Datascience in support of cybersecurity operations : Adaptable, robust and explainable anomaly detection for security analysts

Alexandre Dey

DOCTORAT BRETAGNE LOIRE MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THÈSE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPÉRIEURE
MINES-TÉLÉCOM ATLANTIQUE BRETAGNE
PAYS-DE-LA-LOIRE - IMT ATLANTIQUE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Alexandre DEY

Datascience in support of cybersecurity operations

Adaptable, robust and explainable anomaly detection for security analysts

Thèse présentée et soutenue à IMT Atlantique, Rennes, le 02 décembre 2022
Unité de recherche : IRISA - UMR 6074
Thèse N° : 2022IMTA0328

Rapporteurs avant soutenance :

Hervé DEBAR	Professeur, Télécom SudParis
Michaël HAUSPIE	Maître de conférences, HDR, Université de Lille

Composition du Jury :

Président :	Guillaume DOYEN	Professeur, IMT Atlantique
Examinateurs :	Isabelle CHRISMENT	Professeur, Télécom Nancy
	Hervé DEBAR	Professeur, Télécom SudParis
	Michaël HAUSPIE	Maître de conférences, HDR, Université de Lille
Dir. de thèse :	Yann BUSNEL	Professeur, IMT Atlantique
Co-dir. de thèse :	Éric TOTEL	Professeur, Télécom SudParis

Invité(s) :

Benjamin COSTE	Ingénieur de recherche, Ph.D, Airbus Defence and Space
Adrien BECUE	Responsable Technologie, Airbus Protect

Notice

Les travaux présentés dans cette thèse ont été effectués dans le cadre d'un partenariat entre Airbus Defence and Space et l'IMT Atlantique. Bénéficiant du soutient de l'ANRT dans le cadre d'une convention CIFRE, ces recherches ont pu profiter d'une complémentarité entre les points de vues académique et industriel.

Un grand merci à Airbus Defence and Space pour avoir financé et encadré ces travaux de recherche.

Remerciements

Tout d'abord, je tiens à remercier Hervé Debar et Michaël Hauspie pour avoir porté un intérêt à mes travaux et avoir rapporté cette thèse. De même, je remercie Isabelle Chrisment et Guillaume Doyen pour avoir accepté de faire partie du jury en tant qu'examinateur. Je remercie également Yann Busnel pour avoir accepté de prendre le rôle de directeur en fin de thèse.

Je tiens tout particulièrement à remercier Eric Totel pour avoir dirigé mes travaux tout au long de ma thèse. Bien que cette période n'ait pas toujours été facile, ses conseils avisés, ses critiques pertinentes et le temps dédié à mon encadrement m'ont grandement apporté. Je remercie également Adrien Bécue pour avoir rendu cette thèse possible et son encadrement côté industriel, et Sylvain Navers pour son encadrement au début des travaux. Par ailleurs, merci à Nicolas Prigent pour les discussions passionnantes et son aide pour la relecture du présent manuscrit.

Ces années n'auraient pas été les mêmes sans tous mes collègues et la très bonne ambiance qu'ils ont entretenue dans les jeunes locaux rennais d'Airbus Defence and Space. Un grand merci à Frédéric Dubois pour m'avoir accueilli dans les locaux avant même leur inauguration officielle et pour m'avoir soutenu et aidé à naviguer au travers des processus parfois sinueux de l'entreprise. Merci également à tous mes collègues sur Élancourt, avec qui j'ai eu des discussions pertinentes qui m'ont permis de prendre beaucoup de recul sur mes travaux.

Pour les longues heures passées au bureau à se soutenir mutuellement, les débats qui se poursuivaient autours de verres ensuite, ainsi que les conseils avisés qui m'ont permis de prendre du recul et de regagner confiance en mon travail, un grand merci à Benjamin Costé. Merci également à Gilles Marion et Nicolas Challard, pour avoir partagé leurs expériences et m'avoir guidé pendant ces années, vous avez grandement participé à construire la personne que je suis aujourd'hui.

À tous mes amis et ma famille qui ont été à mes côtés autant pour profiter des bons moments que pour me soutenir dans les mauvais. À mon frère, Benjamin, qui a vécu ses années de thèse en même temps que moi. Merci est un mot faible pour

exprimer ce que je ressens.

Enfin à ma compagne Marie-Kerguelen, pour son soutien indéfectible, me faire vivre au quotidien les meilleurs moments et me permettre de supporter les pires, je dédie ce manuscrit et les années de travail dont il découle.

Abstract

To defend against sophisticated cyber-criminal organizations and APTs, IT system operators should define and enforce strict security policies. However, defining and maintining perfect security policies that block all attacks and do not impact usability of the system is impossible. Therefore, security monitoring and incident response are often entrusted to Security Operation Centres (SOC) and Computer Emergency Response Teams (CERT). When monitoring an IT system, legitimate behaviours repeatedly triggers false alarms. This causes alert fatigue, as analysts are overloaded to the point they can miss subtle variations that are the consequence of adversary behaviours. This thesis describes methods to allow security analysts, with no expertise in data science, to create and adapt security analytics that leverage machine learning models to automate more of their investigation procedures. This thesis focuses on anomaly detection to highlight unusual behaviours and clustering to regroup similar alerts and avoid repeating the same alerts again and again. We specifically explore the application of these methods to novel attack detection. To assess our approach, we also describe a method to automatically generate user activity to create realistic datasets.

Table of Contents

List of Figures

Résumé substantiel en français

Depuis la fin des années 70 et la mise à disposition des ordinateurs personnels pour le grand public, la société humaine est devenue de plus en plus dépendante des technologies de l'information. De nos jours, la plupart des entreprises opèrent des Systèmes d'Information (SI) et seraient très impactées si elles en perdaient le contrôle. Pour cette raison, ces dernières décennies ont vu une forte augmentation de la profitabilité de la cyber-criminalité. Ceci a entraîné l'émergence d'organisations cyber-criminelles hautement structurées. En parallèle, des acteurs étatiques ont développé leurs capacités offensives dans le cyber-espace. En plus d'objectifs militaires (e.g., le cas du ver Stuxnet [55] qui a significativement impacté le programme d'armement nucléaire de l'Iran), ces acteurs étatiques peuvent poser une réelle menace pour les entreprises, que ce soit au travers du cyber-espionage (e.g., le groupe AQUATIC PANDA [1], qui est supposé avoir ciblé des industries travaillant dans les secteurs gouvernemental, des technologies de l'information et des télécommunications), ou au travers d'opérations destructives qui impactent des secteurs d'activité entiers (e.g., Lazarus Group, qui est supposé être responsable de l'attaque ayant visé Sony Pictures Entertainment en 2014 [2]). Les acteurs soupçonnés d'être financés par ou affiliés à des états-nations sont communément appelés Advanced Persistent Threats (APT). Ils sont constitués de personnels hautement qualifiés et organisés, ont accès à des financements abondants, et sont par conséquent en mesure de mener des opérations à long terme et de grande ampleur en obtenant et maintenant des accès illégitimes et discrets à des SI.

Se défendre face aux menaces

Pour se défendre face aux organisations cyber-criminelles sophistiquées et APT, les opérateurs de SI doivent définir et appliquer des politiques de sécurité strictes.

1. Un rapport sur AQUATIC PANDA est disponible ici: `https://www.crowdstrike.com/blog/overwatch-exposes-aquatic-panda-in-possession-of-log-4-shell-exploit-tools/`
2. Un rapport sur Lazarus Group est disponible ici: `https://home.treasury.gov/news/press-releases/sm774`

Leurs objectifs sont de minimiser la surface d'attaque et l'impact des attaques réussies. En revanche, définir, affiner et maintenir une politique de sécurité parfaite, qui prémunit de toutes les attaques actuelles et futures sans pour autant impacter les fonctionnalités du système, est impossible. Par conséquent, dans le but de réagir rapidement (i.e., avant d'atteindre des dommages irréversibles) aux attaques qui vont nécessairement avoir lieu, il est nécessaire de superviser les SI pour détecter les violations de sécurité. Pour les grosses entités, cette supervision ainsi que la réponse aux incidents de sécurité sont confiées aux Security Operation Centres (SOC) et aux Computer Emergency Response Teams (CERT).

Au sein d'un SOC, les analystes en cybersécurité configurent des systèmes de détection pour analyser les événements de sécurité et alerter dès que des schémas d'attaque connus sont repérés. Souvent, des alertes sont déclenchées par des comportements légitimes. Par conséquent, les analystes doivent investiguer les alertes pour trouver celles qui sont causées par des attaquants et doivent être considérées comme incidents de sécurité. Quand un sous-ensemble de comportements légitimes déclenche des alertes à répétition, les analystes peuvent rapidement être surchargés au point de ne pas apercevoir des variations du comportement ayant conduit à l'alerte, ce qui empêche de qualifier certains incidents en tant que tel. Ce phénomène est appelé *alert fatigue* et est l'un des principaux problèmes que rencontrent les analystes SOC.

Les CERT fournissent une assistance pour gérer la réponse aux incidents sécurité. Pour ce faire, au moment de la réception d'un cas d'incident avéré (e.g., détecté par un SOC), les analystes en cybersécurité au sein du CERT mènent une investigation poussée de données de sécurité (e.g., événements de sécurité venant de la supervision, analyses forensiques des machines infectées, etc.) afin de découvrir un maximum d'information sur les attaquants. Cette information est exploitée pour évincer durablement les attaquants des systèmes impactés et pour enrichir des bases de connaissance contenant du renseignement sur la menace cyber (ou *Cyber Threat Intelligence* en anglais).

Cette CTI est composée d'indicateurs de compromission (e.g., adresses IP et noms de domaines liés aux infrastructures des attaquants, signature de logiciels malveillants, etc.) ainsi que de documentation des comportements adverses observés. Ces comportements sont en général nommés Tactiques, Techniques et Procédures (TTP), les Tactiques décrivant à haut niveau des étapes d'attaque, les

Techniques représentant les méthodes possibles pour effectuer une étape d'attaque et les Procédures étant des implémentations spécifiques de Techniques. En termes de détection, les indicateurs de compromission sont simples à repérer dans les événements de sécurité mais peuvent être facilement modifiés par l'attaquant, tandis que les TTP requièrent des stratégies de supervision avancées et des configuration fines des systèmes de détection, mais sont beaucoup plus complexes à modifier complètement pour les attaquants (voir Fig. 1).

