C4PTION : Pourquoi caractériser les auteurs de portions de code ?

C4PTION : Why Characterise the Authors of code PorTIONs?

Pierre-Marie Satre[1], Olivier Gesny[1], Robin De Saint Jores[1], Tudy Gourmelen[1], Christophe Genevey-metat[1], Dorian Bachelot[1], Loan Veyer[1], Maximilien Chaux[1] and Pierre Delesques[1]

[1]*Silicom, 15 Boulevard des Chênes, Guyancourt, 78280, France*

Abstract

The supply chain's attacks as the one suffered Solwarinds are full of consequences. Today's companies can no longer ignore the risks linked to such practices. However, how to successfully prevent it ? We already use security protocoles for our systems of software versions management such as git, but it does not seem to be sufficient. The DevSecOps automating software security analysis in a continuous integration and deployment chain constitutes a solution to answer these issues. It is within this context that we developed the solution why *Characterise the Authors of code PorTIONs*? (C4PTION). It completes the arsenal of source code analysis tools with a detection of injected code by unauthorized authors (humans or bots). The C4PTION tool is based on innovating AI techniques that help learn syntactic, lexical and behavioral habits of developers. Integrated in a CI/CD such as Gitlab, C4PTION transmits qualified alerts to project responsibles/CISO (Chief Information Security Officer) : level of developer's usurpation risk, level of confidence in the decision and an explicable report. In this article we show that combined source code and git metadata analysis by trained AI models as well as the adaptable aspect of C4PTION present a tremendous number of assets (especially the one increasing the elements of the confusion matrix by 15%) to prevent any cyber attack in a software development's supply chain.

Keywords

Identity and access management, zero trust, artificial intelligence, git metadata, source code, automation

Abstract

Les attaques au sein de la supply chain comme celle subie par Solarwinds sont lourdes de conséquences. Les entreprises d'aujourd'hui ne peuvent se permettre d'ignorer les risques liés à de telles pratiques. Et cependant, comment réussir à s'en prémunir ? Nous disposons déjà de protocoles sécurisés pour nos systèmes de gestion de versions logicielles tels que git, mais ceux-ci semblent ne pas suffire. Le DevSecOps, automatisant l'analyse de la sécurité logicielle dans une chaîne d'intégration et de déploiement continu (CI/CD), constitue une solution permettant de répondre à ces enjeux. C'est dans ce cadre que nous avons développé la solution

C&ESAR'22: Computer Electronics Security Application Rendezvous, November 15-16, 2022, Rennes, France

pmsatre@silicom.fr (P. Satre); ogesny@silicom.fr (O. Gesny); rdesaintjores@silicom.fr (R. D. S. Jores); tgourmelen@silicom.fr (T. Gourmelen); cgeneveymetat@silicom.fr (C. Genevey-metat); dbachelot@silicom.fr (D. Bachelot); lveyer@silicom.fr (L. Veyer); mchaux@silicom.fr (M. Chaux); pdelesques@silicom.fr (P. Delesques)

why *Characterise the Authors of code PorTIONs*? (C4PTION). Elle complète l'arsenal d'outils d'analyse de code source avec une détection de code injecté par des auteurs non autorisés (humains ou bots). L'outil C4PTION s'appuie sur des techniques d'Intelligence Artificielle (IA) innovantes pour apprendre les habitudes syntaxiques, lexicales et comportementales des développeurs. Intégré dans une CI/CD telle que gitlab, C4PTION transmet aux responsables projets et/ou aux responsable de la sécurité des systèmes d'information (RSSI) des alertes qualifiées : niveau de risque d'usurpation de développeur, niveau de confiance dans la décision, ainsi qu'un rapport d'explicabilité. Dans cet article, nous montrons que l'analyse combinée du code source et des git metadata par des modèles IA entraînés ainsi que le caractère adaptable de C4PTION, présentent de nombreux atouts (notamment, celui d'améliorer les éléments de la matrice de confusion de près de 15%) pour prévenir toute cyberattaque en amont de la supply chain de développement logiciel.

1. Introduction

Les cyberattaques par injection de backdoors dans le code source se sont accentuées ces dernières années à l'instar de ce qu'a pu subir l'éditeur Solarwinds au niveau de sa supply chain en 2020. Les logiciels infectés sont innocemment vendus en toute quiétude à des entreprises. Malheureusement, en dépit des systèmes de sécurités EDR et XDR, les backdoors sont mises à profit par les cyber attaquants, notamment à des fins d'espionnage, de destruction ou de corruption des données des entreprises acquéreuses. Lorsque le dommage est constaté, ces entreprises se retournent contre l'éditeur qui va subir des pertes financières considérables pour faire face à des attaques en justice. S'ensuivront : une perte de la confiance client, une perte de l'image de marque ainsi que d'indispensables investissements pour mettre à jour des process d'entreprise. Mais alors comment se prémunir de tels risques ? Nous disposons déjà de protocoles sécurisés pour nos systèmes de gestion de versions logicielles tels que git, mais ceux-ci semblent ne pas suffire. Le challenge est donc maintenant de remettre en question les auteurs responsables des commits de nos projets. Et ce par le biais d'un système qui pourrait s'apparenter à une extension d'authentification git, non triviale à leurrer (même en cas de dépôt git analysé en profondeur par un attaquant). Dans un premier temps nous verrons brièvement ce que peut proposer C4PTION. Ensuite, nous porterons notre intérêt sur le moteur d'IA SIlicom Versatile Artificial intelligence (SIVA) ainsi que sur son exploitation précise par C4PTION. Enfin nous présenterons nos résultats actuels ainsi que les objectifs moyen terme portés par ce projet.

