Quand l'IA confond syntaxe et sémantique

Exemples adversariaux fonctionnels contre les modèles de similarité de binaires

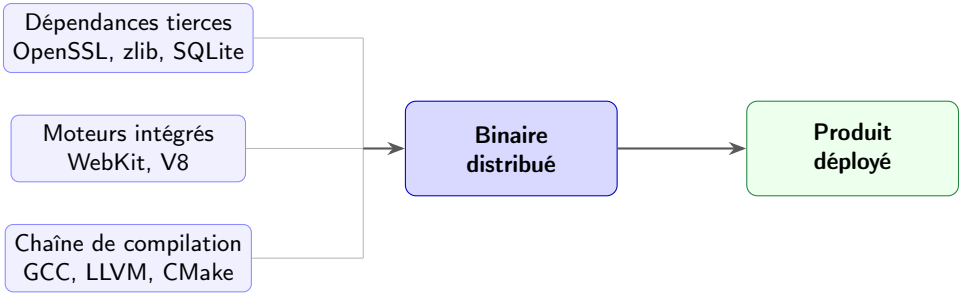
Victor Matrat
Université de Lorraine, CNRS, LORIA – Projet SECUBIC

Campus Cyber — 24 septembre 2026

SECUBIC

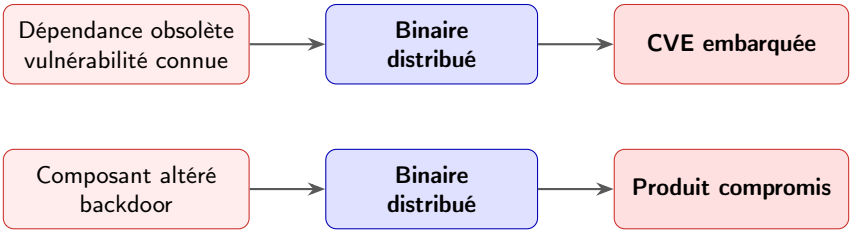
Travail conjoint avec Gabriel Sauger, Jean-Yves Marion, Sazzadur Rahaman, Nicolas Bailluet, Vincent Tourneur et Muaz Ali

Un produit logiciel est une chaîne d'approvisionnement



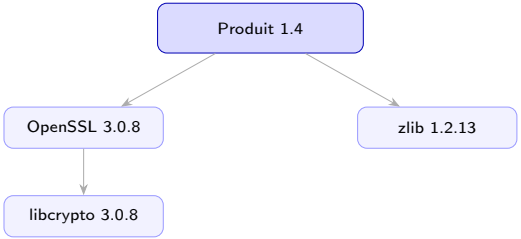
Le produit final agrège du code maintenu, versionné et compilé par plusieurs acteurs.

Les risques se propagent jusqu'au produit livré



Une faiblesse ou une altération en amont peut affecter tous les produits qui embarquent la brique concernée.

Software Bill of Materials (SBOM)



composants, versions,
relations de dépendance

Une nomenclature exploitable

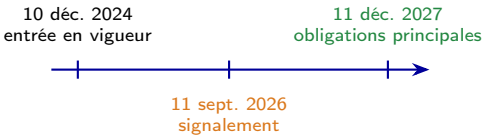
- suivre les composants et leurs versions ;
- relier un produit aux vulnérabilités connues ;
- prioriser les correctifs et répondre aux incidents.

Cyber Resilience Act (CRA) : exigences sur tout le cycle de vie

Pour les produits comportant des éléments numériques

Le CRA introduit des exigences de cybersécurité sur tout le cycle de vie :

- gestion et correction des vulnérabilités ;
- documentation des composants ;
- SBOM dans un format couramment utilisé et lisible par machine, couvrant au moins les dépendances de premier niveau.



Le jour de cette présentation, les obligations de signalement sont déjà applicables.