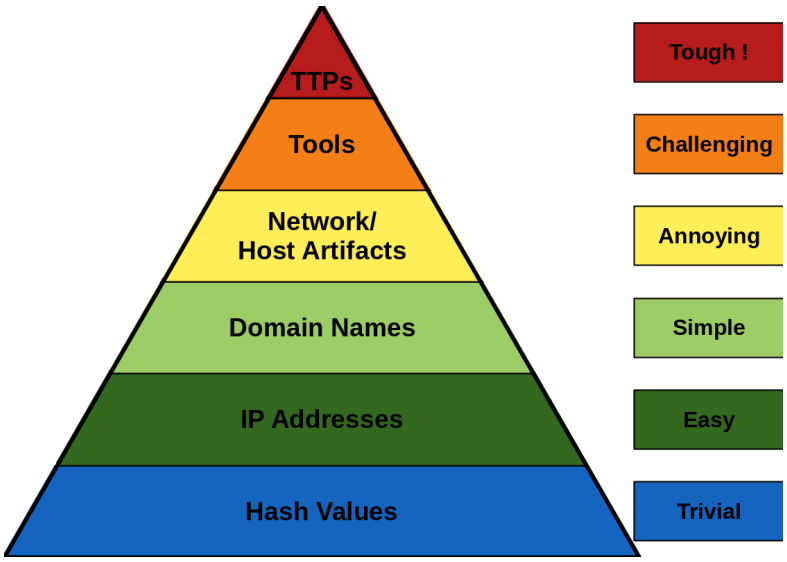


Figure 1 – La *Pyramid of Pain* du *threat hunting* [13]

Quand elle acquiert de nouveaux renseignements sur les groupes d'attaquants, une équipe spécifique d'analystes appelée *threat hunters* peut chercher des preuves d'activité de ces attaquants au sein de l'historique des données de sécurité. Au sein d'un SOC, les *threat hunters* se focalisent sur la détection de violations qui auraient échappées à la détection, tandis que dans les CERT, ils visent également à découvrir de nouvelles méthodologies ou des variations de méthodologies connues. Dans tous les cas, les *threat hunters* analysent de vastes quantités de données et doivent par conséquent se reposer sur des outils pour accélérer et automatiser autant de parts de l'analyse que possible.

Le cas de la science des données au sein des opérations de cyber sécurité

La science des données vise à trouver des méthodes pour extraire des informations d'intérêt et de la connaissance à partir de données. Cependant, bien que divers aspects du travail des analystes en cybersécurité gravitent autour de l'extraction d'information liées aux incidents depuis les données de supervision, les méthodes de science des données, et en particulier le *Machine Learning* (ML), ne sont que peu adoptées au sein des CERT et SOC. En particulier, l'application de la science des données à l'assistance des analystes en cybersécurité se heurte à trois principaux défis [116, 92, 3].

En premier lieu, la plupart des analystes n'ont pas l'expertise et la connaissance en science des données requises pour concevoir et opérer des méthodes à base de ML. De manière similaire, les *data scientists* ne sont souvent pas familiarisés avec les contraintes spécifiques aux opérations de cybersécurité. En particulier, la donnée de supervision provient de multiples capteurs hétérogènes qui fournissent un niveau de visibilité différent sur les comportements adverses, ainsi que des degrés variés de volumétrie et de bruit. Ceci demande des pré-traitements complexes pour appliquer des modèles de ML à ces données. Cette fracture au niveau des connaissances entre les *data scientists* et les analystes en cybersécurité exacerbe la nature de boite noire souvent attribuée au ML par les analystes. De plus, les compétences acquises intuitivement par les analystes à travers leurs expériences sont difficilement formalisables sous une forme que les *data scientists* peuvent exploiter pour concevoir des méthodes adaptées aux opérations de cybersécurité.

En second lieu, contrairement à la plupart des domaines d'application de la science des données, les attaquants cherchent activement à contourner les outils employés par la défense. Par conséquent, une attention particulière doit être portée au moment de la conception pour s'assurer que les outils n'introduisent pas de nouvelles vulnérabilités exploitables par les attaquants. Pour les modèles de ML, il s'agit de prendre en compte les perturbations adverses (i.e., modifications des données spécifiquement pensées pour contourner les modèles), l'empoisonnement (i.e., modifier les données d'entrainement pour dégrader les performances des modèles), mais également prendre en considération la totalité de la chaîne d'acquisition: si les attaquants peuvent modifier les données de supervision, ils peuvent impacter

très sévèrement les capacités défensives.

En dernier lieu, il est difficile d'acquérir des jeux de données représentatifs de cas d'usage de cybersécurité pour expérimenter avec des méthodes issues de la science des données. En effet, les données de supervision contiennent des données sensibles et à caractère personnel. Bien que l'anonymisation des données peut réduire l'exposition des individus, afin d'être efficace (i.e., conserver l'anonymat malgré un croisement des données), elle dégrade la représentativité des jeux de données. Par conséquent, les SOC et les CERT ont en général une interdiction légale, que ce soit au travers de contrats ou de législations (e.g., le RGPD en Europe), de partager de la donnée de supervision réelle. Les jeux de données partageables doivent par conséquent venir d'un environnement de laboratoire avec des utilisateurs factices. La génération d'activité utilisateur réaliste est requise pour fournir des données représentatives et constitue un défi ouvert.

Conception d'analytiques pour traiter des événements de sécurité hétérogènes

Afin d'améliorer la compréhension par les analystes en cybersécurité des résultats fournis par des méthodes issues de la science des données, tout en profitant de leur expertise, nous proposons une approche mettant l'analyste au cœur de la conception d'analytiques de sécurité (i.e., des procédures de traitement automatique de données de sécurité pour assister les analyses). La méthode proposée est adaptée d'un procédé communément employé par les *threat hunters* pour concevoir des analytiques aidant à la recherche de TTP adverses [27]. La méthode proposée permet aux analystes d'incorporer dans ces analytiques des modèles de ML pour la détection d'anomalies et le regroupement par similarité ou *clustering*. Un exemple d'un tel analytique est illustré en Fig. 2.

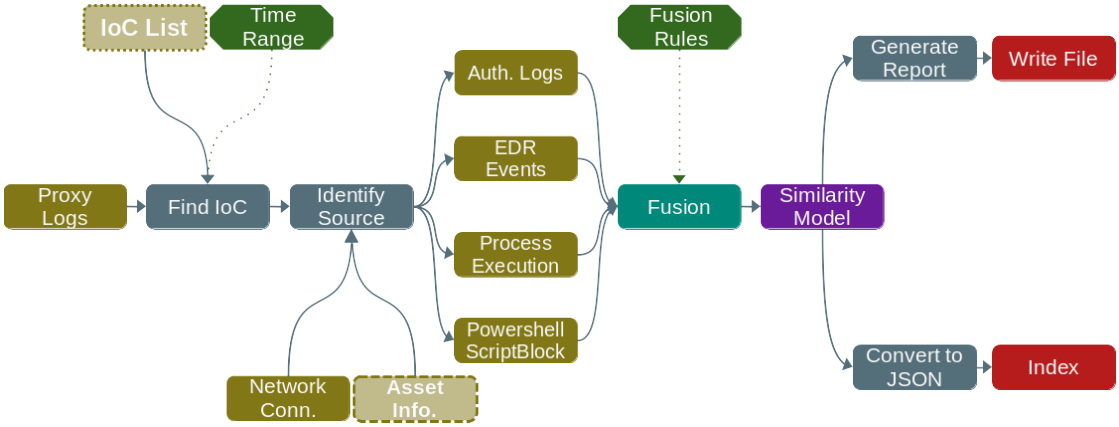


Figure 2 – Exemple de cas d'usage d'un analytique incorporant un modèle de ML: accélérer le *threat hunting* à partir d'indicateurs de compromission.

Réseaux de neurones auto-encodeurs pour traiter des événements de sécurité

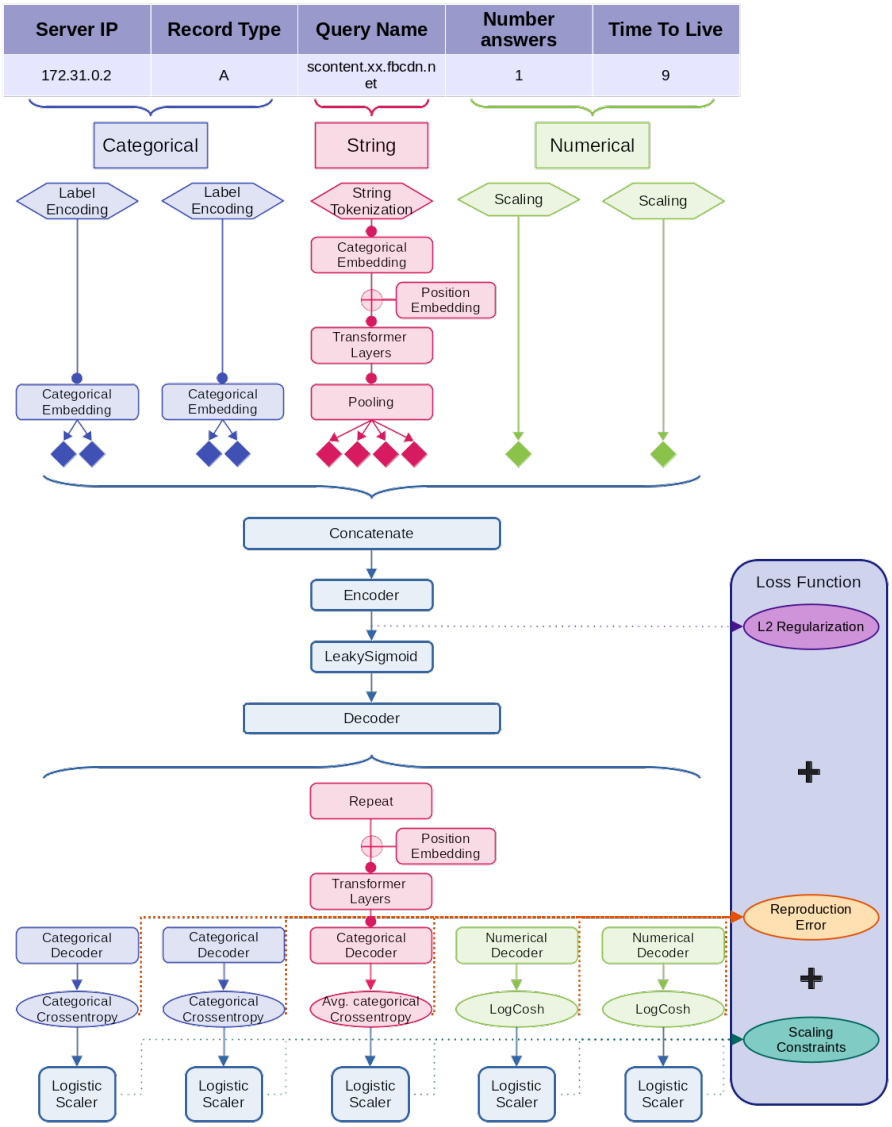


Figure 3 – Structure d'un auto-encodeur pour analyser des événements DNS

L'approche proposée se focalise sur l'abstraction et l'automatisation de la configuration des modèles de ML et des fonctions de pré-traitement associées (nécessaires pour permettre aux modèles de traiter les données). Pour cela, l'approche repose sur du *Deep Learning* (DL), et plus précisément des réseaux de neurones auto-encodeurs, afin de minimiser l'implication de l'humain dans l'étape de trans-