2. Verrous scientifiques et techniques

Les verrous scientifiques et techniques soulevés par la caractérisation puis la détection de développeurs potentiellement malveillants via apprentissage machine sont variés :

- `Authentification des développeurs ou bots lors d'un push git` : L'authentification d'un auteur lorsqu'il pousse du code est inexistante sur git, faillible sur gitlab, et ce notamment pour l'accès aux tokens lors des upgrades de dépendances automatiques [1]

- `Décision mono modèle/donnée` : Une décision provenant d'un seul modèle ou d'un seul type de données est très susceptible de souffrir de biais de données et risque de fournir des résultats insuffisants en usage réel [2]
- `Manque d'explicabilité` : Une solution non explicable ne donne pas confiance à l'humain
- `Nouveaux développeurs en cours de projet` : Les modèles d'IA à nombre de classes fixes sont non exploitables pour ce besoin (ex : Deep Learning)

3. État de l'art

Il existe dans la littérature scientifique des algorithmes permettant d'attribuer un auteur à un code donné. Les approches varient : certains utilisent Deep Learning et data/gradient augmentation[3] afin de diminuer le taux de succès d'attaque obfusquant les auteurs de code. D'autres ont simplement pour objectif d'exploiter des technologies de type TF-IDF[4](*term frequency-inverse document frequency*) pour attribuer du code d'applications Android à leur auteur. Enfin d'autres contribuent en améliorant les performances de classification et en accélérant les vitesses d'apprentissage[5]. Néanmoins, à notre connaissance, aucun outil ne permet d'authentifier le commit d'un développeur dans une chaîne d'intégration continue de type Gitlab CI. Seuls des algorithmes présentés comme essai expérimental se retrouvent dans ces papiers.

4. Solution proposée

4.1. Vue d'ensemble

Notre solution C4PTION trouve tout son sens dans un cadre d'attaque par supply chain. Elle introduit un nouveau moyen de satisfaire le zero trust et l'Identity and Access Management (IAM) via une surveillance du code produit par les développeurs dans git. C4PTION s'incorpore dans une chaîne d'intégration continue en classifiant les auteurs des commits et en détectant des anomalies comportementales d'auteurs. Chaque commit poussé est analysé par le système. Son rôle est de confirmer que le code commité correspond au style de codage, comme aux habitudes de commit de l'auteur responsable. Ainsi l'auteur peut être authentifié, ou l'imposteur démasqué, grâce à la comparaison entre les habitudes apprises et les habitudes de codage décelées. Pour une meilleure adhésion par les entreprises et équipes projets, C4PTION fournit des décisions explicables et fonctionne en mode Software As A Service (SaaS) ou avec le logiciel dans l'infrastructure réseau du client (on premises).

C4PTION s'appuie sur une collaboration multi-IA (classification et détection d'anormalité) pour prendre des décisions avec un niveau de confiance maximal.

4.2. SIVA

Le coeur de C4PTION est alimenté par plusieurs instances de notre IA SIVA, configurées pour la classification (mode supervisé) et la caractérisation de la normalité (mode

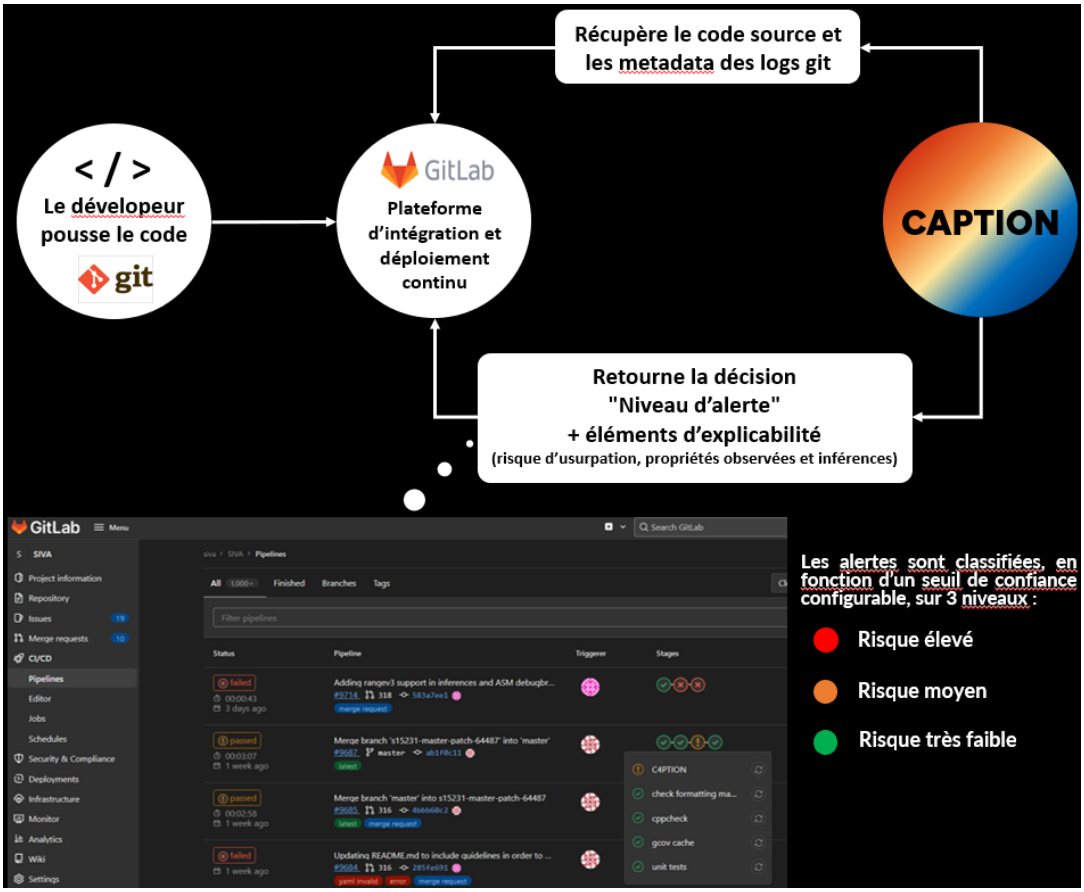

Figure 1: *Vue d'ensemble des interactions de C4PTION avec Gitlab*

non supervisé). SIVA se démarque par ses stratégies évolutionnistes faisant émerger des observations inédites en phase d'apprentissage, ainsi que par sa forte capacité de corrélation entre **observation** de l'environnement, **récompenses** et **décisions possibles**. Grâce à la combinaison de techniques d'apprentissage supervisé/non supervisé, de genetic programing (GP) et de mécanismes d'attention, elle construit un modèle. Sa capacité **d'adaptabilité à un nombre de classes variables**, et donc à de potentiels nouveaux auteurs, rend son utilisation tout à fait adaptée à un cadre de détection de code source injecté par un attaquant ayant usurpé l'identité d'un développeur.

