹42 morts, dont 11 membres du Hezbollah, 3500 blessés. C.f. <https://www.theguardian.com/world/2024/sep/20/we-are-isolated-tired-scared-pager-attack-lebanon-in-shock>, <https://abcnews.go.com/International/hezbollah-vows-reckoning-after-thousands-lebanon-injured-exploding/story?id=113798347>.

²C.f. <https://www.computerweekly.com/news/366552520/New-revelations-from-the-Snowden-archive-surface>.

³C.f. <https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies>, <https://www.bloomberg.com/features/2021-supermicro/>.

– 5.1 Présentation

5.1.1.0.2 Conclusion

Quel niveau de confiance accorde-t-on

- aux producteurices des composants et du matériel que l'on utilise ?
- à la façon dont les dépendances sont produites, et distribuées ?

– 5.1 Présentation

Producteurice interne ou tierc.e.s.

5.2.0.0.1 Une sauce, une danse, mais aussi ?

SLSA is a specification for describing and incrementally improving supply chain security, established by industry consensus. It is organized into a series of levels that describe increasing security guarantees.

– <https://slsa.dev/spec/v1.0/> [14]

provenance It's the verifiable information about software artifacts describing where, when and how something was produced.

– <https://slsa.dev/spec/v1.0/provenance> [14]

– 5.2 Supply chain Levels for Software Artifacts (SLSA)

5.2.0.0.2 Axes

SLSA levels are split into tracks. Each track has its own set of levels that measure a particular aspect of supply chain security.

— <https://slsa.dev/spec/v1.0/levels> [14]

À ce jour, seul l'axe “Build” à 4 niveaux :

0. Aucune exigence.
1. Le document de provenance décrit comment l'artefact est produit.
2. Le document de provenance est produit et signé par une plateforme dédiée à la fabrication de l'artefact.
3. La plateforme de production de l'artefact est bien sécurisée.

– 5.2 Supply chain Levels for Software Artifacts (SLSA)

La spéc est pas traduite en français... La traduction en “axe” est personnelle, c'est “track” dans le texte original.

5.2.0.0.3 Utilisation

- Générer des documents de provenance dans vos outils d'intégration continue¹ ;
- Publier les artéfacts avec leur document de provenance associé
 - ▶ Par exemple : dans le registre de paquets JavaScript/TypeScript npm², des registres de paquets Python³, ou des registres pour images de conteneurs⁴, etc.
- Vérifier les artéfacts importés avec les documents de provenance associés⁵.

¹Par exemple à l'aide de : <https://github.com/slsa-framework/slsa-actions-template>, <https://github.com/slsa-framework/slsa-jenkins-generator>, ou <https://github.com/in-toto/attestation>.

²Depuis Avril 2023, c.f. <https://github.blog/security/supply-chain-security/introducing-npm-package-provenance/>.

³Respectant <https://peps.python.org/pep-0740/#provenance-objects>.

⁴Provenance stockée comme un manifeste avec l'image, c.f. <https://docs.docker.com/build/metadata/attestations/attestation-storage/>.

