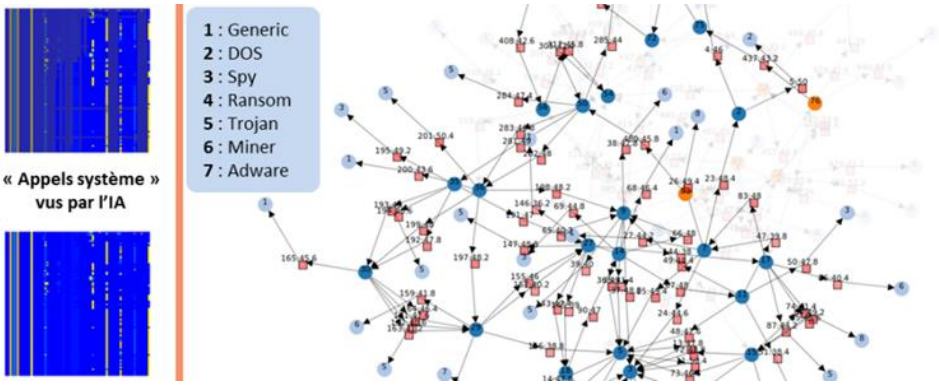
CBWAR : Classification de Binaires Windows via Apprentissage par Renforcement



Olivier GESNY, Pierre-Marie SATRE, Julien ROUSSEL

SILICOM
{ogesny, pmsatre, jroussel}@silicom.fr

Abstract.

Cela fait maintenant plusieurs années que de nouvelles méthodes viennent enrichir le monde de l'intelligence artificielle pour répondre à des défis de notre société. C'est dans cette optique que cet article introduit une nouvelle approche dans la classification de binaires sains et malveillants de type Windows PE. Cette approche s'appuie sur des évolutions de l'algorithme d'apprentissage par renforcement Tangled Program Graphs (TPG) pour permettre la classification multiple de binaires. Le modèle développé introduit un objet mathématique doté de capacités d'apprentissage non supervisé de propriétés de l'environnement. Ce modèle montre de bonnes capacités de découvertes de règles métier pour faciliter le travail des analystes et des sondes de détection de binaires malveillants.

1 Introduction

L'apprentissage par renforcement s'est dernièrement doté d'un nouveau type d'algorithme, le TPG [1]. Celui-ci est capable de diviser la tâche qui lui incombe au travers d'ensembles d'équipes coopératives. Il crée également un processus d'émergence de modularité par le biais de sélection et de combinaison des équipes les plus intéressantes, ce qui conduit à une complexification du système de prise de décision. Cet algorithme constitue la base de notre travail. Nous l'avons modifié afin d'accroître ses capacités exploratoires et prédictives. Nous avons notamment enrichi la plus petite entité qu'est

l'instruction, de manière à lui conférer une aptitude de recherche et de prédiction d'une propriété observable de l'environnement. De même les diverses mutations déjà présentes au sein des différents niveaux du TPG ont été étendues et leurs orientations affinées. Jusqu'à présent, le TPG était principalement éprouvé dans le domaine du jeu vidéo. Que ce soit au travers de l'Atari Learning Environment (ALE) ou bien du ViZDoom FPS environnement [1][2][3]. Bien que nous ayons également eu recours à ce type d'environnement pour comparer nos résultats avec l'état de l'art, la finalité de notre projet était différente. Il s'agissait d'appliquer l'algorithme sur la prédiction multiclasse de binaires. Ainsi, nous présenterons dans cet article l'algorithme TPG qui a inspiré notre travail. Puis nous évoquerons la version que nous avons développée afin d'obtenir un nouvel algorithme prédictif semi supervisé mêlant apprentissage par renforcement, programmation génétique et graphes. Enfin nous nous intéresserons à son application sur la classification de binaires (sain, malveillant, malveillant de type 1, malveillant de type 2, etc).

2 Motivation et travaux relatifs

Les projets de cybersécurité incorporent de plus en plus de techniques d'Intelligence Artificielle (IA). Ainsi, nous avons eu l'idée d'explorer cette voie par le biais d'un algorithme peu exploité jusque-là : le TPG. Ce dernier est lui-même inspiré du Symbiotic Bid-Based (SBB) [4] qui a déjà fait preuve de bons résultats en matière de détection de flux réseau malveillants [5] ou de détection de nom de domaine générés automatiquement [6]. Pour rappel, le système a pour rôle de faire émerger des équipes, de faire émerger les liens entre équipes les plus pertinents au regard de l'objectif fixé et d'exécuter ces équipes. Une équipe est chargée d'exécuter des programmes et d'indiquer au contrôleur IA la meilleure action à réaliser. Un programme est chargé d'exécuter ses instructions et de rendre une offre qui indique le chemin à suivre vers l'action finale ou une autre équipe. Une instruction est chargée de renseigner la valeur d'un registre sur la base de ces 4 opérandes : 1/ Opération ; 2/ Registre ; 3/ Constante, adresse du stimulus ou registre ; 4/ Constante, adresse du stimulus ou registre.