formation des caractéristiques (les réseaux de neurones profonds ont la capacité d'adapter automatiquement les paramètre des fonctions de transformation). Ces auto-encodeurs servent à fournir un score d'anomalie aux événements analysés et aux caractéristiques qui les composent, ainsi qu'un vecteur (appelé représentation latente), qui est exploité pour regrouper les événements similaires. La structure du réseau est construite de manière automatique grâce à des composants génériques pour chacun des types d'attribut les plus communs dans les événements de sécurité (i.e., catégoriel, numérique, et chaines de caractères). Un exemple de structure est donné en Fig. 3

Adaptation à la dérive conceptuelle par apprentissage actif

Au sein d'un SI, le comportement des utilisateurs évolue continuellement. En particulier, les nouveaux comportements légitimes peuvent déclencher des alertes et d'anciens comportements peuvent cesser de toute manifestation à l'intérieur des données de supervision. En science des données, ce phénomène se dénomme "dérive conceptuelle" et il requiert d'entraîner les modèles continuellement. Cependant, un adversaire peut exploiter cet apprentissage en continu afin d'empoisonner le modèle progressivement en distillant des traces d'apparence bénigne de son attaque jusqu'à ce que la totalité de l'attaque soit considérée comme étant normale par le modèle (communément appelé "attaque de la grenouille ébouillantée"). Pour s'en prémunir, les données servant au ré-entraînement du modèle doivent être sélectionnées avec soin (i.e., uniquement les fausses alertes). L'approche proposée exploite la qualification d'alertes déjà effectuée par les analystes afin de sélectionner les données qui doivent être ré-insérées dans le modèle.

Dans le cas de la détection d'attaque inconnues, les analytiques doivent traiter en grande partie (>99%) des événements causés par des actions légitimes des utilisateurs. Par conséquent le taux de fausses alertes peut être élevé. Bien que le *clustering* aide à réduire le volume d'information à analyser, pour les SI les plus grands, ce n'est pas suffisant pour atteindre un nombre d'alertes acceptable. Afin de réduire ce nombre et fournir un maximum de contexte aux alertes pour accélérer le processus d'investigation, l'approche proposée repose sur la fusion des événements causés par les mêmes actions. Un algorithme permettant d'effectuer

cette fusion à l'échelle est proposé. Les analystes peuvent l'exploiter en fournissant la liste des variables pivots (i.e., attributs communs d'une source d'événement à une autre) qu'ils utilisent habituellement pour naviguer au travers des différents événements de sécurité pendant une investigation. Les événements ainsi fusionnés forment des ensembles de composition variable. La structure de l'auto-encoder doit ainsi être adaptée dynamiquement, pour chacun des ensembles d'événements (Fig. 4). En croisant l'information de tous les événements qui composent l'ensemble d'entrée, cet auto-encodeur dynamique est capable de détecter des anomalies en fonction de leur contexte (e.g., des événements normaux indépendamment, mais qui ne devraient pas se manifester ensemble).

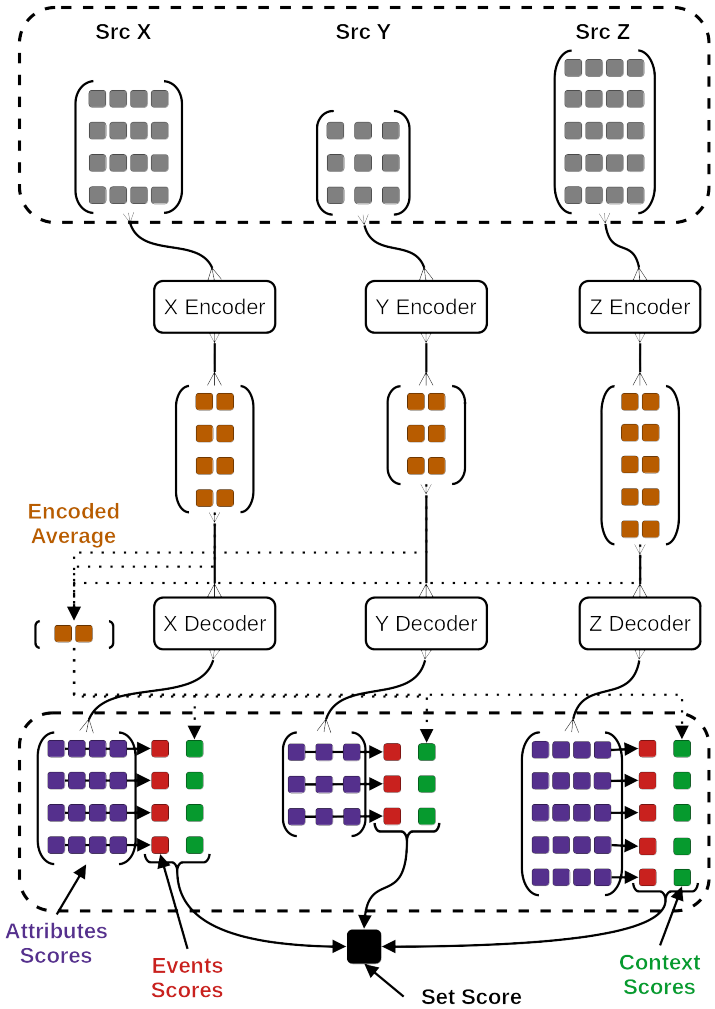


Figure 4 – Structure dynamique de l'auto-encoder en fonction de la composition des ensembles d'événements

En la combinant avec le *clustering*, cette approche permet de réduire le nombre d'alertes par deux ordres de grandeur. Étant donné que tous les événements liés à l'alerte sont présentés en même temps aux analystes, ils n'ont pas besoin de naviguer aux travers de multiples sources d'événements pour reconstruire le contexte derrière une alerte, ce qui réduit le temps d'investigation. Ceci rend l'approche viable pour de l'apprentissage actif (*active learning*), où la qualification des alertes à l'issue des investigations est utilisée comme moyen de rétro-contrôle du modèle (i.e., les faux positifs sont ré-insérés dans le modèle).

Constitution de jeux de données réalistes

La représentativité des jeux de données constitués au sein d'environnements de laboratoire est souvent limitée par la qualité de l'activité qui y est générée. En particulier, l'activité légitime des utilisateurs est la plus grande cause de faux positifs pour les environnements de production, mais la qualité de sa représentation est pourtant limitée au sein des jeux de données publics (e.g., activité réseau uniquement, des agents qui sont visibles dans les logs, etc.).

Afin de constituer des jeux de données plus réalistes pour l'évaluation de méthodes issues de la science des données, l'approche proposée permet d'automatiser la génération de l'activité légitime des utilisateurs dans une environnement de laboratoire. Cette approche repose sur un agent qui instrumente les machines de l'environnement au travers de la souris, du clavier, et de l'écran. Il devient alors possible de déporter cet agent à l'extérieur des machines instrumentées et par conséquent le rendre invisible du point de vue des outils de supervision. L'agent est conçu en plusieurs couches d'abstraction afin de permettre aux opérateurs de rapidement définir et mettre en œuvre des scénarios d'activité adaptables aux spécificités de l'environnement (e.g., différentes versions de logiciels, systèmes d'exploitation hétérogènes, etc.). Un orchestrateur se charge de coordonner un ensemble d'agents (un par machine), afin de générer de l'activité à l'échelle du système. Cet orchestrateur s'assure que chaque agent effectue la bonne action au bon moment sur la bonne machine (e.g., un utilisateur ne peut pas répondre à un e-mail qui n'a jamais été reçu).

Cette méthode a été employée afin de générer un jeu de données contenant sept jours d'activité et trois scénarios d'attaques complets. C'est ce jeux de donnée qui

a servi à l'évaluation des analytiques conçus au moyen de l'approche décrite dans cette thèse. Il sera mis à disposition afin de permettre la reproduction des résultats par des tiers et l'évaluation d'autres méthodes.

Perspectives

Au terme de ces travaux de recherche, quatre axes d'investigation majeurs ont été identifiés. En particulier, l'approche proposée consiste en la définition de composants qui sont requis mais non suffisant pour créer des analytiques de sécurité complets reposant sur des méthodes issues de la science des données. Il manque effectivement des composants requis par les analystes en cybersécurité (e.g., visualisation d'alertes). Également cette approche n'a été évaluée que sur un unique environnement et trois scénarios d'attaque. Bien que les attaques visant les modèles ont été prises en compte pendant la phase de conception, de plus amples évaluations de robustesse face à ces attaques sont requises. Enfin, des tests plus poussés pourraient mettre en évidence des problèmes de performance, aussi bien en termes de temps de calcul que de détection.