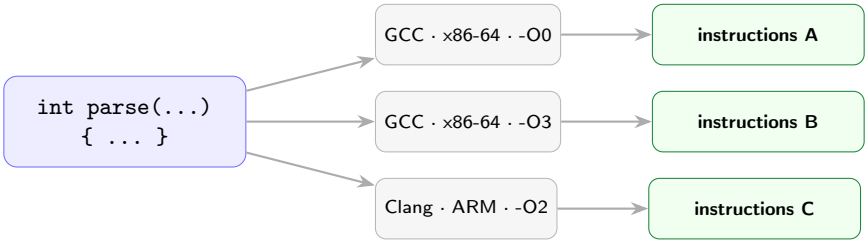
. Règlement (UE) 2024/2847, annexe I, partie II ; calendrier de la Commission européenne. Union européenne 2024 ; European Commission 2026

Du SBOM déclaré au logiciel réellement livré

Une question concrète

Le SBOM déclare la présence d'**OpenSSL 3.0.8**. Comment la confirmer directement à partir du binaire livré ?

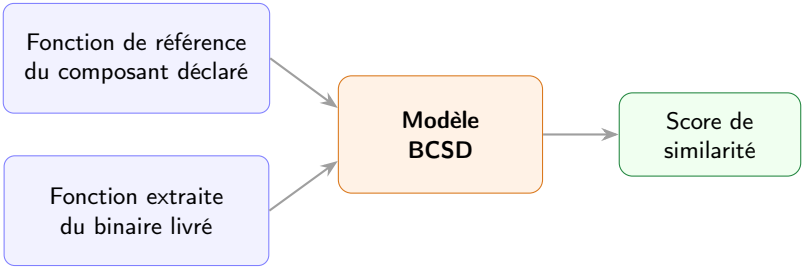
Une même source produit de nombreux binaires



Le défi

Être invariant aux transformations bénignes de compilation, tout en restant sensible aux changements de comportement.

Binary Code Similarity Detection (BCSD) : retrouver le code déclaré dans le binaire



La BCSD estime si une fonction du produit correspond à une fonction de référence, même lorsque la compilation en a modifié la forme.

Trois familles de modèles BCSD

Les familles diffèrent par **la façon de représenter une fonction binaire**.

Séquence — « le code comme du texte »

Les instructions deviennent des **tokens**, traités avec des méthodes inspirées du NLP. *Asm2Vec, jTrans*

Graphe — la structure comme signal

Le modèle apprend sur le **graphe de flot de contrôle** et les attributs de ses nœuds. *GMN*

Hybride — au-delà des instructions

Le code est traduit dans une **représentation intermédiaire**, enrichie par le flot de données et les effets. *HermesSim*

Ces modèles n'observent jamais directement la sémantique : ils l'infèrent à partir de caractéristiques syntaxiques.

. Exemples de modèles : Asm2Vec Ding et al. 2019, GMN Li et al. 2019, jTrans Wang et al. 2022, HermesSim He et al. 2024.

L'écart entre syntaxe et sémantique devient une surface d'attaque

Ce que le modèle voit

opcodes · instructions · CFG
artefacts de compilation

$\neq$

Ce qu'il doit décider

comportement · effets
équivalence fonctionnelle

Hypothèse

Si deux fonctions aux comportements différents peuvent produire des caractéristiques syntaxiques proches, alors un attaquant peut manipuler ces caractéristiques pour fabriquer une fausse similarité.

Un exemple adverse exploite volontairement cet écart

Une entrée construite pour provoquer une perception erronée

L'entrée reste valide et fonctionnelle, mais sa représentation conduit le modèle à prendre une décision erronée.

Syntaxe manipulée
la représentation perçue est modifiée

Sémantique préservée
le payload conserve son comportement

L'attaquant modifie ce que le modèle voit, sans modifier ce que le programme fait.

Modèle de menace : une attaque supply-chain sans accès au modèle

Objectif

Transformer un payload pour que le classificateur l'identifie comme une **cible bénigne choisie**, tout en préservant son comportement exécutable.