Cet algorithme dispose de plusieurs propriétés particulièrement intéressantes. Il est tout d'abord rapide du point de vue du temps de calcul comparé à des méthodes telles que le Deep Q Learning. Moins de paramètres sont à régler comparativement aux techniques basées sur l'optimisation de la politique par un réseau de neurones. Par ailleurs, il dispose de propriétés d'émergence particulièrement intéressantes résumées dans le tableau ci-après.

Vocable RL classique	Analogie avec le vocable TPG	Initialisation avant période d'apprentissage	Pendant la période d'apprentissage
Agent	Equipes [Team] et programmes	Nombre et capacité des agents fixés dans le code ou par paramétrage	**Emergence de nouveaux agents** qui surveillent certains états
Critères multi objectifs [Multi Objective Reinforcement Learning]	Aucun	Pas de gestion multi objectifs dans la méthode originelle	Pas de gestion multi objectifs dans la méthode originelle
Objectif final/critère de fin d'apprentissage	Performance [Fitness]	Critère fixé par paramétrage Objectif absolu : par exemple, le score doit être de X Objectif relatif : par exemple, le score absolu doit être supérieur au meilleur score précédent	Pas d'émergence de nouveau critère de fin d'apprentissage
Récompense atteinte/objectif intermédiaire	Récompense atteinte	Critère de récompense fixé dans le code ou par paramétrage	Pas d'émergence de nouvelles récompenses
Déclenchement de l'évaluation de la performance	Déclenchement de l'évaluation de la performance	Critère : déclenchement de l'évaluation après chaque action réalisée	Pas d'émergence de nouveau critère de déclenchement de l'évaluation de la performance
Environnement	Environnement	Environnement observable fixé par paramétrage	**Emergence d'états nouveaux** et suppression d'états non intéressants
Corrélations	Corrélations	Aucune corrélation fixée	**Emergence de corrélations entre états de l'environnement et action à réaliser**
Action	Action	Actions intrinsèques fixées dans le code (reproduction et élimination d'équipes, mutation d'équipes et de programmes) Actions unitaires paramétrées dans l'algorithme	Pas d'émergence d'actions unitaires nouvelles Mais **émergence de comportements nouveaux**
Politique	Politique	Politique initiale intrinsèque fixée dans le code ou par paramétrage	**Emergence de nouvelles politiques** et renforcement des meilleures politiques à appliquer en fonction de l'état de l'environnement A aucun moment on ne sait si la meilleure politique est atteinte

Figure 1. Propriétés d'émergence du TPG.

Ces capacités sont notamment l'émergence de nouveaux agents ayant de nouveaux comportements, de corrélation entre états de l'environnement et des actions réalisées par les agents, de comportements nouveaux de la part des agents et plus généralement de nouvelles politiques.

Enfin, ayant moins d'opérations à effectuer, la durée d'apprentissage et les temps de calculs en phase d'exploitation sont réduits. Ce sont autant de raisons qui nous ont conduit à considérer cet algorithme pour classifier des binaires. Il convient de souligner que les travaux ont été partiellement financés par DGA MI dans le cadre de la montée en puissance de l'utilisation de techniques d'Intelligence Artificielle appliquée au besoin de la cyberdéfense.

3 Description du modèle

Nous avons développé un modèle dont l'architecture générale est illustrée ci-dessous. Elle se compose de 2 modules distincts : un module IA et un module métier.