Introduction

Since the introduction of the Personal Computer (PC) to the mass-market in the late 1970's, human society have become overly reliant on Information Technologies (IT). Today, most companies operate Information System(s) and can be severely impacted by a loss of control over them. Due to this, in the past few decades, profitability of cyber-criminality has risen and multiple highly organized structures have emerged. In parallel, state actors have developed their offensive capabilities in the domain, and in addition to military objectives (e.g., the case of the StuxNet worm [55], which significantly hindered Iran's nuclear program), they can also pose a threat to major enterprises either through industrial espionage (e.g., the AQUATIC PANDA threat actor [3], which is believed to target industries in the governmental, information technology and telecommunication sectors), and through destructive operations that impact entire activity sectors (e.g., the Lazarus Group which is reportedly behind the attack against Sony Pictures Entertainment in 2014 [4]). Threat actors that are allegedly sponsored by or affiliated to nation-states are usually called Advanced Persistent Threats (APT). They are made of highly skilled and organized personnel with abundant funding and are therefore capable of maintaining stealthy access for a long period of time to conduct large-scale operations.

Defending against threat actors

To defend against these sophisticated cyber-criminal organizations and APTs, IT system operators define and enforce strict security policies to minimize the exposure of their systems and limit the impact of successful attacks. However, defining and refining perfect security policies that account for natural evolution of an IT system and the future threats that will target it, while not diminishing

3. See a report on AQUATIC PANDA here: `https://www.crowdstrike.com/blog/overwatch-exposes-aquatic-panda-in-possession-of-log-4-shell-exploit-tools/`

4. See a report on Lazarus Group here: `https://home.treasury.gov/news/press-releases/sm774`

crucial functionalities, is nearly impossible. Therefore, to react quickly (i.e., before damage is irreversibly done) to successful attacks that are bound to happen, it is mandatory to monitor the system for potential security breaches. For large entities, security monitoring and incident response are entrusted to Security Operation Centres (SOC) and Computer Emergency Response Teams (CERT).

In a SOC, security analysts configure detection systems to analyse security events and alerts whenever manifestations of known attack patterns have been detected. Often, these alerts can be triggered by legitimate behaviours. Therefore, analysts must investigate them to find the ones that are caused by attackers and are to be considered as security incidents. When a small subset of legitimate behaviours repeatedly triggers the same alerts, analysts can be quickly saturated and fail to detect small but important variations in the behaviours, that, in turn, prevent them from accurately qualifying a security incident as such. This phenomenon is usually called *alert fatigue*, and is one of the biggest problems faced by SOCs.

A CERT provides help to entities for handling security incidents response. To do so, upon receiving an incident case (e.g., from a SOC), security analysts in the CERT perform a thorough investigations of the security data (e.g., security events from monitoring, forensic analysis of infected machines, etc.) to uncover as much information as possible regarding the attackers. This information is exploited to remove the attackers from the impacted system (and prevent them from gaining back accesses), and it is also used to enrich a knowledge base containing intelligence on threat actors.

This Cyber Threat Intelligence (CTI) is made of Indicators of Compromise (e.g., IP addresses and domain names of attackers' infrastructures, malware signatures, etc.) as well as documented adversary behaviours. These behaviours are often referred to as Tactics, Techniques and Procedures (TTPs), where Tactics describe the attackers high level steps when conducting an attack, Techniques represent possible methods to perform an attack step, and Procedures are specific implementations of Techniques. In terms of detection, IoC are simple to detect in security data but are easily modified by attackers, while detecting TTPs require advanced monitoring strategy and careful configuration of the detection system, but take more time for attacker to completely modify (see Fig. 5).

Upon acquiring new intelligence on threat actors, a specific team of security analysts called threat hunters can look for evidence of these actors activity inside

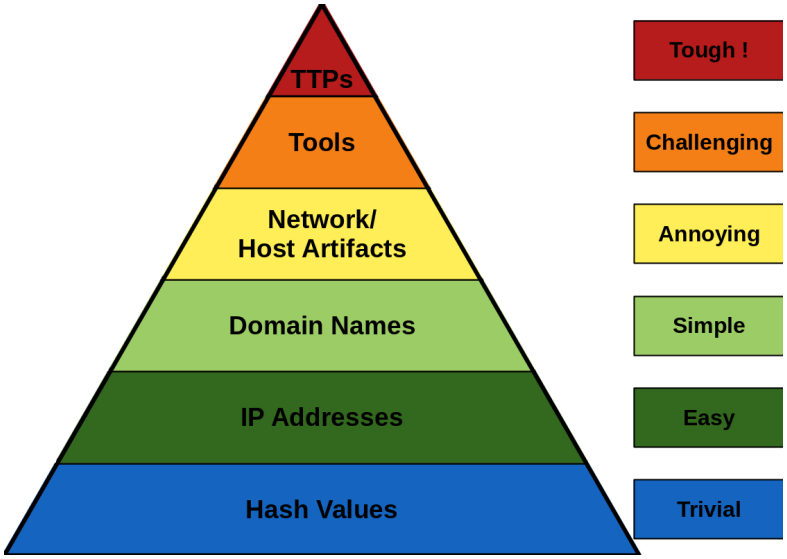


Figure 5 – Threat hunting Pyramid of Pain [13]

historical data. In a SOC, threat hunters will focus on detecting breaches they potentially missed, while in a CERT, threat hunter also aim at discovering novel attack patterns or variations of existing ones. In any case, threat hunters analyse massive amount of data and therefore need to rely on tools to accelerate and automate as many parts of the analysis as possible.

The case of data science for security operations

Data science aims at finding methods to extract valuable information and knowledge from data. However, while various aspects of security analysts work revolves around extracting information regarding security incidents from monitoring data, data science methods, and especially Machine Learning (ML)-based methods, are only scarcely adopted within SOCs and CERTs. In fact, the application of data science to support security analysts faces three main challenges [116, 92, 3].

First, most security analysts lack the required expert knowledge in data science to design and operate machine learning based methods. Similarly, data scientists are often not accustomed to the specific requirements of cyber security operations. For instance, monitoring data comes from multiple heterogeneous sensors that provide varying levels of visibility on attackers behaviours with varying volume

and noise, which requires complex pre-processing to apply ML models to this data. This knowledge gap between data scientists and security analysts exacerbates the black-box nature of ML sometimes perceived by the latter. Furthermore, the intuitive skills acquired through experience by security analysts is often difficult to formalize in such a way that data scientists can design relevant methods to support security operations.

Second, contrarily to most application domains in data science, attackers will actively try to circumvent the defenders' tools. Therefore, special care should be taken when designing these tools to ensure it does not introduce additional vulnerabilities that attackers can exploit. For ML models, it means accounting for adversarial perturbations (i.e., modifications to the data specifically meant to evade the model), poisoning (i.e., modifying the training datasets to degrade the performance of the models), but also consider the whole data acquisition pipeline: if attackers can modify monitoring data, they can severely impact defenders' capabilities.

Finally, acquiring representative datasets to experiment with data science inspired methods for cyber security operations is difficult. In fact, security data contains sensitive information and personal data. While data anonymisation can reduce the exposure of individuals, to be effective (i.e., keeping anonymity even when correlating data), they degrade the representativeness of the datasets. Therefore, SOCs and CERTs are often legally prevented, both through contracts and regulations (e.g., GDPR in Europe) to share real life security monitoring datasets. Shareable datasets should therefore come from a laboratory environment with fake users. Generating realistic user activity is mandatory to provide representative datasets, and is still an open challenge.

Outline

In this thesis, we propose data science inspired methods to support security analysts. Specifically, we focus on security analytics (i.e., automated data processing to support analysis) that exploit anomaly detection and clustering to highlight suspicious behaviours and reduce the volume of information that is presented to analysts.

To improve results' explainability and reduce the black-box nature of data science methods perceived by security analysts, we put security analysts at the

centre of the analytics design process. To do so, we draw inspiration from state-of-the-art deep learning methods applied to other domains (e.g., natural language processing) to help automate data pre-processing, and thus lower the need for data science knowledge.

To account for the natural evolution of legitimate behaviours, the models are trained continuously. To prevent attackers from poisoning the models by distilling gradually traces of their attack while staying under the detection threshold (known as the frog-boiling attack), we propose to use an active learning approach, where analysts provide feedback to the model by annotating false positives (i.e., legitimate behaviours that triggered alerts) that should be reintegrated inside the model. To lower the number of alerts to investigate and reduce the investigation type, we rely on event fusion and clustering to contextualise alerts.

Chapter 1 details the context and state of the art in analytics for security monitoring. The deep learning approach to handle heterogeneous security events is described in chapter 2. Chapter 3 explains the active learning approach that relies on event fusion and dynamic neural networks to provide contextualised anomaly detection and clustering capabilities. Finally, in chapter 4, we describe the method we propose to automate user activity in order to generate a dataset, and we assess our approach using this dataset. The document ends with a conclusion discussing our results and providing perspectives for our work.

Chapter 1

Context

1.1 Introduction to security monitoring

Monitoring is a core part of modern information system security. This section gives an introduction to security monitoring starting by explaining why it is a necessity to effectively defend a system. It gives an overview of what aspects of the systems can be monitored, why they should be, and how it can be done. The organisation of the technical and human resources behind today's security operations is also given, with a strong focus on Security Operations Centres (SOC). We conclude with the notion of monitoring strategy, and gives a high level overview of how such a strategy should be constructed.

1.1.1 The need for security monitoring

Definition of an attack

The security of an information system revolves around three fundamental properties: confidentiality, integrity and availability (also known as the CIA triad). Confidentiality means that only authorised people, resources and processes can access the information. Integrity means that the information should not be modified without authorisation (whether it is accidental or not). Availability means that any authorized user can access the information whenever needed. An intrusion is therefore an attempt to compromise any of these three properties, and an attack is an intrusion attempt.

Protecting an information system

The protection of an information system consists in ensuring that actions performed by users and processes on the system are properly authorized. But more

importantly, it also means that authorized users can use the system as they require. The corner stone of IT system protection is therefore to authenticate users and processes, and to control accesses to the required resources of the system. Access Control refers to the management and enforcement of these accesses. In summary, protecting an information system relies on specifying what users and services are authorized to do or not, and granting or refusing these authorizations. The specification is part of the access control policy of an information system. Properly defining this policy is essential to the security of an information system. Preventing users from doing something they should be able to do impacts availability. Allowing them to do actions they should not perform may impact confidentiality and/or integrity of the system. In practice, from the network point of view, the protection of an IT system consists in isolating machines in their own sub-networks and interconnecting these sub-networks as required using the appropriate network equipments (e.g., firewalls, manageable switches, etc.). From the applicative point of view, the informations regarding users, machines, processes and their respective rights on the systems are most often stored in a directory (e.g., Active Directory, LDAP, etc). Users, machines and processes can then be authenticated and authorizations to resources of the system are granted or denied based on the attributed rights.