L'attaquant dispose de

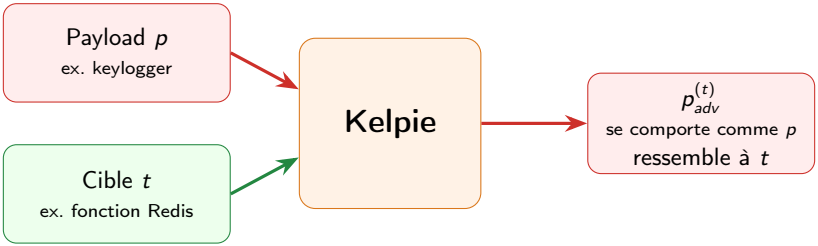
Le **code source du payload**, une cible bénigne de son choix et la capacité de recompiler puis de tester le code.

Du côté du défenseur

Le modèle, ses caractéristiques et ses données d'entraînement restent privés. Aucune requête, aucun score ni label ne sont exposés ; seul le binaire final est analysé.

L'attaque est donc **black-box** et **zero-query** : elle n'est guidée par aucun retour du classificateur.

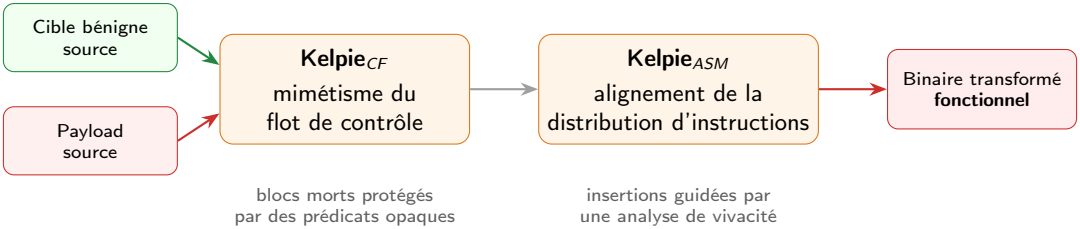
Kelpie exploite l'écart en imitant la syntaxe d'une cible bénigne



Un keylogger reste un keylogger, mais le modèle le rapproche d'une fonction Redis.

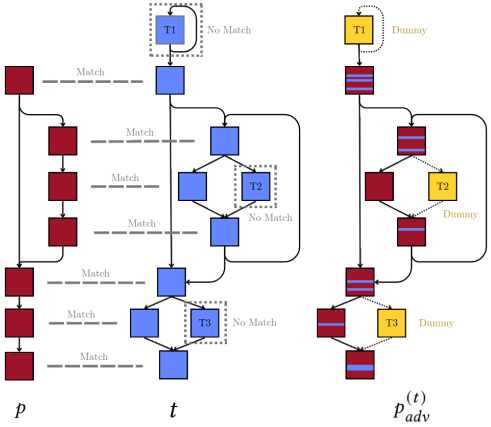
. G. Sauger, J.-Y. Marion, S. Rahaman, V. Matrat, V. Tourneur et M. Ali, « Attacking the First-Principle: A Black-Box Query-Free Targeted Mimicry Attack on Binary Function Classifiers », *IEEE EuroS&P*, 2026.

Kelpie modifie la forme du code, pas son comportement



Les deux transformations modifient les **indices syntaxiques** sans modifier le comportement exécuté.

Kelpie rapproche les graphes avec des blocs morts



Lecture du schéma

- **Rouge** : basic blocks de p ;
- **Bleu** : basic blocks de t ;
- **Jaune** : blocs morts ajoutés.

Les basic blocks sans correspondance guident la transformation du payload, qui reste **fonctionnel**.

. Adapté de la Figure 1 de l'article Kelpie. Sauger et al. 2026

Après l'attaque, aucun modèle ne reste fiable

8 modèles

Asm2Vec, CLAP, GGSNN, GMN
HermesSim, jTrans, Trex, Zeek

Moins fiables que le hasard

AUC $< 0{,}50$ pour tous les classificateurs

Même les représentations dites « sémantiques » restent vulnérables

Même le meilleur modèle devient moins fiable qu'une décision aléatoire (AUC de 0,42). En comptant les évasions non ciblées, plus d'une transformation sur deux trompe le modèle.