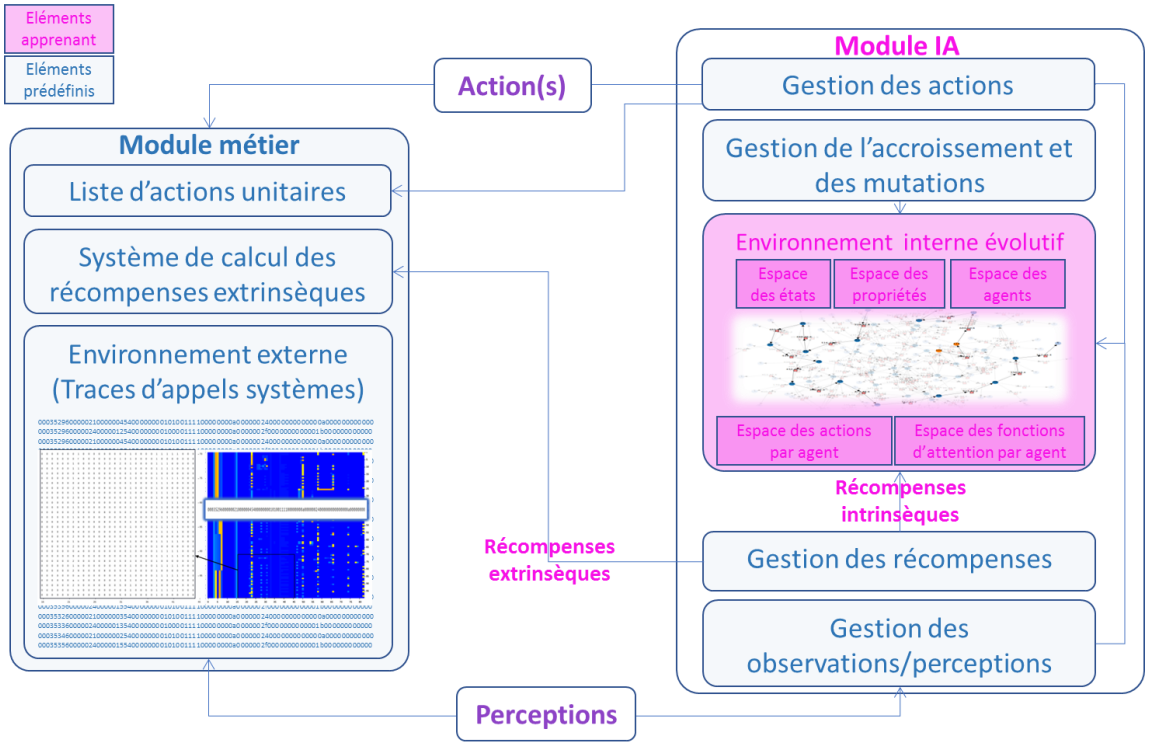


Figure 2. Architecture générale du modèle avec représentation des états des traces d'appels système sous forme d'image.

3.1 Description générale du module IA

Nous avons repris la topologie en graphe du TPG ainsi que ses règles d'évolution, mais avons enrichi le système général de façon à ce que le module IA puisse faire émerger le plus rapidement possible la meilleure politique sur la base des informations apportées par le module métier et son environnement interne évolutif.

Les ambitions de développement de notre modèle ont été de doter le modèle TPG de capacités supplémentaires :

- **D'attention et de curiosité** : découverte de nouveaux états pour augmenter l'espace des états de l'environnement externe ; découverte de nouvelles propriétés de l'environnement externe pour augmenter l'espace des propriétés grâce à l'utilisation de fonctions d'attention ; découverte, dans un rayon d'observation autour d'une adresse définie dans une instruction, d'une propriété déjà perçue à cette adresse
- **De prédiction de propriétés** dans l'environnement externe observable

Afin de satisfaire ces ambitions, nous avons augmenté les capacités de toutes les entités du TPG, de l'instruction jusqu'au gestionnaire maître en passant par les programmes et les équipes.

3.1.1 Modification des instructions

Nous avons augmenté le nombre d'opérandes des instructions pour leur conférer une aptitude de prédiction de propriété de l'environnement (*predictedProperty*) grâce à une opération d'attention qui s'appuie sur des adresses de l'environnement passé ou présent, un rayon d'observation autour de ses adresses et la valeur d'état de chacune de ces adresses. Les opérandes sont les suivants :

1. ***operation*** : fonction d'attention
2. ***tpgRegister*** : registre TPG
3. ***currentRefAddressInEnvironment*** : adresse de l'"objet référence" dans l'environnement présent
4. ***currentCompAddressInEnvironment*** : adresse de l'"objet comparé" dans l'environnement présent
5. ***elapsedDurationSinceLatestObservation*** : saut temporel écoulé depuis la dernière observation de l'"objet "
6. ***foveaSize*** : rayon d'observation de la fonction d'attention
7. ***currentRefState*** : état courant à l'adresse *currentRefAddressInEnvironment*
8. ***currentCompState*** : état courant à l'adresse *currentCompAddressInEnvironment*
9. ***passedRefState*** : état passé de l'adresse *currentRefAddressInEnvironment* à *t-elapsedDurationSinceLatestObservation*
10. ***passedCompState*** : état passé de l'adresse *currentCompAddressInEnvironment* à *t-elapsedDurationSinceLatestObservation*
11. ***distanceRefComp*** : distance entre les 2 objets comparés
12. ***predictedProperty*** : propriété prédite
13. ***acceptablePercentageOfErrorToExactProperty*** : pourcentage d'erreur acceptable pour la propriété prédite au regard de la propriété exacte
14. ***numberOfAcceptablePredictions*** : nombre de prédictions acceptables
15. ***meanInstructionPredictionQuality*** : moyenne de la qualité de prédiction de l'instruction. Mise à jour exclusivement lorsque la prédiction est acceptable
16. ***numberOfExecutions*** : nombre d'exécutions de l'instruction

A l'initialisation de toute instruction, la valeur des 13 premiers opérandes est tirée aléatoirement.

3.1.2 Fonction d'attention

Les opérandes de l'instruction relatifs aux adresses, aux états et aux distances sont autant de paramètres qui peuvent conduire à calculer une propriété de l'environnement. Nous avons développé une cinquantaine de fonctions d'attention qui permettent à toute instruction de prédire une propriété parmi les types : état, transition, distance et valeur mathématique (par exemple une surface). Un état est simplement une valeur à un instant donné, telle que la valeur d'un octet ou d'un bit dans une trame, ou d'un pixel précis sur un écran. Une propriété d'état peut être recherchée en fonction d'un état donné et d'une distance. Les propriétés de transition entre deux états, de distance, de surface et propriétés mathématiques sont trouvées de la même manière, à ceci près que les paramètres utilisés diffèrent nécessairement selon l'opération.

3.1.3 Modification des programmes

Nous avons modifié les programmes de façon à leur conférer des aptitudes de mesure de la qualité de prédiction de propriétés de chaque instruction en plus de leur capacité unique initiale à calculer une offre (bid) sur la base de la valeur contenue dans le registre de la dernière instruction exécutée. L'algorithme général d'un programme est illustré par la figure suivante.