The limits of access control

In practice, exhaustively specifying what users can and cannot do is close to impossible. Besides, such a specification needs to be constantly updated as the needs of the users and the capabilities of the system evolve. There is always a trade-off between the exhaustiveness of the access control policy and the cost of defining and maintaining it.

Furthermore, enforcing an access control policy is not trivial. In fact the authentication methods of users can be compromised (e.g. passwords can be stolen), and adding additional authentication factors (e.g., biometry) has a non-negligible impact on usability and cost of maintenance of the system. Besides there is a risk the user itself might be compromised, whether it is intentional (e.g., a user that wants to harm the company) or not (e.g., an attacker successfully tricking a user).

Finally, systems have flaws that an attacker can exploit. When these vulnerabilities are known (e.g., publicly disclosed on sites like MITRE CVE [24]) actions

can be taken to mitigate them (e.g., by updating the system), but many of these vulnerabilities are not documented and can be exploited by attackers.

Therefore additional measures need to be taken to verify that the access control policy is properly enforced and to ensure that attacker(s) haven't managed to bypass it. This is done by tracing and recording the activity of the systems to look for indicators of a violation of the policy or of a compromise by an attacker. This process is called security monitoring.

Protection and supervision are two complementary pillar in cybersecurity. Monitoring for illegitimate accesses to the system can help improve the access control policy by identifying flaws in the policy or in its enforcement. On the other hand, if the policy is too permissive, discerning illegitimate accesses from legitimate ones is impossible.

1.1.2 Collecting monitoring data

The data collected during the monitoring process is meant to detect and characterise the activities that are a risk to the monitored system's security. It should contain information that can be used to reconstruct the steps and the causes behind a security incident (i.e., a verified illegitimate access to the system which impacts its confidentiality, integrity and/or availability). Having a good understanding of the origin and the evolution of a security incident facilitates the definition of an incident response plan.

Definition of security events and alerts

Security monitoring consists in recording the activity of the IT system in the search of illegitimate activity. Actions performed on a system change its state, and information regarding these changes can be recorded inside logs in as events. Security events are the events that may be of interest to identify illegitimate actions. To describe what has changed, events have variable parts that we call attributes. For security events, most attributes are either numerical, categorical or text. Numerical attributes are real numbers with a notion of order (e.g., 1 is bigger than 0.1), categorical are attributes with a finite set of values with no ordering (e.g., *red* is neither lower or higher than *blue*), and text are strings of printable character with a specific syntax and semantic and a virtually infinite number of different

values (e.g., the name of a file).

An alert is triggered when one or more events indicate that an activity that is considered illegitimate has been detected. For example, a process creation event may indicate that a word processing software executed a command line interpreter, which triggers an alert.

Network security monitoring

IT systems essentially rely on networks of interconnected machines. While performing their normal operations, these machines communicate with one another. Similarly, the attackers will interact with the network's machines, and traces of their presence can be seen inside network communications. Network monitoring consists in analysing these communications in order to record and detect the ones that can be attributed to attackers. In practice, this task is performed by Network Intrusion Detection Systems (NIDS), that look for evidence of known misuse, or by network sensors that analyse network protocols and record events indiscriminately. As an example, Snort [1] and Suricata [2] are commonly used as NIDS, and Zeek [3] and the IPFIX/NetFlow protocol are frequently employed to record network activity.

Network security monitoring is relatively easy to deploy as it can be done passively, by capturing network traffic on the system (e.g., through SPAN links on network switches or network TAPs). It is also likely that attackers will perform actions that are visible from the network point of view (e.g., exfiltrating data to their servers on the Internet). However, on complex network architectures, the multiple layers of Network Address Translation (NAT) and proxies can make it difficult to extract the exact source and destination machines from the network capture. Capturing the traffic at the right spots inside the system is therefore critical for network security monitoring to cover the desired perimeter and record useful information. Besides, using encrypted communication has become the norm for both legitimate and malicious uses, considerably reducing the detection performance of NIDS when not using SSL/TLS inspection, which is harder to deploy.

1. `https://www.snort.org/`
2. `https://suricata.io/`
3. `https://zeek.org/`

File analysis

Data is commonly stored on machines as files. By extension, executable code is stored as files. Attackers are likely to drop malicious code if the form of files (malware) during their attack. Analysing files to look for suspicious patterns (static file analysis) can therefore help detecting malware, which might indicate the system has been compromised. This analysis can also be performed by executing the file in a controlled and sealed environment to verify how it behaves and detect suspicious intents (dynamic file analysis).

Traditionally, static analysis is performed by anti-viruses software that look for characteristic features of known malware (signatures) when analysing files. Often, small modifications of malware are enough to bypass this analysis. Dynamic analysis is performed within sandboxes that look for suspicious behaviour. It is much more difficult to modify a malware behaviour without impacting its functionalities, making dynamic analysis more robust than static one. However, as it requires running the executable code in real-time, this analysis requires much more time, and sandboxes often chose to stop the analysis after a certain amount of time, which can be exploited by malware to avoid detection. Also, sandboxes simulate interaction by the users, and this simulation can be detected as such by malware, which can chose to stop their execution or perform only benign activity.

File analysis systems can be deployed directly on machines of the system (e.g., mail gateways, workstations, etc.) and perform their analysis as the files are written on the machine, or they might be deployed as a dedicated platform for analysts and security tools to submit files to.

Endpoint security monitoring

While network monitoring has been deployed in priority in the last few years, endpoint level monitoring has seen an increase in popularity for production environment. In particular, the generalized use of communication encryption strongly limits the visibility of network probes, or requires the complex deployment of traffic interception and decryption. On the other hand, actions performed on the system by attackers can be monitored at the Operating System (OS) level. For example, the execution of a malware can be logged as any other process execution event, and its activity can also be recorded by the OS.

This endpoint monitoring often consists in recording the execution of specific system calls (e.g., the ones that are executed by specific and or privileged services only), and checking system integrity (e.g., ensuring no configuration file has been corrupted, that access to the registry are as expected, etc.). This host-based monitoring is performed by Host Intrusion Detection Systems (HIDS), also known as Endpoint Detection and Response (EDR).

Although endpoint monitoring provides a far better visibility on the attackers actions, they are harder to deploy than network based solutions. Specifically, deploying, managing and securing these highly privileged tools on a complete IT system can be challenging. Besides, this specific type of sensor usually cannot be deployed on Industrial Control Systems (ICS) and Internet of Things (IoT) devices, due to the overhead endpoint detection tools carry, which is incompatible with the specific availability requirements of such systems. Finally, once the attackers have gained sufficient privileges on the system, they can simply disable endpoint monitoring, or temper with the information that is collected.

Application layer monitoring

Any IT system hosts a collection of applications (e.g., specific software, network services, etc.) that are used by the users. Often, production-grade applications have the ability to log their activity (e.g., for debugging purposes, for traceability, to monitor system health, etc.). These applications can also be subverted or targeted by attackers. Misuses might be recorded by applications and therefore, application logs can be valuable for security monitoring.

Nevertheless, each application can chose to implement its own logging system. Extracting and normalizing valuable information from an unknown application is complex. Due to this, on large systems with potentially numerous custom applications, it is close to impossible to monitor all these applications.

Digital Forensics

Real-time monitoring (i.e., files, events and alerts collection from network, endpoints and applications) is aimed at reducing the mean time to detect security incident. It should provide information that can be used to define the nature of the attack and the impacted assets. It is however not possible to predict all the

attackers potential actions and capabilities. Therefore, the monitoring strategy is bound to have visibility gaps.

On the other hand, attackers are unlikely to reinvent their complete methodologies and tooling with every new attacks. Specifically, developing offensive tools and training for new attack methodologies requires time and specific skills, which is incompatible with the cost imperative of cyber-criminal organisations and can also be of concern for state actors. In fact, many threat actors acquire tooling from third parties, and often exploit legitimate adversary simulation tools (e.g., Cobalt Strike [4], Metasploit [5], etc.). Therefore, many tools and methodologies are reused from one operation to another (even from different threat actors), making it possible for the defenders to detect the attackers at some point (even if it means detecting them after damage has been done). At this point, it is possible to investigate supposedly impacted assets to confirm (or infirm) the attacker's presence, and gather evidences (for legal purposes). This investigation can also lead to the discovery of novel attack methodologies that can later be used to detect similar attacks.

While real-time monitoring can focus on a restricted view of the system, digital forensic can have a wider scope as it focuses only on impacted machines and not the whole system. The forensic procedure consists in taking snapshots of the state of the machines (e.g., running processes, memory dumps, disk image, etc.) and analysing them. However, advanced attackers will try to erase the trails they leave on the systems, limiting the effectiveness of forensic measures.

1.1.3 The Security Operations Center

The security monitoring capabilities of large companies are often handled by a dedicated entity (either by an internal unit or by an external service provider) that is called the Security Operation Center (SOC), and is the combination of technical and human resources. A SOC is in charge of detecting security incidents and qualify them (i.e., decide whether the incident can be considered as an attack or not). SOC analysts investigate security incidents to provide as much context as possible to the incident response team. This team is often part of another entity called Computer Security Incident Response Team (CSIRT) or Computer

4. `https://www.cobaltstrike.com/`
5. `https://www.metasploit.com/`

Emergency Response Team (CERT), and is in charge (among other things) of performing the digital forensic investigation. While the methodologies followed by these two units have similarities, in this thesis, we focus on analysing data gathered as part of the real-time monitoring process and therefore, we will focus on the SOC context.