. Résultats de classification, Table 2 et §4.3 de l'article Kelpie. Sauger et al. 2026

Cette faiblesse atteint des tâches de sécurité concrètes

Masquer une vulnérabilité

vulnérable ⟶ « corrigée »

- 60 paires issues de FFmpeg, Tcpdump et OpenSSL ;
- **25–35 %** de mimétisme ciblé ;
- VulSeeker également affecté.

Dissimuler un payload

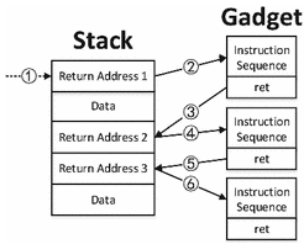
keylogger / wiper ⟶ bénin

- keylogger et wiper insérés dans Masscan et Redis ;
- les deux payloads restent fonctionnels ;
- dissimulés dans **plus de 8 cas sur 10** ;
- PSS échoue à identifier les deux payloads.

. Études de cas, §5–6 de l'article Kelpie. Sauger et al. 2026

Peut-on exploiter ce même écart en plaçant la **sémantique** dans le **flot de données**, hors des caractéristiques syntaxiques observées par le modèle ?

Ropie place le comportement hors du flot de contrôle visible



Une chaîne ROP enchaîne des fragments de code existants, ou *gadgets*.

L'intuition

Le calcul est piloté par des **adresses et des valeurs sur la pile**. Il peut donc ne pas apparaître comme un nouveau chemin dans le CFG reconstruit par l'analyse statique.

Ropie

- injecte une chaîne ROP dans un ELF x86-64 ;
- réutilise les gadgets déjà présents ;
- n'ajoute que les gadgets manquants ;
- préserve le fonctionnement normal du binaire hôte.

. Principe du ROP Shacham 2007 ; Roemer et al. 2012. Ropie : travaux en cours.

Ropie reste dans la variabilité naturelle de compilation

Ce que l'on mesure

On compare l'écart introduit par Ropie à celui qui sépare naturellement deux versions du même code compilées avec des niveaux d'optimisation différents.

Si l'écart n'est pas plus grand, l'injection se confond avec la compilation.

4 modèles

Asm2Vec, GGSNN, GMN et jTrans

84–98 %

des fonctions injectées aussi proches que des recompilations

76–93 %

des binaires injectés dans la variabilité inter-compilation

. Résultats préliminaires : expériences en cours, non publiées.

Deux attaques différentes, un même écart entre syntaxe et sémantique

Kelpie

Modifie la **syntaxe** observée par le modèle — CFG, basic blocks et opcodes — tout en préservant la sémantique du payload.

Ropie

Place la **sémantique** du payload dans le flot de données, hors de l'abstraction syntaxique observée par le modèle.

Dans les deux cas, le modèle suit la **syntaxe** sans capturer la **sémantique exécutée**.

Conclusion

Les représentations actuelles du code binaire ne sont pas robustes face à des transformations adversariales.

Conclusion

Limite actuelle

Les modèles actuels de BCSD reconnaissent des similitudes entre compilations, mais restent trop dépendants du CFG, des opcodes et d'autres indices syntaxiques. À eux seuls, ils ne permettent donc pas de vérifier de manière fiable qu'un binaire livré correspond aux **déclarations d'un SBOM**.

Perspectives

Développer des représentations qui capturent plus fidèlement le **flot de données**, les effets du code et la **sémantique exécutée**, tout en restant invariantes aux transformations bénignes de compilation et robustes face aux transformations adversariales.

Merci pour votre attention !

Avez vous des questions ?

Article Kelpie sur HAL

Université de Lorraine, CNRS, LORIA

Gabriel Sauger — gabriel.sauger@loria.fr
Jean-Yves Marion — jean-yves.marion@loria.fr
Victor Matrat — victor.matrat@loria.fr
Vincent Tourneur — vincent.tourneur@loria.fr

Nicolas Bailluet — nicolas.bailluet@ens-rennes.fr

University of Arizona

Sazzadur Rahaman
sazz@cs.arizona.edu
Muaz Ali
muaz@arizona.edu

cnrs Loria SECUBIC UNIVERSITÉ DE LORRAINE