Algorithm 1: Exécution d'un Programme

Notation : P : Programme, I : Instruction

```
for I_i ∈ P do
    Recherche de la propriété prédite par I_i dans son rayon d'observation
    par:
      -Calcul de la propriété exacte dans le présent
      -Calcul de la propriété exacte dans le passé
      -Calcul du range maximum de la propriété
      -Calcul de la performance de prédiction de propriété. Toute
      découverte de performance intéressante met fin à la recherche

    if Mauvaise prédiction then
        if Probabilité de suppression then
            Suppression de I_i
        else
            Mutation de I_i
        end
    else
        Ajout probabiliste d'une I au P courant
        Correction des prédictions présente et passée en fonction de la
        propriété exacte observée
        if Prédiction parfaite then
            Ajout probabiliste d'un clone de I_i plus précis
            Ajout probabiliste d'un clone muté de I_i
        else
            (Prédiction acceptable)
            Ajout probabiliste d'un clone muté de I_i
            Suppression probabiliste de I_i
        end
    end
    Suppression probabiliste de I_i selon sa qualité de prédiction
    Enregistrement de la qualité de prédiction de I_i
end
```

Figure 3. Exécution d'un programme.

Il convient de souligner que le programme a plusieurs objectifs pour une instruction donnée :

- Vérifier que la prédiction de propriété est acceptable. C'est le cas si :

$$\frac{delta}{Range} \leq \text{acceptablePercentageOfErrorToExactProperty}$$

  Où :
  - delta= $abs(Propriété\ prédite - Propriété\ exacte)$
  - Range ≥ 1 est le range maximum d'une propriété et est fonction du type de l'opération qui calcule la propriété :

$$Range = Val_{Max}(Propriété) - Val_{Min}(Propriété)$$

- Corriger exactement la prédiction avant d'ancrer l'instruction. La correction est réalisée lorsque la prédiction est acceptable pour la première fois

- Calculer la performance de prédiction :

$$Performance = \begin{cases} 100, si\ Propriété\ prédite = Propriété\ exacte \\ (100 - acceptablePercentage)\left(1 - \frac{delta}{Range}\right), \quad si\ la\ propriété\ est\ acceptable \\ 0, \quad sinon \end{cases}$$

- Recalculer la performance moyenne de prédiction de l'instruction à chaque fois que la prédiction est acceptable

Au fur et à mesure de leur exécution, les instructions qui présentent une qualité de prédiction acceptable sont ancrées avant d'être clonées et mutées au sein du programme alors que les autres sont mutées ou détruites par les programmes.

3.1.4 Modification du gestionnaire maître

En phase d'apprentissage, le contrôleur crée (par clonage et mutation) des root teams qui 'lancent' des groupes d'agents (teams, programme et instructions) chargés de retourner leur décision. Chaque root team est exécutée N fois pour un challenge donné. A chaque exécution les opérations suivantes sont effectuées :

- La root team exécute un programme qui retourne son offre à son équipe (team)
- Chaque équipe (team) réalise la décision du programme qui a rendu la meilleure offre. Une décision est soit une action (ex : classification) soit un passage de témoin à une autre équipe
- Lorsqu'une action est réalisée, une récompense intermédiaire (partie intégrante du système de récompenses métier) peut être retournée à l'IA. Il peut s'agir d'un écart entre la réalité prédite et la réalité observée (par exemple une classification)

- Après finalisation du challenge, une récompense finale (score du système de récompense métier) est retournée pour chaque root team
- Lorsque les root teams ont toutes réalisé le batch de challenge, le contrôleur effectue les opérations suivantes :
    - Il calcule la performance de chaque root team (calcul impliquant notamment le score moyen de chaque root team et le score maximum obtenu par au moins l'une des root teams)
    - Il supprime la moitié des root teams ; celles dont les performances sont les plus médiocres
    - Il supprime les programmes les plus médiocres en termes de qualité de prédiction en accord avec la probabilité renseignée par l'utilisateur dans le fichier des hyperparamètres)
    - Il clone N fois la meilleure root team en compensant la suppression des root teams les plus faibles
    - Il mute les nouvelles root teams en accord avec les probabilités renseignées par l'utilisateur : suppression/ajout/mutation de programmes, suppression/ajout/mutation d'instructions, mutation des décisions. La mutation des décisions favorise les actions originales. La mutation des programmes favorise l'exploration de l'environnement et la découverte de nouvelles propriétés

Notre algorithme dispose d'une grande capacité d'évolution. Celle-ci se retrouve au niveau du principe de mutation. La mutation intervient au niveau des équipes, des programmes, des instructions et est au service de l'exploration. Après l'évaluation des équipes racines, une fraction de celles-ci est supprimée. La même quantité d'équipes est rajoutée, avec cependant des mutations à tous les niveaux : équipes, programmes et instructions. De plus, d'autres changements interviennent lors de l'exécution de l'algorithme afin d'accroître l'exploration. Par exemple une équipe peut être ajoutée par clonage en fin de graphe. Auquel cas, les programmes qu'elle référencera seront en partie des programmes existants, mais également des programmes complètement originaux. De même, au sein d'un programme, les meilleures instructions sont clonées, et subissent de légères mutations pour explorer de nouvelles possibilités. Enfin dans un souci d'amélioration permanente, après chaque première exécution d'une instruction, sont mises à jour les adresses ainsi que la prédiction du résultat de l'opération. Cela permet de renforcer l'efficacité de chaque instruction. Le principe général de notre algorithme est illustré ci-dessous.