4.2.1. Modèle

SIVA est un algorithme multi agents décliné des solutions TPG [6] et CBWAR [7]. Au regard de ces deux solutions, il conserve la méthode de parcours du graphe et sa nature évolutionniste pendant l'apprentissage, mais a fait largement évoluer deux éléments : l'instruction qui est désormais remplacée par un agent ; et le système de décision, qui ne s'appuie plus sur le noeud final atteint lors du parcours du graphe, mais prend désormais

en compte l'expérience des agents exécutés lors du parcours du graphe.

4.2.1.1. Les agents

Pour prendre une décision, les agents observateurs sont consultés, et donnent leur avis à SIVA sur la base de leur spécialité d'observation. Un agent de SIVA se définit par trois caractéristiques émergentes et corrélées entre elles : une **propriété observée**, les **décisions possibles** et les **récompenses espérées** pour chaque décision. Ainsi, pour les contextes d'apprentissage supervisé, un **agent** observe une propriété de l'environnement et crée ainsi des corrélations entre cette **propriété**, les **décisions (auteurs)** et les **récompenses espérées** (1 [auteur corrélé à l'observation] ou -1 [auteur non corrélé à l'observation]). Dans un contexte d'apprentissage non supervisé, un **agent** crée des corrélations entre la **propriété** (qui inclut les auteurs possibles), la **décision possible** (commit normal) et la **récompense espérée** (1 [commit normal]). En résumé, pour un même agent, les corrélations peuvent concerner toutes les combinaisons de décisions et de récompenses. Ainsi, un même agent a la capacité de donner son avis sur plusieurs décisions différentes, avec des espoirs de récompenses potentiellement différents. Le type d'un agent dépend du type de l'observation qu'il effectue : une suite de caractères bruts, un mot ou une suite de mots, un style de commentaire doxygen, un type d'indentation, etc. Dans le contexte C4PTION, le type d'un agent est souple et accorde une adaptabilité à divers langages de programmation. Le but étant qu'avec des types généralistes, l'apprentissage fasse émerger entre autres des observations propres à un langage précis. De cette manière, notre solution est adaptable à différents langages de programmation. Les types d'observation sont nombreux, et aisément extensibles au travers de futurs développements. Un agent ne conseille une décision que si son observation est avérée dans l'environnement.

4.2.1.2. Initialisation du modèle

Le modèle de SIVA est composé d'un graphe d'agents que vous retrouvez fig. 2 ainsi que d'une base d'agents experts. Les éléments du graphe sont la racine, le noeud, le groupement d'agents ainsi que les feuilles. La racine est un noeud qui est le point d'entrée d'une exécution du graphe. Le noeud est un point de passage possédant plusieurs groupements d'agents. Comme leur nom l'indique, les groupements d'agents sont de simples agglomérations d'agents qui pointent vers le noeud suivant ou vers une feuille. Enfin les feuilles représentent la fin d'exécution du graphe.

L'entraînement de SIVA débute avant toute chose par une étape d'initialisation. Celle-ci a pour but de créer les premières racines ainsi que leurs premiers groupements d'agents. Le nombre de racines, de groupements par racine et d'agents par groupement sont choisis aléatoirement entre des bornes qui sont des hyper paramètres. Pour chaque agent, un type d'observation est aléatoirement choisi. Une fois les racines et les agents définis, le graphe est prêt à être exécuté sur les premiers challenges. Au moment de la mise à jour des inférences des agents du graphe, tout nouvel agent est dupliqué dans la base d'agents experts. Tout agent devenant non causal est supprimé du graphe mais son jumeau est conservé dans la base d'agents experts qui contient des agents mémorisant des inférences