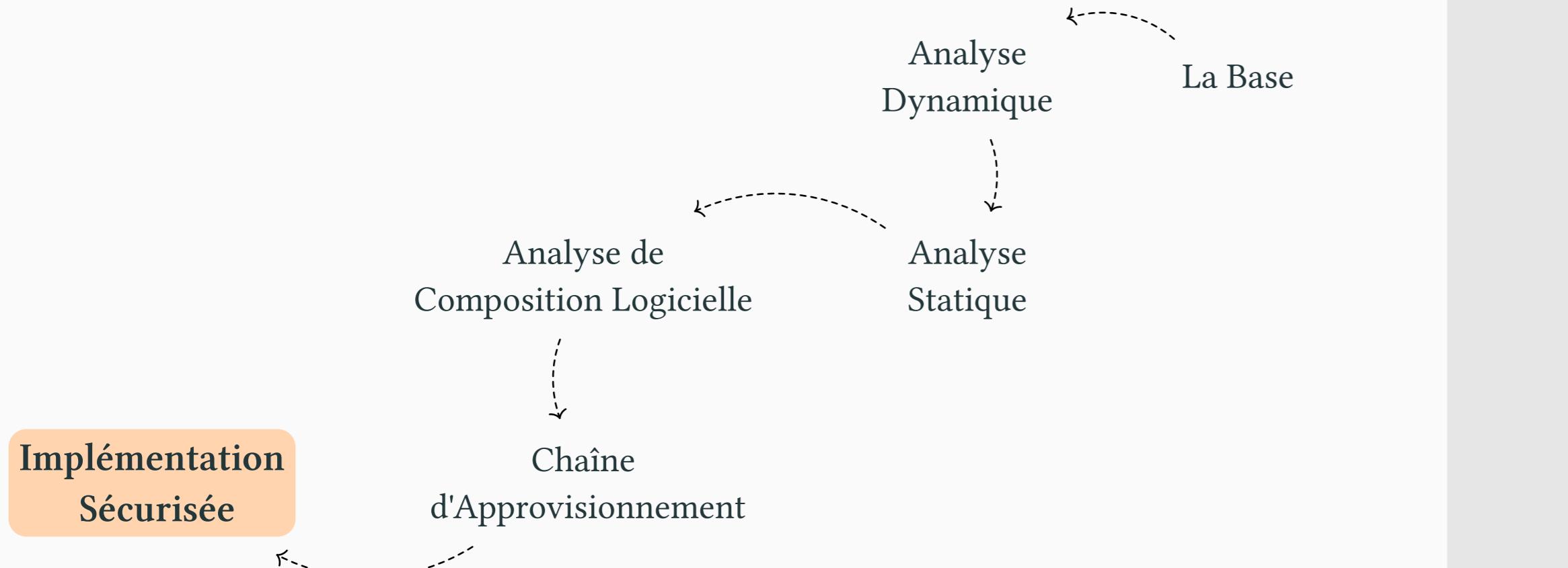
⁵Avec <https://github.com/slsa-framework/slsa-verifier>, <https://github.com/in-toto/attestation-verifier>, ou autre...

– 5.2 Supply chain Levels for Software Artifacts (SLSA)

Je n'ai pas trouvé d'informations au sujet de l'utilisation de SLSA, en particulier la publication de documents de provenance, par les gestionnaires de dépôts de paquets utilisés par les grandes distributions Linux (e.g. apt par Canonical ou Debian, yum par RedHat, etc.).

6 Implémentation Sécurisée

6 Implémentation Sécurisée



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

– 6.1 Principes de Développement Sécurisé

C'est une approche d'extrême gauche.

6.1.0.0.1 Quelques principes

Séparation des tâches Partager les responsabilités ; Découper une grosse fonctionnalité en sous-unités, qui requièrent une coordination pour être exécutée.

Moindre privilège Donner uniquement les accès aux ressources nécessaires à l'exécution d'une tâche.

Défense en Profondeur Stratégie d'accumulation des contrôles de sécurité, à différents niveaux.

Zero trust Vérifier tout ce qui provient de, ou qui est influencé par, l'extérieur du système.

Design ouvert Éviter la “sécurité par l'obscurité” ; La connaissance du design d'un système ne doit pas permettre son exploitation.

Sécurité par défaut Les composants ou états non sécurisés du système doivent être difficile (voire impossible) à atteindre.

– 6.1 Principes de Développement Sécurisé

- Zero trust ~= Aucune confiance.
- Design ouvert ~= Principe de Kerckhoffs.
- Sécurité par défaut, exemples :
 - Une API pour faire du chiffrement devrait utiliser des méthodes sécurisées (à ce jour), et rendre difficile l'utilisation de méthodes obsolètes (si besoin pour des raisons de comptabilité).
 - Organiser la représentation des données de façon à rendre impossible la manipulation de valeurs non sécurisées (c.f. [“Making Impossible States Impossible” by Richard Feldman](#), [“Parse, don’t validate”](#)).