Algorithm 1: Algorithme TPG modifié

Notation : Er : Ensemble des équipes racine, E : Equipe,
P : Programme, I : Instruction, A : Action atomique;

```
while Recherche meilleure politique do
    for Er_i ∈ Er do
        for P_i ∈ Er_i do
            for I_i ∈ P_i do
                Exécution(I_i ∈ P_i);
                Evaluation(I_i ∈ P_i);
                Clonage et mutation, ou suppression des instructions en
                fonction de leur évaluation
            end
            Evaluation(P_i);
        end
        if P_best ∈ Er_i pointe sur une E then
            UpdateFitness(Er_i);
            Réitération dès le premier for avec E
        else
            Ajout d'une E remplaçant l'A d'un P dans Er_i
              Peuplée de P existants et de P originaux
            Fitness(Er_i)
        end
    end
    Classement(Er_i ∈ Er) en fonction de leurs fitness respectives
    Suppression des pires Er_i ∈ Er
    Suppression des P dépourvus d'E
    Ajout d'autant d'Er_i que de supprimées par clonage des Er_i existantes
    Mutation des Er_i ajoutées par :
      - Suppression des pires P
      - Clonage du P_best pour le faire pointer sur une équipe ancêtre ou
    une A
      - Clonage aléatoire d'un P
      - Création originale d'un P
end
```

Figure 4. Principe général algorithmique du module IA.

3.2 Description générale d'un module métier

Un module métier a plusieurs objectifs : constituer l'environnement observable par le module IA, exposer les API permettant au module IA de réaliser des actions unitaires (naviguer dans l'environnement observable, prendre des décisions de classification ou autre) et enfin structurer le système de récompenses indispensable pour l'apprentissage par renforcement.

Un tel module expose des API appelables par le module IA qui lui permettent de récupérer le nombre d'éléments de chaque dimension de l'environnement observable, le nombre d'actions unitaires, des éléments de l'environnement observable et d'effectuer les actions unitaires spécifiques au problème à résoudre (par exemple : se déplacer dans l'environnement observable, choisir une classification, arrêter le challenge, etc).

Ce même module fixe les objectifs d'apprentissage suivants :

- Obtenir le meilleur score moyen sur l'ensemble des échantillons du batch d'apprentissage
- Obtenir le meilleur score moyen sur l'ensemble des échantillons de confirmation d'apprentissage

Le score est le reflet de règles métiers. Pour un challenge de classification d'images, il montre par exemple l'évolution de la somme des écarts entre classifications prédites et classifications réelles rapporté au nombre d'images à classifier. L'évolution du score est formalisée par des récompenses intrinsèques et extrinsèques. Dans notre modèle, les récompenses intrinsèques (mode non supervisé) dépendent d'une motivation (par exemple : environnement entièrement observé). Les récompenses extrinsèques (mode supervisé) sont quant à elles le reflet de règles métier.

Il convient de souligner qu'un nouveau module doit être développé pour tout nouveau challenge impliquant de nouveaux objectifs (par exemple : classifier des textes, résoudre un problème d'optimisation ou de planification).

4 Expérimentations

4.1 Dataset de fichiers « Appels système »

Pour les besoins de classification de binaires, nous avons constitué des fichiers json formalisant les traces d'appels système de binaires issus de sources ouvertes (malekal, stratosphereips, theZoo, binaires sains récupérés via des sites de téléchargement). 8087 binaires Windows PE ont ainsi fait l'objet d'une classification multiple (sous-ensemble de la liste {SAFE, GENERIC, DOS, SPY, RANSOM, TROJAN, MINER, ADWARE}) via Virus Total et d'une exécution en environnement sandboxé Cuckoo pour obtenir des rapports d'analyse. Seuls 4930 binaires étant valides, notre dataset est ainsi constitué de 4930 fichiers json séparés en 2 sous-ensembles, chacun étant associé à un fichier donnant la classification multiple de tous les binaires considérés valides :

- Le premier dataset comprend 2773 fichiers json uniquement dédiés à l'apprentissage
- Le deuxième dataset comprend 2157 binaires d'évaluation

La répartition n'est pas totalement équilibrée car nous avons constaté au cours des tests que le deuxième dataset comprenait des rapports cuckoo vides. Il convient de souligner que chaque binaire réellement exploitable concentre 9579 appels système en moyenne.

Un fichier « Appels système » contient seulement l'extrait des rapports Cuckoo relatifs aux appels système : le nom de l'API système, les timestamps de l'appel (en ms et en s), l'identifiant ayant été donné à l'API durant la phase de transformation des rapports Cuckoo, la valeur de retour de l'appel, le succès de l'appel, une valeur d'intérêt des arguments passés à l'API (0 si aucun des arguments n'est intéressant, 1 sinon), un masque binaire (ex: 101011) permettant de savoir si un argument est utilisé ou pas et enfin la valeur des arguments passés lors de l'appel (pour une string ou une structure : le nombre d'octets la constituant est retourné, pour un entier : la valeur de l'entier est retournée).