The Security Information and Event Management system

While a SOC uses multiple tools (e.g., file analysis platforms, threat intelligence platforms, etc.), the SIEM (Security Information and Event Management) is a central part of these tools. Its role is to centralize, analyse and ease the investigation (e.g., through visualization) of security events and alerts. It is in charge of performing the correlation process which consists in the following steps [112]:

1. **Normalisation**. Alerts are coming from multiple sources and their format can greatly differ from one source to another. Normalising alerts eases the comparison of alerts across sources.
2. **Enrichment**. Inside the monitored system, the various monitoring sensors only have a restricted view of the system as a whole. The enrichment phase aims at completing the information gathered by sensors using external information (e.g., reputation system for IP addresses, using DHCP logs to associate internal IP addresses with the corresponding machine, etc.)
3. **Alert fusion**. This step creates meta-alerts by regrouping alerts coming from multiple sensors that describe the same action (e.g., the network IDS sees a potential exploitation of a vulnerability on a server, and the host IDS alerts on the consequences of this exploitation).
4. **Alert verification**. Partly due to their restricted point of view and their lack of context on the monitored system, intrusion detection systems are likely to generate false alarms. Correlation tools have a better overview of the system and can therefore filter out some of the obvious false positives.
5. **Multi-step correlation**. This step reconstructs the different steps of the attack to detect known patterns.
6. **Global analysis**. The focus here is to prioritize the alerts based on the potential impact on the system.

The SIEM also simplify the investigation of security events by providing the analysts with searching and visualization capabilities.

SOC Analysts

In a SOC, analysts are in charge of investigating the security alerts resulting from detected suspicious behaviours in order to qualify them. This investigation consists mostly in:

1. Escalating an alert as an incident, if it is not a false positive (i.e., an obviously legitimate behaviour);
2. Attributing the incident handling to the right analyst;
3. Gathering context around the incident (i.e., searching for security events that may be linked to it);
4. Determining criticality of the incident and its potential impact.

In most SOC, there are two levels of analysts that handle incidents:

1. Level 1 analysts perform a preliminary investigation of incoming alerts according to pre-defined scenarios. They aim at removing obvious false positives and re-attributing the handling to the suited level 2 analyst.
2. Level 2 analysts conduct a more thorough analysis of the alert and its context to determine the criticality and, in case of a verified incident, its potential impact on the system.

These analysts have to handle a large amount of alerts as fast as possible (especially for level 1 analysts). This may lead to a phenomenon called *alert-fatigue* that causes recurring alerts to be ignored or mishandled as the analysts get used to handling similar alerts mechanically and are increasingly likely to miss subtle details.

In some SOC, a third level also exists, and is in charge of gathering intelligence on attackers (Cyber Threat Intelligence, CTI) and searching for novel attacks traces in events (Threat Hunting). This level is however more represented in the CSIRT, where threat hunting and CTI analysts explore security data to uncover novel attack methodologies and provide actionable intelligence to the SOC.

Cyber Threat Intelligence

In a SOC, the Cyber Threat Intelligence (CTI) team gathers intelligence that might be of interest for the SOC operations. This often consists in aggregating CTI information from vendors, open source feeds, CERTs, etc. and processing this intelligence to make it relevant for the systems monitored by the SOC and easily exploitable through the SIEM. This helps detecting emerging threats more quickly and prioritising the limited time analysts can spend on tasks that are the most relevant.

In addition to that, CERT and CSIRT also focus on producing intelligence. They often start from known incidents and try to find the tools and actions of the attackers that were not accurately characterized by the monitoring tools. In practice, this intelligence results from a combination of the results from log analysis, malware analysis (including reverse-engineering), and from the study of the attackers organizations (their motivations, the infrastructures they rely on, the skills they exhibit, etc.). This helps understand what attackers can do and who they might be targeting. This intelligence is meant to be shared either publicly, or with select partners (e.g., SOC monitoring systems likely to be targeted, other CERT, etc.).

Threat Hunting

The Threat Hunting team is in charge of detecting potentially novel threats that may have been missed by the real-time monitoring systems. In a SOC, Threat Hunting is focused on gathering and analysing the context surrounding the detected incidents to improve future detections (e.g., modify detection rules, add ones that would detect the attacks earlier, etc.). In a similar manner, they investigate the leads provided by the CTI. In a CERT or a CSIRT, the Threat Hunting team works in conjunction with the CTI team to analyse and document emerging threats.

In both cases, threat hunters start working based on hypothesis (e.g., has this system been compromised using these techniques ? knowing the attackers have compromised some assets, what is their usual next step ?) that they try to confirm or infirm by analysing the monitoring data. To do so, they tend to focus on the analysis of fine grained security events (e.g., recorded processes execution and

system calls, IPFIX/netflow or full packet capture, etc.). Therefore, they are likely to rely on data mining tools (e.g., search engines, prioritisation tools, etc.) to skim through the massive amount of data faster.

1.1.4 Defining a monitoring strategy

Fine grained monitoring data (e.g., indiscriminate system call level monitoring, full packet capture, etc.) generates large amount of data. Although this data contains extremely valuable information for threat hunters, storing and analysing it requires equally large amount of both hardware and human resources. As these resources are scarce, the monitoring strategy should balance visibility and volume of collected data, and the monitoring tools need to be adapted accordingly.

The monitoring strategy describes what information to collect and in what parts of the monitored system. It is derived from the identified assets of the system to protect and their associated risks. The monitoring strategy is adapted in accordance with the resources available to analyse the collected data. The configuration and position of the different sensors depend on this strategy.

Visibility vs. volume

A good monitoring strategy prioritise sources of events that brings high visibility on the identified risks while limiting the volume of data it generates. For example, monitoring for the use of all the possible methods an attacker can employ for inter-process communication (e.g., file accesses, sockets, named pipes, etc.) generates massive amount of data (hundreds of thousands of events per machine per day). While it helps better characterise malicious activity, it is difficult to effectively exploit to detect this activity (i.e., there is a vast amount of legitimate inter-process communications and they are not easily discernible from the malicious ones), and thus it only provides marginally better visibility. On the other hand, monitoring process execution greatly improves visibility. Indeed, suspicious process execution is almost certain to happen when the attacker gains access to the system (e.g., unknown executables, unusual combination of legitimate executables, etc.). It is also very likely to occur multiple time during the rest of the attack. Therefore, considering that it generates reasonable amount of data (a few thousands of events per machine per day), the monitoring of process execution has a

higher priority than the monitoring of inter process communications.

Defence in depth

Defence in depth is a core concept of information systems security. Its base principle is that security measures should be redundant, in case the attacker manages to bypass some of them. For example, in addition to auditing the code of an application to discover and patch vulnerabilities, restricting its privileges to a minimum will limit the damages the attackers can do if they discover a vulnerability the audit team did not.

The same principle can be applied to security monitoring. Indeed, attackers will actively try to evade detection to limit the risk of being evicted from the system by the incident response team. These evasions can range from employing methodologies for which the defence has low visibility to impersonating legitimate applications and actively disabling monitoring tools. Besides, even without the intervention of attackers, these tools can fail to record the activity they are supposed to (e.g., due to a bug, a misconfiguration, etc.). Having some redundancy in the visibility provided by sensors can therefore improve resiliency against evasion techniques and limit the impact of a failure of a monitoring tool.

As an example, endpoint monitoring solutions can be disabled once the attackers have gained enough privileges on the endpoint. Relying solely on these solutions exposes to a high risk of complete loss of visibility. It is therefore important to have balance and redundancy between endpoint, network, and application level monitoring.

1.2 Tactics, Techniques and Procedures and Data-driven Security

This section details a core concept of modern attackers behaviour modelling, namely Tactics, Techniques and Procedures (TTP). We then explain the process of defining data analysis methods to help the hunting team look for these attackers behaviours. Finally, we conclude by presenting recent academic work aiming at analysing and representing the attackers behaviours, using data mining techniques.

1.2.1 Introduction to Tactics, Techniques and Procedures

Tactics, Techniques and Procedures (TTP) are a way to describe the behaviours of the attackers that operate on compromised information systems. Tactics describe short term goals of the attackers (e.g., execute code on the system, gain higher privileges, etc.), techniques represent how the attackers reach these goals (e.g., exploit a vulnerability to gain higher privileges, use command line interpreter to execute scripts, etc.), and procedures describe the specific implementation of the techniques (e.g., exploit a specific CVE, use PowerShell, etc.). MITRE ATT&CK [108] is the most famous knowledge base of TTPs and is a common resource in Security Operation Centers. Notably, ATT&CK provides the Enterprise matrix [6] recapitulating common TTPs targeting enterprise IT systems (an extract is provided in Fig. 1.1).

TTPs are often used by the threat hunting team to formulate hypothesis and find evidence of exploitation of techniques inside the security events [27]. Detecting adversaries based on TTPs often requires visibility at specific points inside the IT system, which generate much more data than traditional detection strategies that focus on detecting specific procedures using Indicators of Compromise (IoC). This data is often organized into data sources [7], which represents high level categories of security events and the information they provide (e.g., process execution, network flows, firewall logs, etc.).

6. Available here: `https://attack.mitre.org/versions/v10/matrices/enterprise/`
7. Example: `https://attack.mitre.org/datasources/`

Figure 1.1 – Extract of MITRE ATT&CK Enterprise matrix (version 11.0)

1.2.2 Designing Analytics for TTP Hunting

Security analytics are data analysis processes that aim at detecting evidence of the actions of an adversary. The MITRE corporation provides a repository describing analytics to hunt for common techniques [8]. Daszczyszak et al. [27] also gives an in-depth overview of the design process of threat hunting analytics. Fig. 1.2 provides an overview of this process which we summarize in the following steps:

1. **Data selection**: Define the TTPs to look for and the required data sources;
2. **Enrichment and feature selection**: Enrich sensor data with information from CTI, asset base and external sources (ex: add GeoIP, normalise data according to a data model, derive IP addresses from hostname and DHCP logs, etc.). Choose the data sources attributes that are useful to detect targeted techniques (e.g., detecting processes spawning command line interpreter requires the executable path of the process and its parent);
3. **Define base analytics**: Write the detection conditions and filter out expected False Positives (e.g., SSH network flows coming from a machine that is not in the administrator zone can be an indicator of lateral movement);
4. **Tune analytics for the system**: Test False Positive Rate on available data and adapt the analytics until a satisfactory FPR is reached (e.g., some users may legitimately SSH into other machines). If no suitable detection conditions are found, go back to step 1;
5. **Investigate/correlate remaining hits**: define what additional data should be gathered to verify if the hits are attributable to malicious activities or not, and provide context to the alert;
6. **Automate detection**: deploy the analytics to raise alerts on real-time data;
7. **Maintain ruleset**: Normal evolution of the system behaviour (e.g., new users, machines, software, etc.) may require adaptation of the analytics (step 4).