6.1 Principes de Développement Sécurisé

6.1.0.0.2 Quelques ressources pour se former, s'entraîner

6.1.0.0.2.1 OWASP [15]

- <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 3: 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.

6 Implémentation Sécurisée

– 6.1 Principes de Développement Sécurisé

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.

6.1.0.1 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 !

6.1.0.2 Autres

- [tl;dr sec](#) – Infolettre hebdomadaire sur des outils, des pratiques, des retours d'expérience.

6.2 Modèle de Menaces (Threat Modeling)

Motiver et éclairer des décisions sur le design, l'implémentation, les tests, le déploiement, etc.
Reconnaitre 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 [1]

6.2.1 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.

6 Implémentation Sécurisée

– 6.2 Modèle de Menaces (Threat Modeling)

6.2 Modèle de Menaces (Threat Modeling)

6.2.2 Diagramme de flux de données (Data Flow Diagram)

Représenter les acteurs, les services et applications, les stockages de données, et les périmètres de confiance du système.

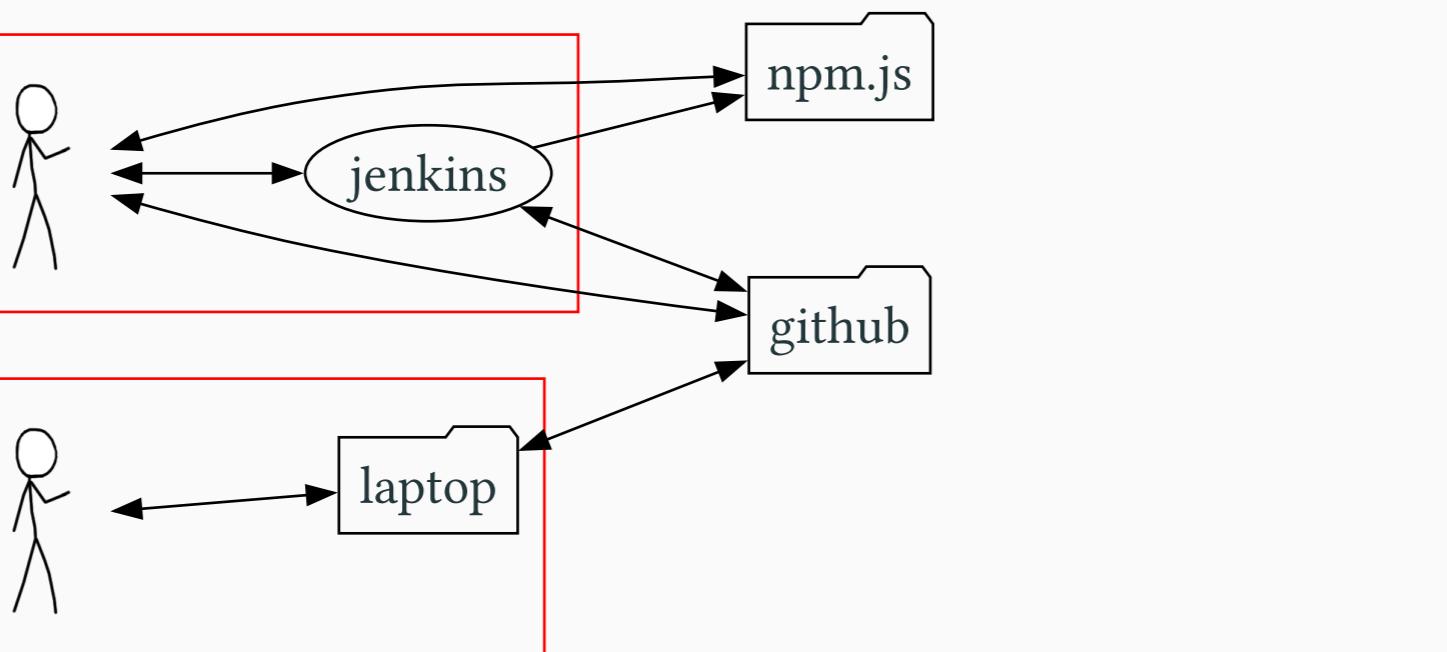


Figure 4: Un exemple de diagramme de flux de données.