4.2 Mode d'entraînement mixte

Dans le but d'éviter l'overfitting, nous avons développé une méthode d'entraînement spécifique basée sur le découpage du dataset d'apprentissage en 2 sous-ensembles :

- Sous-ensemble d'apprentissage permanent : pour un certain nombre d'échantillons tirés aléatoirement parmi le sous-ensemble, le modèle demande l'évolution du score après chaque action qu'il entreprend.
- Sous-ensemble de confirmation d'apprentissage : pour tous les échantillons de ce sous-ensemble, le modèle demande l'évolution du score seulement à l'issue de chaque challenge.

L'évaluation de la performance des root teams n'est effectuée qu'à l'issue de la réalisation d'un batch du sous-ensemble d'apprentissage permanent et de l'intégralité des challenges du sous-ensemble de confirmation d'apprentissage.

4.3 Module métier

Le module métier développé fournit, sur invocation d'une API, N traces d'appels système selon un format hexadécimal. Diverses actions unitaires offertes à l'IA ont été implémentées dans le module métier : arrêt du challenge, déplacement d'un tiers de la fenêtre de temps dans le futur ainsi que les 7 classifications malveillantes. Sur demande de déplacement, le module construit en temps réel une matrice constituée de 100 lignes (hyperparamètre du problème) et au minimum 83 colonnes représentant l'intégralité des informations relatives à l'appel système (timestamp, id de l'API, valeur des arguments, etc).

Les règles métier implémentées permettent de calculer le FPR, le TPR, le FNR et TNR sur la base de l'écart entre la multiclasse prédite d'un binaire et la multiclasse réelle formalisée dans les fichiers json. Dans le cadre de nos travaux de détection/classification de malwares, l'objectif est un FPR de 0%. Le score retourné à chaque demande du module IA est le résultat d'un écart entre la classification réelle et la classification rendue par le module IA après chaque action ou à l'issue d'un challenge. La classification est de la forme $\{X_i\}$ où $i \in [1;8]$.

Sur action de classification unitaire par le module IA, le module métier la mémorise et détermine en cours et à la fin du challenge la récompense sous la forme d'un écart entre la

classification multiple rendue par l'IA et la classification multiple réelle. Cet écart est le résultat du calcul suivant :

$$\frac{\text{TP x PondTP + TN x PondTN}}{\text{FP x PondFP + FN x PondFN}} \; avec \; \text{FP x PondFP} + \text{FN x PondFN} > 0 \quad (1)$$

Où : TP représente le nombre de vrais positifs ; FP, le nombre de faux positifs ; TN, le nombre de vrais négatifs ; FN, le nombre de faux négatifs et enfin les hyperparamètres de pondération PondXX permettant de favoriser ou défavoriser les taux associés aux TP, FP, TN et FN.

4.4 Résultats

Nous avons testé notre modèle sur le dataset d'apprentissage puis l'avons évalué sur le dataset d'évaluation. Les résultats, réalisés sur un ordinateur standard disposant de 16 Go de RAM et de 4 cœurs CPU tournant à 3,4 GHz, sont illustrés (fig. 5) au travers des courbes d'apprentissage ainsi que des chiffres exposés dans le tableau (fig. 6) ci-après.