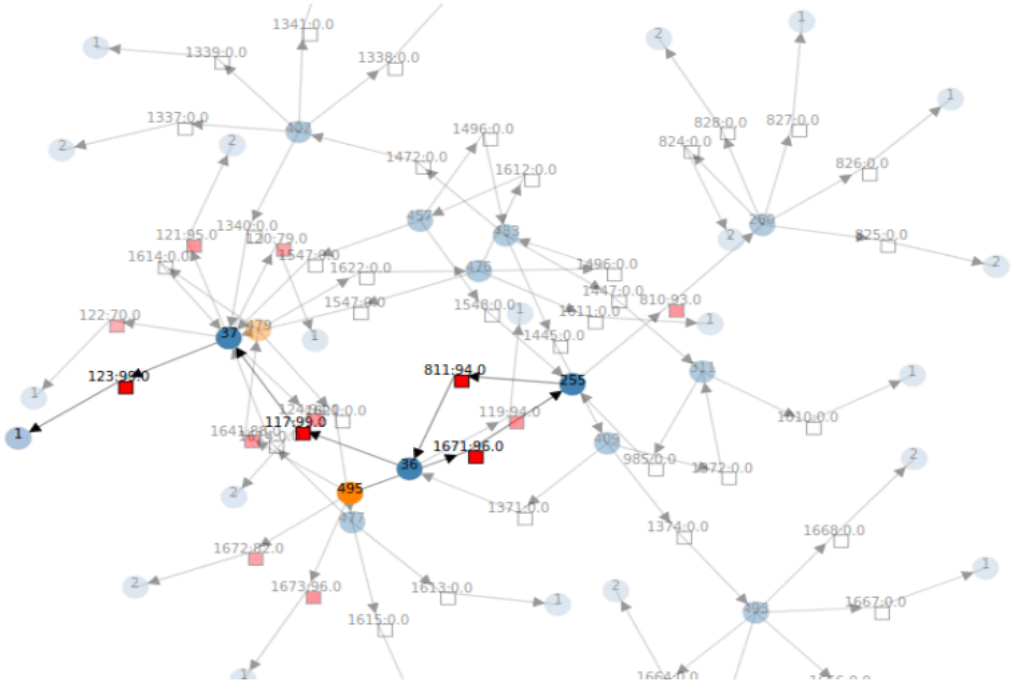

Figure 2: *Représentation du graphe d'agents de SIVA : Racines en orange, noeuds en bleu foncé, groupements d'agents pour les carrés (rouge ou blanc), feuilles en bleu ciel*

causales et des inférences statistiques.

4.2.2. Système décisionnel

4.2.2.1. Exécution des agents

À l'échelle de SIVA, la prise de décision est la résultante des contributions de ses agents :

- SIVA écoute uniquement les agents dont les observations sont effectives
- Des poids sont accordés selon leur expérience d'observateur. Des agents plus âgés, ayant vérifié leur corrélation un grand nombre de fois, sont considérés expérimentés
- La décision est prise aléatoirement si défaut de confiance envers les agents il y a. Ce qui a pour but de favoriser l'exploration des auteurs lors de l'apprentissage

Toute décision est donc prise en accord avec les conseils de tous les agents exécutés dont l'observation est fiable. Comme présenté fig. 3, deux phases sont à distinguer dans la prise de décision (2 et 3). Durant la première (2), seul le graphe d'agents est sollicité. Son exécution revient à l'exécution des agents d'une racine à une feuille. En effet en commençant par une racine, à chaque noeud, le groupement d'agents le plus précis dans ses observations indique le noeud suivant. Et ainsi de suite jusqu'à atteindre un sommet du graphe, soit une feuille, synonyme de fin d'exécution. Puis, c'est par la mise en commun des **corrélations** de chaque agent exécuté, qu'émerge la décision du graphe. C'est uniquement si la confiance en cette décision est trop faible que la base d'agents

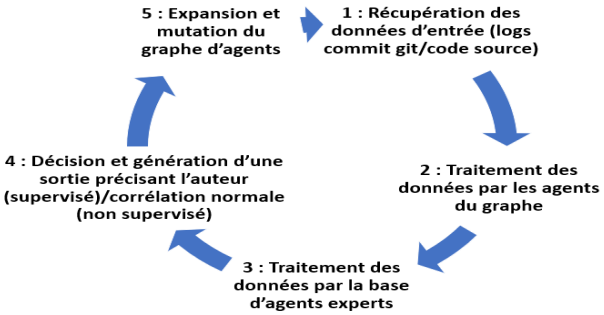

Figure 3: *Principe général d'apprentissage de SIVA*

experts et éprouvés est sollicitée en tant que seconde phase (3). Cette base étant massive et emplie de connaissances, son exécution coûte du temps mais apporte davantage de fiabilité à la décision. Si la confiance en la décision du graphe est suffisante, celle-ci est alors directement appliquée, sans exécution de la base experte. Dans tous les cas, la sortie correspondante est générée (4). Ce système rejoint celui de D. Kahneman qui décrit dans [8] deux modes de pensée : l'un rapide et instinctif, l'autre plus lent, plus réfléchi et plus logique. Dans bien des cas, la spontanéité, la simplicité et la rapidité du système 1, c'est-à-dire le graphe, suffisent à prendre une décision. Mais dans des situations plus complexes où la réflexion doit être poussée, le système 2 ou base experte, prend le relais afin de mener à une décision dûment réfléchie, bien que moins réactive.

4.2.2.2. Prise de décision

Nous l'avons énoncé précédemment, chaque agent mémorise des corrélations entre une propriété observée, des décisions et des récompenses. Pour chaque corrélation d'un agent, un compteur mémorise et indique son nombre d'occurrences. Cet ensemble d'informations forme les inférences d'un agent. Une fois que tous les agents du graphe, et éventuellement ceux de la base d'experts, ont été exécutés, toutes les inférences des agents sont collectées. C'est ensuite à partir de toutes ces inférences qu'un score peut être calculé pour chaque décision. Dans SIVA, différentes méthodes de calcul de score définissent autant de politiques. Et c'est par l'usage d'hyper paramètres qu'on attribue un coefficient à chaque politique. De la sorte, l'importance de chaque politique peut être adaptée très simplement à chaque cas d'usage.

Voici une brève présentation des politiques implémentées :

- *aléatoire pour inférences vides* : par absence d'informations, la décision est prise aléatoirement
- *aléatoire pour exploration de nouvelles décisions* : choix de la décision aléatoirement pour explorer le champs des possibles
- *aléatoire par audace* : choix la décision aléatoirement par audace (hyper paramètre fixé avant l'apprentissage)
- *moyenne* : choix de la décision avec les moyennes de toutes les inférences pour chaque décision

- *inférences positives* : choix de la décision avec les inférences aux récompenses positives pour chaque décision
- *moyenne par agent* : choix de la décision avec la meilleure moyenne de récompense pour chaque décision
- *meilleur agent* : choix de la décision avec les agents ayant la meilleure récompense
- *politique moyenne* : choix de la décision avec le meilleur score, même si elle n'est leader dans aucune politique

Il est important de noter que les agents particulièrement performants que sont les agents causaux, dont la propriété est réellement caractéristique d'un auteur, sont parfois comptabilisés deux fois dans le système de décision : une première fois lors du parcours du graphe et une seconde fois par exécution de la base experte. Ceux-ci étant très bons, nous avons pris le parti de leur accorder davantage de poids. Par ailleurs un autre système d'exploration est également présent. Il force SIVA à prendre toute décision qu'elle ne connaît pas encore durant l'entraînement, jusqu'à ce que l'ensemble des décisions soit testées, de sorte à disposer d'inférences entre toutes les propriétés observées et toutes les décisions prises.