6 Implémentation Sécurisée

– 6.2 Modèle de Menaces (Threat Modeling)

6.2.3 Identifier des menaces

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

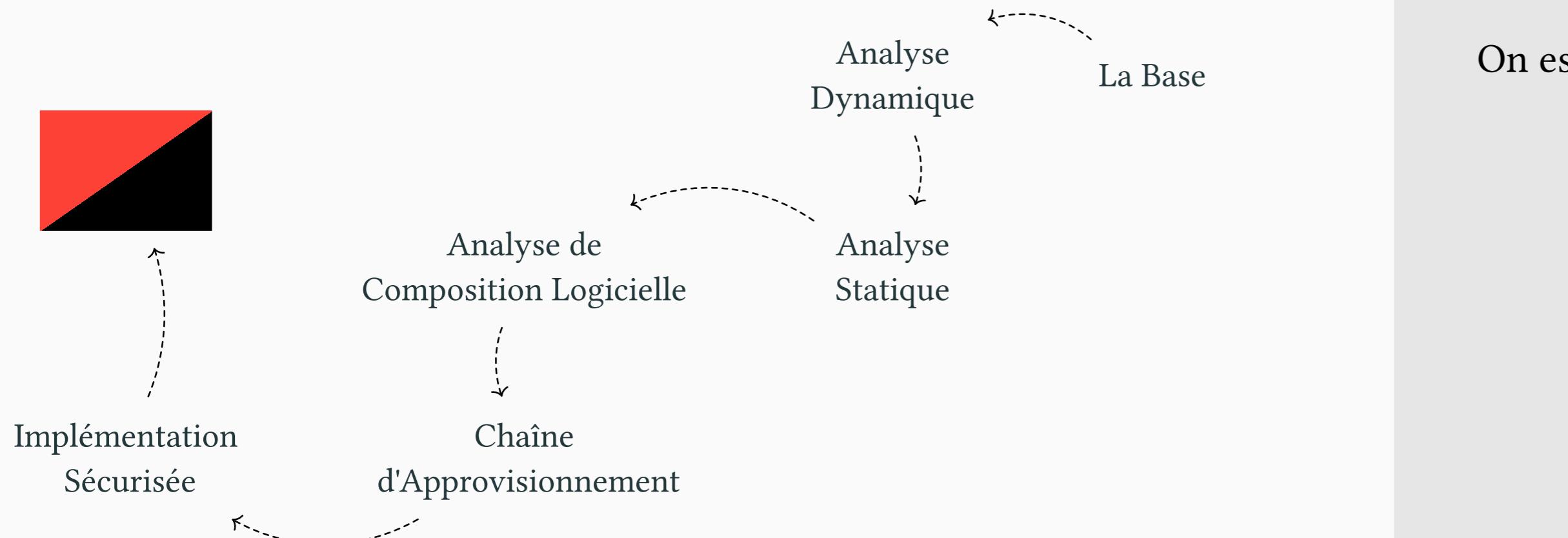


Figure 5: Vous, quand vous voyez un truc qui cloche.

- 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.

– 6.2 Modèle de Menaces (Threat Modeling)

7 Conclusion



—

On est arrivés ? Au bout de la présentation.

- A journey of understanding over a security or privacy snapshot.
- Continuous refinement over a single delivery.