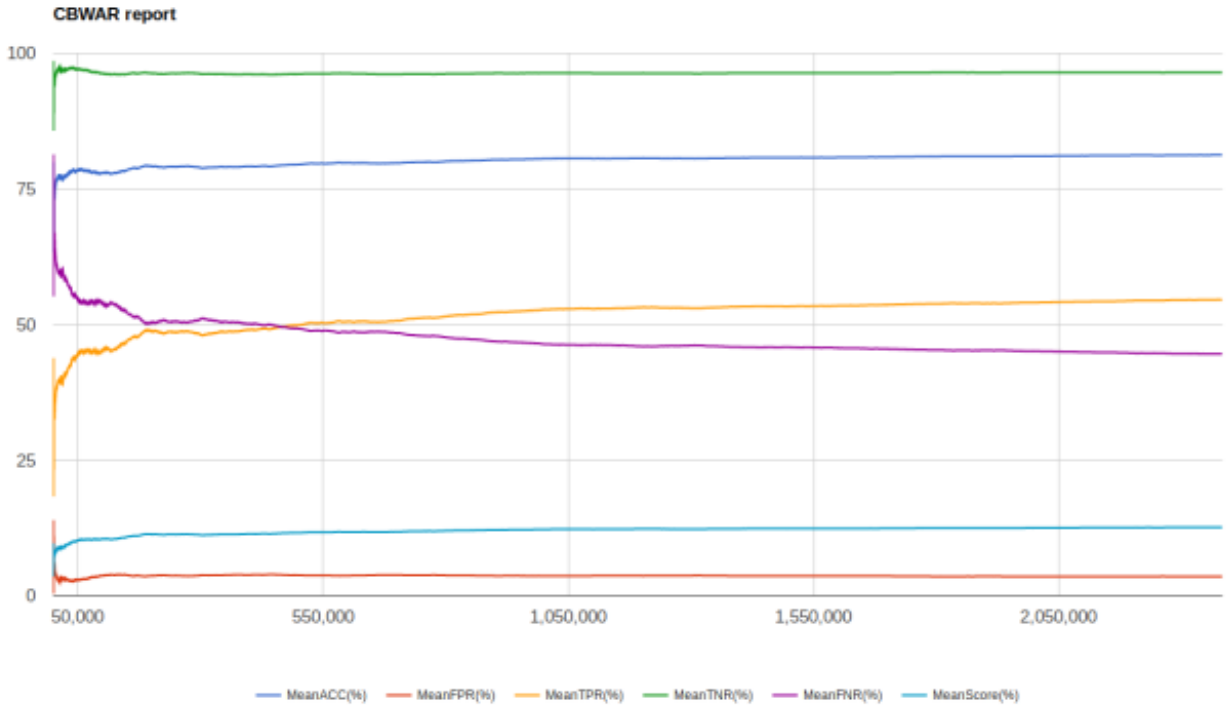


Figure 5. Courbes d'apprentissage.

Essais	Durée d'apprentissage (s)	FPR (%)	ACC (%)	TPR (%)	TNR (%)	FNR (%)
Fenêtre d'acquisition : 100 Pondérations : - TP : 0.75; FP : 1.0; TN : 0.0; FN : 0.0 Batch : 100 Mode mixte : - 1386 binaires d'apprentissage - 1386 binaires de confirmation d'apprentissage Evaluation finale : 2157 binaires	113478 (~32h)	0,28	89,16	71,67	99,76	27,12

Figure 6. Résultat de l'évaluation sur 2157 binaires.

Il convient de souligner que les programmes de l'IA se focalisent notamment sur l'observation des colonnes relatives aux identifiants des appels système ainsi que les colonnes proches (correspondant à la valeur de retour de l'appel et au succès de l'appel). Cela démontre une faculté de découverte d'enchaînement entre perception de propriétés intéressantes d'un point de vue métier. Le FNR (False Negative Rate) est élevé du fait d'une pondération PondFN nulle qui ne favorise en aucun cas l'atteinte rapide d'un faible FNR.

4.5 Discussions et interprétation

Le modèle développé répond à un besoin de recherche automatique de discriminants dans un ensemble de données générées lors de l'exécution de binaire. Dans l'optique d'une classification multiple la plus fine possible, il vise un FPR de 0%. Même si de nombreuses solutions existent quant à la détection de malwares [7] grâce à des techniques de data mining diverses, à notre connaissance, aucune solution permettant de faire de la classification multiple de binaires n'existe.

4.5.1 Avantages

Le modèle est basé sur la coopération entre des équipes (teams) de programmes qui permettent de faire émerger des corrélations entre objectifs intermédiaires intrinsèques, actions et perceptions (passées et présentes). La prise de décision plus rapide repose sur les aptitudes suivantes du modèle :

- Ancrage de prédiction de propriétés réelles constatées dans l'environnement grâce à un mécanisme d'attention et d'acceptation de prédictions proche de celles constatées antérieurement par le modèle. Cet ancrage de prédictions évite d'observer l'intégralité de l'environnement avant de prendre une décision
- Utilisation de l'environnement interne : notre modèle permet de tirer parti de son apprentissage passé ancré dans sa mémoire (graphe de programmes) et des liens entre programmes au travers des équipes, pour donner une direction vers la-

quelle la décision doit s'orienter ; soit une classification, soit un report de décision par une autre équipe

- A l'instar de l'algorithme A3C [8], le modèle dispose de nombreux acteurs (teams et programmes) et d'un critique (le gestionnaire maître) qui assure à terme de choisir le trajet vers la meilleure décision possible à un instant donné étant donné l'apprentissage passé
- Conservation des programmes et création de liens entre les équipes de programmes qui amènent à la meilleure décision au regard de l'objectif fixé au départ
- Elimination des instructions, programmes et root teams qui ne proposent pas de bonne classification ou prédiction
- En phase d'exploitation du modèle, seules quelques millisecondes sont utiles pour rendre une classification
- Le modèle dispose d'un mécanisme de récompenses intrinsèques qui permet de découvrir dans l'environnement des propriétés discriminantes propres à une classe de binaire

4.5.2 Inconvénients

Malgré de bons résultats initiaux, le modèle est encore perfectible :

- Les mutations ne sont pas suffisamment orientées pour permettre un apprentissage rapide. Le côté aléatoire inscrit dans l'algorithme TPG est encore de rigueur et prime sur les quelques mutations orientées nouvelles qui favorisent la suppression de programmes ayant une qualité de prédiction moyenne insuffisante (inférieure à un seuil fixé par hyper paramètre)
- Certains trajets qui passent par des programmes exposant des qualités de prédiction instantanées élevées ne sont pas parcourus faute de lien entre ces programmes. Les liens entre programmes au travers des équipes sont créés par mutations aléatoires à l'issue d'un challenge. Or les liens d'intérêts sont ceux qui réunissent les programmes fournissant une qualité de prédiction instantanée élevée. Par conséquent, ces liens d'intérêt n'émergent pas facilement
- La confiance de classification n'est pas rendue par le modèle. Or des opérandes pourraient être rajoutés à l'instruction pour conserver la relation entre la valeur de la propriété ancrée et la classification réelle
- Dans notre modèle, la notion de registre du TPG a été détournée dans le sens où elle n'est plus vraiment utilisée pour partager des informations entre instructions. Il serait intéressant de réutiliser les registres pour échanger des informations entre instructions (par exemples des prédictions de score)

5 Conclusions et perspectives

Les développements réalisés (en C++) permettent désormais de disposer d'une maquette de classification de binaires intégrant un modèle reposant sur des techniques d'IA d'apprentissage par renforcement et en particulier l'algorithme TGP modifié.