Algorithm 1: Prise de décision de SIVA en entraînement

```
for agents and experts' base do
    agent execution;
Graphe browsing and activation of the browsed agents;
for graphe's agents_activated do
    inferences collection;
PoliticsPonderation(hyperparameters);
for each politic do
    compute score of each decision;
decision = decision_bestScore;
if trust_decision < threshold then
    for experts' base do
        inferences collection;
    if Inferences_collected == empty then
        decision = random;
    if mode == training & exploration == true then
        decision = random_exploration;
    for each politic do
        compute score of each decision;
    decision = decision_bestScore;
    return decision
```

4.2.3. Mutations génétiques

La dernière étape de la boucle d'apprentissage de SIVA concerne l'expansion et la mutation du graphe d'agents (5). SIVA intègre des composantes génétiques proches des virus qui sont exploitées dans son système exploratoire :

- Elle est tout d'abord capable de supprimer ses agents les moins performants. Elle possède donc une capacité de purge des populations d'éléments qui ne présentent que peu d'intérêt parce que peu discriminantes par rapport à une classification ou un comportement donné.
- Elle dispose également d'une capacité de **clonage** des meilleures populations d'agents. Celle-ci est suivie de mutations afin de garantir l'exploration au sein du clonage. Cela permet de couvrir le plus d'éléments possibles et pertinents au sein de l'environnement.

Deux systèmes de clonage distincts se retrouvent dans SIVA. L'un est un clonage de précision suivi de **mutations** portant sur des agents unitaires particulièrement performants. Il garantit la reproduction d'agents pertinents tout en ne délaissant pas la dimension exploratoire. Alors que l'autre va cloner des groupements d'agents étant au global performants. Il délaisse ainsi légèrement le clonage de performance au profit de l'exploration.

4.2.4. Interprétabilité / explicabilité

Un des avantages indéniable de SIVA réside dans son potentiel d'explicabilité. Pour chaque décision prise, un pourcentage de confiance est émis par SIVA. Il octroie une prise de recul pour qui essaie d'expliquer les résultats. Hormis dans le cadre de décision exploratoire aléatoire, la prise de décision est déterministe. Et il est possible de retracer le raisonnement de l'algorithme, en partant des observations interprétables de ses agents, jusqu'à la décision finalement choisie. SIVA n'a donc rien d'un algorithme boîte noire. Bien au contraire, nous savons précisément quels agents ont contribué à une décision donnée, et pourquoi ceux-ci ont contribué grâce aux inférences qu'ils ont mémorisé tout au long de l'apprentissage.

La fig. 4 reprend un exemple trivial de prise de décision de classe dans un cadre de seulement trois agents. Nous supposons ici que tous les agents ont été exécutés. Seuls le premier et le troisième effectuent avec succès l'observation de leur propriété : terme "goto" ainsi qu'une indentation sont trouvés dans l'environnement observable. Ceux détectant leur propriété voient leurs inférences collectées. Les agents 1 et 3 donnent ainsi donc leur conseil concernant respectivement les décisions Steeve et Mark. Or, l'expérience de chaque agent entre en ligne de compte. Et il est évident que l'expérience de l'agent 3 surpasse celle de l'agent 1, car 500 est nettement supérieur à 36. Ainsi dans ce contexte précis, l'algorithme choisit la décision "Mark".

Nous avons adopté une politique unique dans cet exemple afin de le rendre clair. Cette politique est celle des inférences positives. Ainsi les auteurs proposés par les deux agents avaient comme poids uniquement leur expérience brute. Cependant, à l'aide d'hyper paramètres, d'autres politiques auraient pu être activées. Ainsi les expériences peuvent

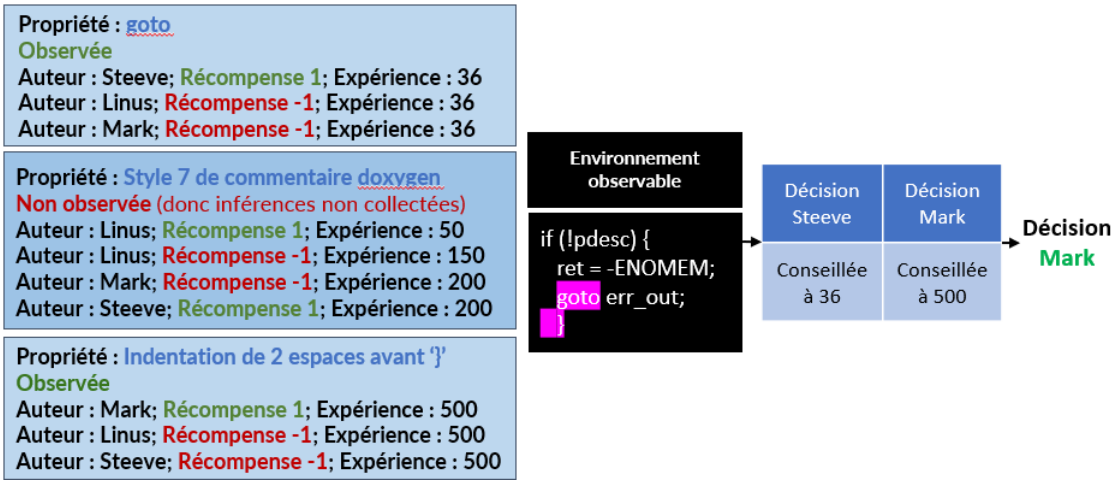

Figure 4: *Exemple de prise de décision*

être prises en compte de différentes manières. Ce qui contribue à accroître le potentiel apport de chaque agent dans une décision de SIVA.