8. MITRE CAR: `https://car.mitre.org/`

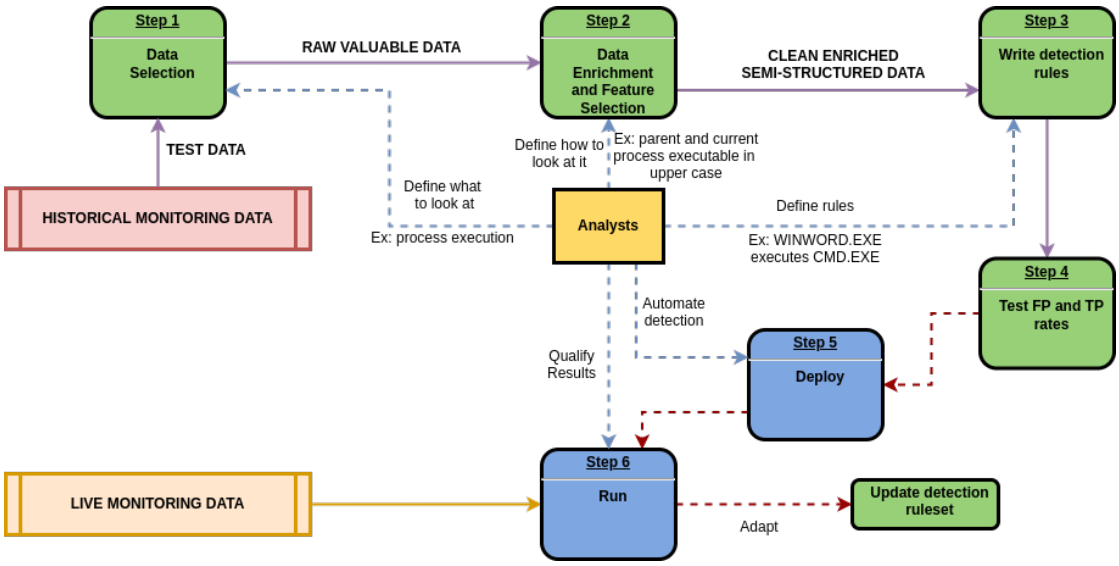


Figure 1.2 – Threat Hunting process

1.2.3 Exploiting and visualizing security data

An important step of the analytics design process is to define what additional data should be gathered to provide context to the detection. This context can help analysts verify if the detection is truly attributable to a malicious activity, and if it is the case, facilitates the handling of the incident.

To provide this context, provenance tracking (also known as information-flow tracking) have been employed. It consists in tracing the entities and processes that have influenced the state of the information (i.e., produced, delivered and/or modified it). It is often represented as a Directed Acyclic Graph (DAG), where nodes correspond to entities that hold or manipulate information (e.g., files, processes, etc.) and edges to events that indicates a potential manipulation of information (e.g., process P has written data to file F). Brogi et al. [16] proposed terminAPTor which use provenance graph to highlight dependencies between the traces left by the attacker. Similarly, with SLEUTH, Hossain et al. [51] combined provenance graph and rules to detect and reconstruct attack scenarios. Building upon SLEUTH, Milajerdi et al [80] proposed HOLMES, which provides a higher level representation that is easier to analyse, and filter out false positives using anomaly detection. This anomaly based detection is also used by Han et al [43] in Unicorn. With ANUBIS, Anjum et al. [5] used supervised machine learning to detect malicious traces. Hassan et al. [44] use TTP-based detection from an

Endpoint Detection and Response (EDR) system and contextualise it with provenance tracking. Recent work by Xosanavongsa et al. [122] focused on providing a formal definition of causality for security events (i.e., causal relationships between events), to accurately contextualise detection. Causal dependencies is then used to compute the provenance.

While provenance tracking is effective at providing context, the necessary information is unlikely to be found inside the events collected by SOC, although authors worked on making it available to production systems [10, 88]. An approximation of provenance or causality using available data can be used to correlate events in graph. With HERCULE, Pei et al. [91] use a graph structure and supervised machine-learning to detect strongly correlated events that may indicate attacks. Leichtman et al. used this graph structure for its visual representation [69], and combined it with anomaly detection to detect attacks and investigate attacks [68, 70].

1.3 Machine Learning for Security Analytics

This sections introduces the core topic of this thesis, the application of data science in support of security monitoring. We summarise state-of-the-art of the anomaly detection for security, and highlight the limitations that this work tries to address. Specifically, we focus on the possibility for security analysts (who have no knowledge in data science) to use these methods for their specific needs, and make the results they provide easy to understand. We also focus on the possibilities for the attackers to evade these detection techniques (i.e., mimicry and data poisoning methods) and how we can propose methods that are more robust against these types of evasion. We conclude by explaining the operational use cases we aim at addressing in this thesis.

1.3.1 Introduction to ML for security analytics

Tuning the detection conditions of an analytics for the target system specificities is a time-consuming process. ML algorithms have the ability to build models directly from data without the need of human assistance. Applying them to the fine tuning of security analytics could therefore reduce the time analysts dedicate

to the adaptation of the analytics to the monitored system.

Before any model can be applied, it is necessary to prepare the data to be processed. This consists in selecting the right data to solve the desired problem, removing the noise (e.g., malformed data points, unwanted outliers, etc.), and performing feature engineering. This last step consists in selecting the right characteristics of the data points to model (a.k.a., feature selection), and transforming them into a representation that is suitable for the model (usually, vectors of real numbers). Once data is ready, ML models are often prepared in three phases:

1. A training step, where the model parameters are adapted (or learned) to perform as good as possible on an initial set of data.
2. A testing step, which consists in validating that the model performs accurately on data that has not been used for training.
3. An inference phase, where the model is deployed and applied to new data.

We can distinguish two major kinds of learning strategies, namely supervised and unsupervised learning. In supervised learning, the training and testing datasets are annotated (or labelled) with the expected output of the model. The model is then trained to output values as close as possible to the provided annotations. Supervised learning is useful to derive decision logic automatically from expert knowledge (transmitted via the labels) without writing code explicitly. However, it requires a data annotation phase from human experts. In cybersecurity, supervised learning is often applied to malware analysis [111], mainly because it is possible to collect malware and goodware that can be found on many different IT systems. While authors have applied supervised learning to intrusion detection systems [25], it is very difficult to use it on production environment because every IT system is unique. Therefore malicious and legitimate activity will exhibit different manifestation from one system to another. This would require analysts to manually annotate legitimate and malicious behaviour that are specific to the monitored system, which is unrealistic. Also, while supervised ML models are often tolerant to small variation of the data, they are ineffective against sufficiently novel data (e.g., targeted attacks, new malware families, etc.).

On the other hand, unsupervised learning consists in training models on data with no label. The model usually learns an approximation of the structure of the data. This structure can be used the find similarities between data points to regroup them (clustering), or find data points that share very little characteristics

with the rest of the data (outlier or anomaly detection). When applied to security analytics, unsupervised learning can help reduce the investigation time by regrouping similar alerts, or filtering out recurring false positives by highlighting anomaly detections. While anomaly detection models have the ability to detect novel attacks, they often come with high False Positive Rates as anomalies are not necessarily attributable to malicious behaviour. Also, while a characteristic (e.g., an attribute of an event) can be a highly discriminating feature that helps form clusters, this does not mean that the identified clusters will be useful for the analysts. For example, while two processes from two different machines can share the same PID (process identifier), it is unlikely that these two processes are similar. Similarly, a PID that was never seen before might be found to be abnormal by the model, but has no real meaning from a security point of view. Therefore, unsupervised machine learning greatly benefits from expert knowledge during the design and feature selection phase to perform accurately.

It is also possible to combine an unsupervised model with a supervised one (semi-supervised learning). In this case, the unsupervised model is used to learn a simplified representation of the structure of the data and the supervised model is trained on a few labelled data points relying on this simpler structure.

1.3.2 Machine Learning for anomaly detection

In this thesis, we focus on security analytics for intrusion detection. Considering that analysts do not have time to annotate data manually and that we want to be able to detect novel attacks, we will focus on unsupervised models.

The application of machine learning to anomaly detection for different data types have been studied by researchers for a few decades now. Kriegel et al. [15] computes the anomaly score based on the distance with the nearest neighbours, with a high distance to the other points indicating a potential anomaly. Pang et al. [85] proposed a nearest neighbours based method, that scales to larger datasets (several millions of events) by computing the pairwise distance between random samples of point instead of the whole dataset. Ester et al. [39] proposed DBSCAN, an approach that identifies high density of points as clusters and classify points inside low density region as anomalies. However, these approaches are sensitive to a high dimensional data [11], which can manifest in security events, depending on the feature engineering step (e.g., many attributes, attributes that are transformed

into multiple variables, etc.). As a consequence Kriegel et al. [61] proposed a work that scales to large number of attributes in data. All the methods mentioned above rely on a notion of distance that needs to be defined specifically for the problem at hand. That can prove difficult, especially for complex data structures (e.g., the distance between two strings, two events with heterogeneous attributes types, etc.)

A variant of Principal Component Analysis has been also used by Pascoal et al. [87] to propose an approach that is robust to noise in the training dataset (e.g., a few attack traces in the normal data). Scholkopf et al. [103] proposed one of the most used algorithm for anomaly detection by training SVM (Support Vector Machines). Data Mining techniques have been used by He et al. [47] to measure the level of anomaly of a transaction. This type of approach have been extended by Akoglu et al. [2] to limit the number of frequent pattern used to compute the anomaly score. Pattern mining algorithm requires categorical data as input, and therefore a suitable transformation of the input data should be found for numerical data.

The use of Bayesian Networks [90] has been tested by Wong et al. [120] to perform anomaly detection. This type of approach permits also to diagnose and explain a detected anomaly. However, Bayesian methods requires to identify the most likely probability distribution for the events, which can be challenging.