L'algorithme ne fait pas seulement la distinction binaire sain versus binaire malveillant puisqu'il a la particularité d'établir une classification multiple choisie parmi 8 classes (SAFE, GENERIC, DOS, SPY, RANSOM, TROJAN, MINER ou ADWARE). A ce titre, les résultats obtenus sont très encourageants puisque d'ores et déjà, sans optimisation particulière, le modèle a montré des capacités de classification multiple intéressantes. En effet, après un apprentissage sur 2772 binaires, il est capable d'apprendre leur classification multiple en quelques dizaines d'heures sur un ordinateur standard. L'évaluation sur un dataset d'évaluation distinct du dataset d'apprentissage montre un taux de faux positifs de 0,28%.

Pour conclure, nous parions sur ce modèle et de futures évolutions remédiant aux inconvénients exposés pour améliorer le travail des analystes de binaires pour lesquels disposer d'un taux de faux positif le plus proche possible de 0% est primordial.

Afin de rapprocher davantage le TPG des fondamentaux Q-Learning [9], des techniques récentes de prise en compte de la mémoire interne dans les algorithmes d'apprentissage par renforcement [10][11] et des hypothèses d'amélioration exposées dans [12], nous ambitionnons de doter notre algorithme de capacités complémentaires, notamment : **amélioration des capacités de mutations** (immédiatement après perception) pour ancrer les corrélations élevées entre perception de propriétés et dernières actions réalisées (notion d'empowerment) ; **prédiction plus fine des meilleures actions à réaliser** par une **prise en compte plus intensive de la mémoire interne** telle que les actions réalisées antérieurement ou en cours de réalisation (notion de proprioception) et les niveaux de récompenses intrinsèques et extrinsèques déjà obtenus. A noter que les récompenses extrinsèques sont considérées, de notre point de vue, comme des signaux physiologiques (notion d'embodiement) ; **réalisation de plusieurs actions simultanément**.

6 Références

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

[2] Kelly S., Heywood M.I.*: Multi-Task Learning in Atari Video Games with Emergent Tangled Program Graphs.* In: *Proceeding GECCO '17 Proceedings of the Genetic and Evolutionary Computation Conference Pages 195-202,* 2017.

[3] Smith R.J., Heywood M.I.*: Scaling Tangled Program Graphs to Visual Reinforcement Learning in ViZDoom.* In: *Castelli M., Sekanina L., Zhang M., Cagnoni S., García-Sánchez P. (eds) Genetic Programming. EuroGP 2018. Lecture Notes in Computer Science, vol 10781. Springer, Cham 2018.*

[4] Lichodzijewski P., Heywood M.I.*: Symbiosis complexification under* GP. In: *Proceeding GECCO '10 Proceedings of the 12th annual conference on Genetic and evolutionary computation Pages 853-860, 2010.*

[5] Habbadi F., Runkel D., Zincir-Heywood A.N., Heywood M.I.: *On botnet behaviour analysis using GP and C4.5*. In: Proceeding GECCO Comp '14 *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation Pages 1253-1260, 2014.*

[6] Haddadi F., Kayacik H.G., Zincir-Heywood A.N., Heywood M.I. (Dalhousie University): *Malicious Automatically Generated Domain Name Detection Using Stateful-SBB. In: European Conference on the Applications of Evolutionary Computation Pages 529-539, 2013*

[7] Souri A., Hosseini R.: *A state-of-the-art survey of malware detection approaches using data mining techniques*. *In: Human-centric Computing and Information Sciences, 2018.*

[8] Mnih V., Badia A.P., Mirza M., Graves A., Lillicrap T.P., Harley T., Silver D., Kavukcuoglu K. (DeepMind, Montreal Institute for Learning Algorithms (MILA)): *Asynchronous methods for deep reinforcement learning*. *In International Conference on Machine Learning (ICML 2016)*

[9] Sutton R.S.,Barto A.G.: *Reinforcement learning: An introduction (second edition) 2018*

[10] Schaul T., Quan J., Antonoglou J., Silver D. (DeepMind): *Prioritized experience replay. In: International Conference on Learning Representations (ICLR 2015)*

[11] Unknown authors: *R2D2: Recurrent experience replay in distributed Reinforcement Learning. In: Under review as a conference paper at International Conference on Learning Representations (ICLR 2019)*

[12] Doncieux S., Filliat D., Díaz-Rodríguez N., Hospedales T. ,Duro R., Coninx A., Roijers D.M., Girard B., Perrin N., Sigaud O.: *Open-Ended Learning: A Conceptual Framework Based on Representational Redescription. In: Frontiers in Neurobotics 2018*