4.3. C4PTION

4.3.1. Typologie des données accessibles

Tant pour l'apprentissage que pour les tests, C4PTION construit et fournit à SIVA 6 environnements observables différents :

1. `Log commit git` : Horodatage, commentaire du commit et fichiers modifiés
2. `Codes ajoutés` : Concaténation des codes source développés exclusivement par l'auteur du commit
3. `Codes bruts` : Concaténation des codes source modifiés et/ou ajoutés par l'auteur du commit
4. `Log commit git + nom auteur` : Concaténation du `Log commit git` et du nom de l'auteur
5. `Codes ajoutés + nom auteur` : Concaténation des `Codes ajoutés` et du nom de l'auteur
6. `Codes bruts + nom auteur` : Concaténation des `Codes bruts` et du nom de l'auteur

Il convient de souligner que les datasets intégrant les noms d'auteurs servent à l'apprentissage non supervisé de la normalité comme décrit ci-après.

4.3.2. Plusieurs modèles

Les environnements 1 à 3 sont utilisés par les SIVA configurées en mode supervisé pour **classifier** des auteurs. Elles construisent ainsi des modèles **syntaxiques et lexicaux** propres

à un langage donné par apprentissage sur des branches de dépôts en source ouverte. Pour chaque décision, un niveau de risque est calculé grâce à la différence entre deux pourcentages de confiance issus de SIVA : celui de l'auteur le plus probable et celui de l'auteur responsable du commit. Selon la valeur de ce niveau, une alerte est levée.

Les environnements 4 à 6 sont quant à eux utilisés par les SIVA configurées en mode non supervisé. Elles construisent des modèles qui effectuent l'apprentissage **comportemental** des auteurs et la **détection d'anormalité** en exploitation. En mode détection, le risque est le ratio des agents qui ont détecté une anomalie par rapport à l'ensemble des agents.

L'utilisation d'environnements différents pour la prise d'une décision unique renforce la densité des critères décisionnels et donc la qualité du modèle final. C'est donc sur l'exploitation de plusieurs SIVA sur des jeux de données différents que se base C4PTION. Une méta décision permet de fusionner les décisions des différentes SIVA afin d'avertir le responsable projet par le biais d'une seule notification en cas d'auteur suspect détecté.

Les modèles syntaxiques, lexicaux et de normalité appris reposent sur les pratiques de développement d'une entreprise dont les dépôts forment les données d'entraînement. Attention, que les pratiques de développement soient bonnes ou mauvaises au sein d'une entreprise, celles-ci sont apprises et constituent un modèle d'entreprise. Ces pratiques apprises constituent des connaissances transférable pour tout dépôt. Après "transfer learning" de cette base et réapprentissage des pratiques des développeurs pour un dépôt donné, C4PTION dispose de 6 modèles opérationnels et est en mesure de rendre un verdict dès lors qu'un développeur pousse son code.

5. Expérimentations et résultats

5.1. Datasets

Les 13 dépôts de code C et C++ choisis pour nos expérimentations sont issus de projets github. Ils regroupent 3 à 179 auteurs uniques chacun. Et 6 d'entre eux comportent moins de 15 auteurs. La répartition en auteurs est donc variée. Nos datasets sont générés par des scripts qui, par utilisation des commandes "git blame" et "git diff", "parsent" les commits des dépôts sélectionnés afin de créer des datasets automatiquement annotés. Ainsi, nulle intervention humaine fastidieuse n'est nécessaire pour créer nos datasets. Voici la liste des dépôts utilisés.

Pour la partie test, 50% des commits de test ont vu leur auteur modifié afin de représenter un commit censé déclencher une alerte. L'altéreration de l'auteur est symptomatique d'une attaque post usurpation d'identité portée par un attaquant qui se serait totalement approprié le style et le comportement d'un développeur du dépôt. Cette malinisation des commits est réalisée par le biais d'un script pouvant altérer les données du commit : l'auteur, le nom de fichier, le message de commit, l'heure, le jour, la time zone et le code.

URL Dépôt (préfixe https://github.com/)	Langage	Nombre d'auteurs uniques
bfgroup/b2	C++	179
yamashi/CyberEngineTweaks	C++	28
DFIR-ORC/dfir-orc	C++	6
ehids/ecapture	C	3
ros/kdl_parser	C++	56
CPPAlliance/NuDB	C++	11
preesm/preesm-apps	C++	14
juliencombattelli/ProjectRomero	C++	6
s-matyukevich/raspberry-pi-os	C	56
brchiu/raytracing	C	4
seL4/seL4	C	119
wkhtmltopdf/wkhtmltopdf	C++	76

Table 1: *Données d'entrée*

5.1.1. Propriétés émergentes

Nous avons développé une librairie de fonctions d'attention destinée à faciliter la gestion des propriétés par les agents. Une fonction d'attention, spécifique à un type d'observation, a pour rôle de faire naître des propriétés observables et parfaitement interprétables par l'humain tout autant que de permettre aux agents de remonter l'état courant de la propriété observée à SIVA. Lors de notre expérimentation, nous nous sommes appuyés sur une fonction d'attention générique existante (LRawP : 2 à 8 caractères alphanumériques consécutifs) et en avons développé de nouvelles plus adaptées au contexte C4PTION. Quelques exemples de fonctions d'attention, spécifiques aux environnements observables de C4PTION, sont présentés ci-après.