— Threat Modeling Manifesto [1]

- Responding to change over following a plan

— Agile Manifesto [16]

—

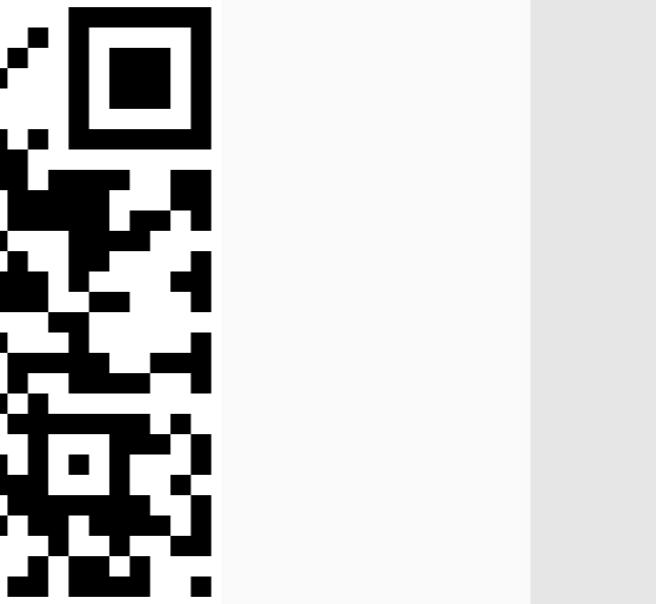
Rappel : la boucle DevSecOps est un continuum ; Les étapes précédentes de la boucle DevSecOps peuvent/doivent nourrir notre vision et compréhension du système, et guider nos prochains pas vers un ensemble plus sécurisé.

On veut être capable de réagir aux changements, et de conserver la sécurité de nos utilisateurices, et de leurs données.
Parce que c'est notre responsabilité.

Questions ?



Feedback 🍖



Merci de m'avoir écouté.

Est-ce que vous avez des questions?

8 Crédits

—

8 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.

—

Bibliography

- [1] "Threat Modeling Manifesto." Accessed: Oct. 01, 2024. [Online]. Available: <https://www.threatmodelingmanifesto.org/>
- [2] 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>
- [3] "Zed Attack Proxy." Accessed: Oct. 01, 2024. [Online]. Available: <https://www.zaproxy.org/>
- [4] "American Fuzzy Lop." Accessed: Oct. 01, 2024. [Online]. Available: <https://lcamtuf.coredump.cx/afl/>
- [5] 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.
- [6] Y. Shoshtaishvili *et al.*, "SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis," in *IEEE Symposium on Security and Privacy*, 2016.
- [7] N. Stephens *et al.*, "Driller: Augmenting fuzzing through selective symbolic execution," in *NDSS*, 2016, pp. 1–16.
- [8] V. A. Alfred, S. L. Monica, and D. U. Jeffrey, *Compilers principles, techniques & tools*. Pearson Education, 2007.
- [9] S. Muchnick, *Advanced compiler design implementation*. Morgan Kaufmann, 1997.
- [10] "SemGrep." Accessed: Oct. 01, 2024. [Online]. Available: <https://semgrep.dev/>
- [11] "CodeQL." Accessed: Oct. 01, 2024. [Online]. Available: <https://codeql.github.com/>
- [12] "System Package Data Exchange (SPDX®)." Accessed: Oct. 01, 2024. [Online]. Available: <https://spdx.dev/>
- [13] "CycloneDX Specification Overview." Accessed: Oct. 01, 2024. [Online]. Available: <https://cyclonedx.org/specification/overview/>
- [14] "Supply-chain Levels for Software Artifacts (SLSA)." Accessed: Oct. 01, 2024. [Online]. Available: <https://slsa.dev/>
- [15] "Open Worldwide Application Security Project (OWASP)." Accessed: Oct. 01, 2024. [Online]. Available: <https://owasp.org/>
- [16] "Agile Manifesto." Accessed: Oct. 01, 2024. [Online]. Available: <https://agilemanifesto.org/>