- Log commit git (environnement 1)
  - `gitLogMessageSize` : taille d'un message de log git
  - `gitLogDayOfTheWeek` : jour du commit
  - `gitLogFileName` : nom de l'un des fichiers "commités"
- Codes ajoutés (environnement 2) et Codes bruts (environnement 3)
  - `majC` : convention typographique d'une variable (camelCase, snake case, caps lock, etc)
  - `DxS` : convention de style doxygen (11 styles différents)
  - `Sexp` : éléments ("goto", "curl", etc) ne faisant pas partie de bonnes pratiques dans du code C/C++
- Log commit git + nom auteur (environnement 4)
  - `gitLogDetectWordTrack` : mot dans un log
- Codes ajoutés + nom auteur (environnement 5) + Codes bruts + nom auteur (environnement 6)
  - `gitBlameDetectWordTrack` : mot dans un code source

5.2. Résultats

Les entraînements et validations ont eu lieu sur les 13 dépôts avec une VM composée de 16 CPU cadencées à 3GHz et 32Go de RAM. Pour évaluer l'efficacité de la méta décision, les commits ne pouvant fournir l'ensemble des données nécessaires aux six environnements ont été ignorés. Au terme de trois époques d'entraînement (utilisation de 80% du dataset pour l'entraînement) des 6 SIVA sur leur environnement respectif, les coefficients de la méta décision sont optimisés afin d'obtenir des résultats satisfaisants sur l'ensemble des critères. C'est-à-dire que les coefficients doivent trouver une valeur d'équilibre afin de maximiser au mieux possible le taux de vrais positifs (TPR) comme le taux de vrais négatifs (TNR). La mesure utilisée ici est l'"Accuracy" que nous définissons comme suit (avec TP vrai positif/vraie alerte, TN vrai négatif/pas d'alerte, P échantillons positifs, N échantillons négatifs) : Accuracy $= \frac{TP+TN}{P+N}$

À l'issue de la phase d'apprentissage, la validation sur le reste du dataset donne les résultats suivants avec la méta décision utilisant les 6 SIVA : 86,5% d'"Accuracy". De ces entraînements, plusieurs informations ressortent :

- Le modèle initial exploitant une seule SIVA effectuait uniquement de la classification sur le code source brut des commits (environnement 3). L'exploitation de ce type de données lui permettait d'atteindre une "Accuracy" de 72.08%. Afin d'améliorer nos performances, nous avons testé SIVA sur le code source ajouté (environnement 2). L'"Accuracy" a alors atteint 80.39%, soit une nette amélioration
- Les résultats précédents étant décevants, nous avons décidé d'exploiter non pas le code source des commits, mais les metadata que sont l'horodatage, les commentaires du commit ainsi que les fichiers modifiés. L'entraînement et le test de SIVA sur ces données a rendu une "Accuracy" de 84.44%. Ce qui est plus satisfaisant comme résultat
- Néanmoins nous souhaitions aller plus loin. C'est pourquoi nous avons mis en place la méta décision qui exploite les 6 SIVA précédemment présentées. Et nous obtenons cette fois une "Accuracy" de 86.52%. Il s'agit clairement du meilleur chiffre obtenu parmi ceux déjà présentés
- Parmi ces résultats, c'est l'aspect **collaboratif**[9] de C4PTION qui est à mettre à l'honneur. En effet, il convient de noter qu'en utilisant uniquement SIVA 3 (*codes bruts en classification*), ou alors l'ensemble des 6 SIVA, les performances sont accrues de près de 15%. Vous pouvez constater cette différence sur la fig. 5 : les pourcentages de risques d'alerte dans le cas de 6 SIVA collaboratives présentent une séparation alerte/pas d'alerte bien plus prononcée que ne le fait l'usage d'une SIVA unique sur le code source. La méta décision bénéficie donc de ces multiples SIVA
- Par ailleurs, l'exploitation de ses seuils permet de choisir un **focus** sur une minimisation du taux de faux positifs (FPR), du taux de faux négatifs (FNR), ou un équilibre entre les deux. Cela permet de **s'adapter aux besoins et ressources de l'utilisateur**
- Malgré tout, la dimension collaborative n'est pas aussi satisfaisante qu'escompté. Il apparaît effectivement qu'utiliser les 3 meilleures SIVA, ou bien l'ensemble des 6

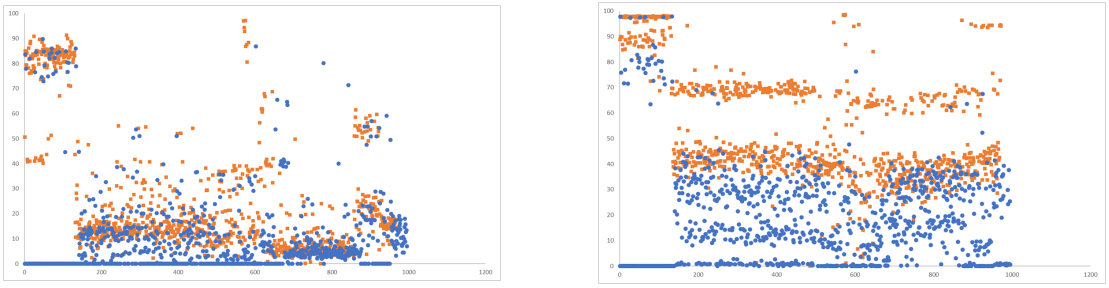

Figure 5: *Pourcentage de risque d'alerte en fonction de chaque commit. En orange sont représentés les commits malins et en bleu les bénins. L'objectif est de maximiser les pourcentages de risque liés aux commits malins, tout en minimisant ceux des commits bénins. Cas code source brut à gauche, cas de 6 SIVA collaboratifs à droite avec entraînement sur 8 000 commits et test sur 2 000. 50% de ces commits sont malins.*

n'octroie qu'une modeste amélioration de quelques dixièmes de pourcentage des performances précédemment présentées

- En outre, les SIVA de détection (utilisant les environnements 4, 5 et 6) ayant pour but de caractériser la normalité n'apportent presque aucune amélioration de résultats. Cela provient très certainement des attentions des agents de ces SIVA. Ce point constitue notre prochain axe de travail pour améliorer notre solution
- Enfin, il est intéressant et logique de remarquer que les niveaux d'usurpation et de confiance sont très dépendants du nombre de lignes par auteur. Ainsi, plus ce nombre est élevé, plus les décisions sont pertinentes. Et ce, avec une valeur étalon de 5 000 lignes pour avoir une confiance marquée quant aux décisions

Avec un protocole de test identique et les mêmes types d'observation, nous avons initié des campagnes sur d'autres langages (de 700 à 1024 commits de tests selon les langages). Les "Accuracy" obtenues sont les suivantes (avec 20 projets par langage) : java (95,12%), javascript (94,78%), php (90,23%) et python (95,61%).

Nous avons également entamé des études préliminaires sur la classification d'auteurs via les transformers [10] sur les datasets 1 à 3. Nous sommes partis d'une architecture type BERT et l'avons ajustée avec de l'apprentissage par transfert, dans un but de classification d'auteurs. La solution parvient à classifier les auteurs avec des taux de classification plutôt corrects : 77% sur l'environnement 1, 61.9% sur l'environnement 2 et 66,5% sur l'environnement 3. Au même titre que la solution SIVA, les transformers parviennent à mieux classifier les auteurs sur les logs de commit que sur le code source.

6. Conclusion

Nous avons développé C4PTION, une solution originale reposant sur une IA apprenant les habitudes de chaque auteur via l'émergence d'agents discriminants, à la fois dans un

graphe évolutif d'agents causaux et une base évolutive d'agents experts. Avec près de 15% gagnés en termes d'"Accuracy", C4PTION démontre que la collaboration entre modèles IA percevant l'environnement de manières différentes est gage de meilleurs résultats et de minimisation de fausses alertes. Il convient tout de même de souligner les limites de cette collaboration, car si passer de 1 à 3 agents est très pertinent, passer de 3 à 6 agents n'accroît que de peu les performances. Par ailleurs, il apparaît plus évident de faire émerger des propriétés discriminantes sur les git metadata que sur le code source. En outre, de gros progrès sont encore à faire sur les attentions de détection d'anomalie.

Afin de consolider les promesses entrevues sur d'autres langages, nous allons entamer une campagne de test massive avant de porter une analyse concernant la qualité de la solution pour des langages ne proposant a priori pas un espace de discriminants aussi vaste que pour le C/C++. De même, les résultats des transformers nous incitent à poursuivre cette piste dans de futurs travaux pour affiner la comparaison avec SIVA, avant d'envisager une intégration dans C4PTION.

En résumé, ces résultats provisoires sont très encourageants car nous savons comment nous améliorer. De plus, nous savons que notre solution s'intègre parfaitement dans une CI Gitlab, ce qui correspond exactement aux objectifs du projet : offrir à nos clients un outil capable de rendre un verdict facilement analysable sur du code poussé sur git, via un rapport d'explicabilité.

Acronymes

C4PTION why *Characterise the Authors of code PorTIONs*?. 1–5, 10–15

CD chaîne de développement continu. 1, 2

CI chaîne d'intégration continue. 1–3, 15

FNR taux de faux négatifs. 13

FPR taux de faux positifs. 13

GP genetic programing. 4

IA Intelligence Artificielle. 2, 3, 14, 15

IAM Identity and Access Management. 3

RSSI responsable de la sécurité des systèmes d'information. 2

SaaS Software As A Service. 3

SIVA SIlicom Versatile Artificial intelligence. 2–15

TNR taux de vrais négatifs. 13

TPR taux de vrais positifs. 13

7. Citations et Bibliographies

References

[1] R. S. R. WG, Security advisory: malicious crate rustdecimal, 2022. URL: https://blog.rust-lang.org/2022/05/10/malicious-crate-rustdecimal.html.

[2] J. I. P. Rave, G. P. J. Àlvarez, J. C. C. Morales, Multi-criteria decision-making leveraged by text analytics and interviews with strategists, 2021. URL: https://link.springer.com/article/10.1057/s41270-021-00125-8.

[3] Z. Li, G. Q. Chen, C. Chen, Y. Zou, S. Xu, Ropgen: Towards robust code authorship attribution via automatic coding style transformation„ 2022. URL: https://arxiv.org/pdf/2202.06043.pdf.

[4] W. Wang, G. Meng, H. Wang, K. Chen, W. Ge, X. Li, A3ident: A two-phased approach to identify the leading authors of android apps, 2020. URL: https://arxiv.org/pdf/2008.13768.pdf.

[5] Y. Wang, M. Alhanahnah, K. Wang, M. Christodorescu, S. Jha, Robust and accurate authorship attribution via program normalization, 2022. URL: https://arxiv.org/pdf/2007.00772.pdf.

[6] S. Kelly, M. I. Heywood, Emergent Tangled Graph Representations for Atari Game Playing Agents, cDermott J., Castelli M., Sekanina L., Haasdijk E., García-Sánchez P. (eds) Genetic Programming. EuroGP 2017. Lecture Notes in Computer Science 10196 (2017).

[7] O. Gesny, P.-M. Satre, J. Roussel, CBWAR: Classification de Binaires Windows via Apprentissage par Renforcement, https://www.cesar-conference.org/wp-content/uploads/2018/11/articles/C&ESAR_2018_J2-16_O-GENTRY_CBWAR.pdf, 2018.

[8] D. Kahneman, Thinking, fast and slow, Farrar, Straus and Giroux, New York, 2011. URL: https://www.amazon.de/Thinking-Fast-Slow-Daniel-Kahneman/dp/0374275637/ref=wl_it_dp_o_pdT1_nS_nC?ie=UTF8&colid=151193SNGKJT9&coliid=I3OCESLZCVDFL7.

[9] M. Gasse, D. Grasset, G. Gaudron, P.-Y. Oudeyer, Causal Reinforcement Learning using Observational and Interventional Data, arXiv : 2106.14421, 2021. arXiv:2106.14421.

[10] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, CoRR abs/1810.04805 (2018). URL: http://arxiv.org/abs/1810.04805. arXiv:1810.04805.