The algorithm Isolation Forest proposed by Liu et al. [73] was applied to security by Ding et al. [35]. This permits to classify quickly the abnormal activities. This technique does not require any kind of normalisation on numerical variables, but it requires categorical values to be transformed into numerical values and cannot handle text values without specific transformation methods.

Hawkins et al. [45] propose a method based on neural networks to compute anomaly score. This approach is called Replicator Neural Networks (RNN). With the rise of Deep Learning and more specifically Deep Neural Networks (DNN), RNN have regain interest in the form of deep auto-encoders and a robust variant of the algorithm has been proposed by Zhou et al. [124]. Mirsky et al. [82] relies on an ensemble of auto-encoder to improve the robustness and accuracy. Due to recent advancements in deep learning, auto-encoders can be adapted to various kind of data (e.g., text, time-series, images, categorical, numerical, etc.). However, such a network is computationally intensive to train and is best suited for high volume of training data. Veeramachaneni et al. propose an active learning based approach

for large scale security monitoring [115]. The authors combine a Principal Component Analysis approach, auto-encoders and a distance-based approach for anomaly detection, but they still require complex feature selection and transformation for each event sources.

Security event analysis can be seen as a special case of log analysis. He et al. [46] recently provided a comprehensive review of anomaly detection in logs. Essentially, the anomalous activity identification process can be decomposed in three major steps, namely, parsing the logs (i.e., going from unstructured logs to structured events), extracting interesting features and finally using an anomaly detection algorithm. Bertero et al. [12] and Du et al. [36] drew inspiration from natural language processing to take into account the context of the events. Debnath et al. [28] proposed a generic framework to automate the parsing phase of log analysis. However, both approaches fail to analyze the attributes of the events which make them less effective for security logs. For example, they would not detect a connection to an automatically generated command and control domain because they would see a normal DNS request followed by an HTTP(S) connection and would not see that the domain name seems unusual.

Shen et al. [105] and Liu et al. [74] propose methods that rely on embedding of security-related information. The former's objective is to model the evolution of exploitation methodology for known vulnerability, and is therefore more related to cyber threat intelligence than security monitoring. The latter aims at detecting attacks and relies on complex sets of rules to build graphs, which is hard to adapt to new types of security events and threat models.

1.3.3 Challenges of ML for security analytics

While data science and machine learning method could in theory provide benefits for multiple problems encountered by security operators (e.g., filtering false positives, help hunting for targeted threats, etc.), adoption of these techniques in security operations faces multiple challenges [116, 3]. Mainly, the plurality of use cases, and the omnipresent notion of adversaries that actively tries to evade detection requires the selection and adaptation of the right set of techniques. To do so, it is needed to experiment with datasets that represent accurately the diversity and specificities of the cyber security use cases. The publicly available datasets do not cover all the use cases. In addition to this dataset problem, more recent work

by Sarker et al. [102] have highlighted a list of open issues faced by data science in the cyber security domain. We summarize the issues that we want to address in this thesis.

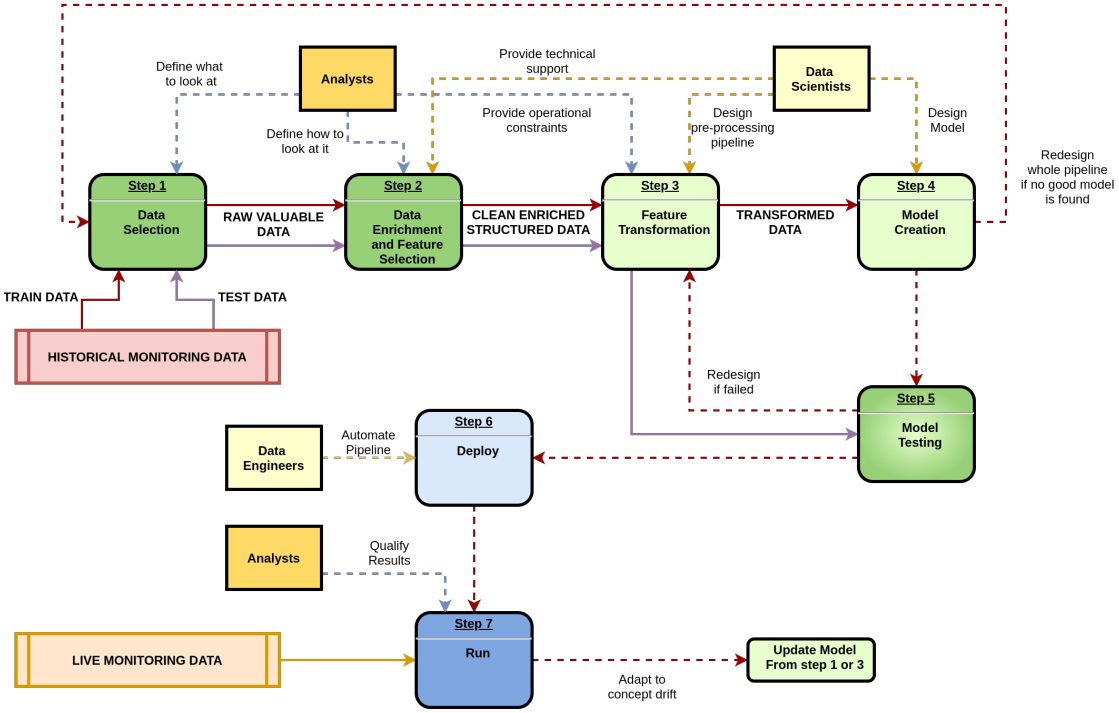


Figure 1.3 – Generic ML process for security analytics

Introducing machine learning (ML) algorithms to security analytics requires significant changes to the design process 1.3. Indeed, while the data selection step only requires collecting more data to serve as a baseline to train the models, the remaining steps relies on knowledge from the data science domain.

First, the feature selection and data enrichment step may need to be adapted to the capabilities of the available ML algorithms. Specifically, feature types that are commonly handled by rule-based analytics (i.e., categorical data and strings) are not trivial to process with ML algorithms, and may require specific enrichments. Second, most ML algorithms deal with numerical features and often behaves better when the input space has specific mathematical properties [54] (e.g., normalised inputs, linearly separable clusters, etc.). That introduces the need for a feature transformation step. Its goal is to ensure that the data provided as input to the models has the required properties. Consequently, the choice of algorithm for the models has a strong impact on the analytics as different algorithms will require

different feature transformations and will have different performances. Finally, deploying a ML based processing pipeline for security analytics may require specific tooling such as distributed computing framework.

While this data science knowledge is a hard requirement for implementing ML-based security analytics, SOC analysts rarely have this expertise. Similarly, data scientists with a good understanding of the specificities of cybersecurity operations are also difficult to find. This gap between the two domains emphasize the "black box" effect of machine learning algorithms perceived by security analysts.

Additionally, ML-based analytics tend to have better False Negative Rates (FNR), at the cost of higher False Positive Rates (FPR): less attack goes under the radar but with more false alarms. This is especially true for anomaly-based detections because anomalies can be caused by legitimate (albeit rare) activities. The higher FPR combined with lower explainability of the results (i.e., diagnosis of the cause of the alert is more difficult) increases the alert fatigue for analysts.

Finally, mitigating detection evasion techniques specifically targeted at ML model have been an active research topic for years [9, 7, 26, 52, 97]. For anomaly detection, these evasion techniques can be regrouped in two categories:

— Mimicry attacks [118, 62, 86], whose principle is to modify attacks to look as normal as possible;

— Poisoning attacks [14, 53, 19], that consists in introducing attack traces inside training data. The frog-boiling attack [58, 18, 60] is a special case targeting anomaly detection models that learns continuously.

1.3.4 Target use cases

To reinforce the explainability of the results and limit the dependency on data science expertise, we propose an ML-based analytics design process (Fig. 1.4) that puts the security analysts at the centre to benefit from its expertise, while limiting the need for data science knowledge. We do so by automating the parts of the traditional ML analytics design process (Fig. 1.3) to make it more similar to the rule-based hunting analytics process (Fig. 1.2). We cover three different anomaly detection use cases by relying on deep learning models. These use-cases are designed to be of increasing complexity. They can be viewed as maturity milestones to accompany security analysts towards taking full advantage of ML-based approach in their operations.

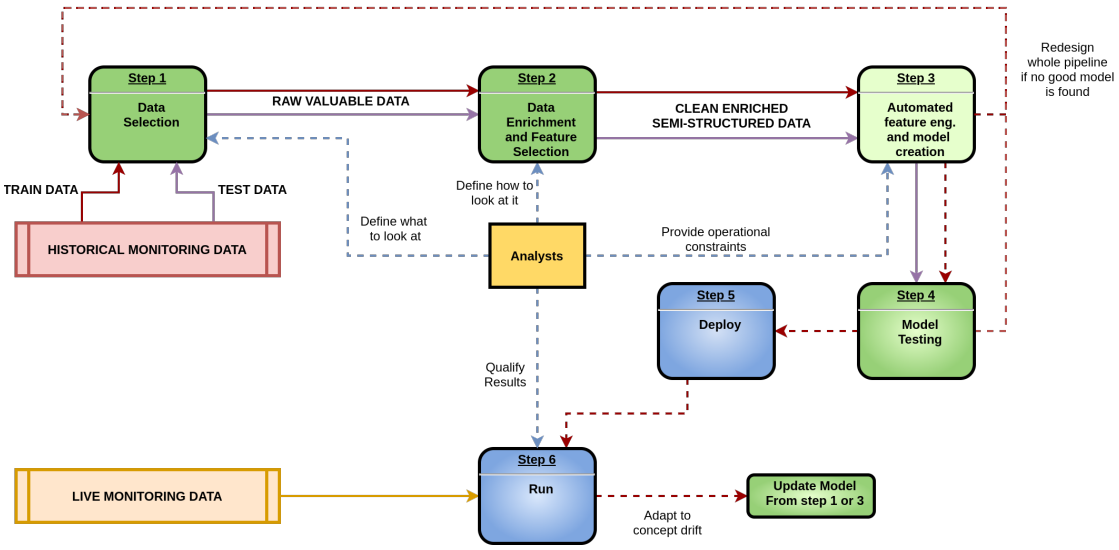


Figure 1.4 – Target process

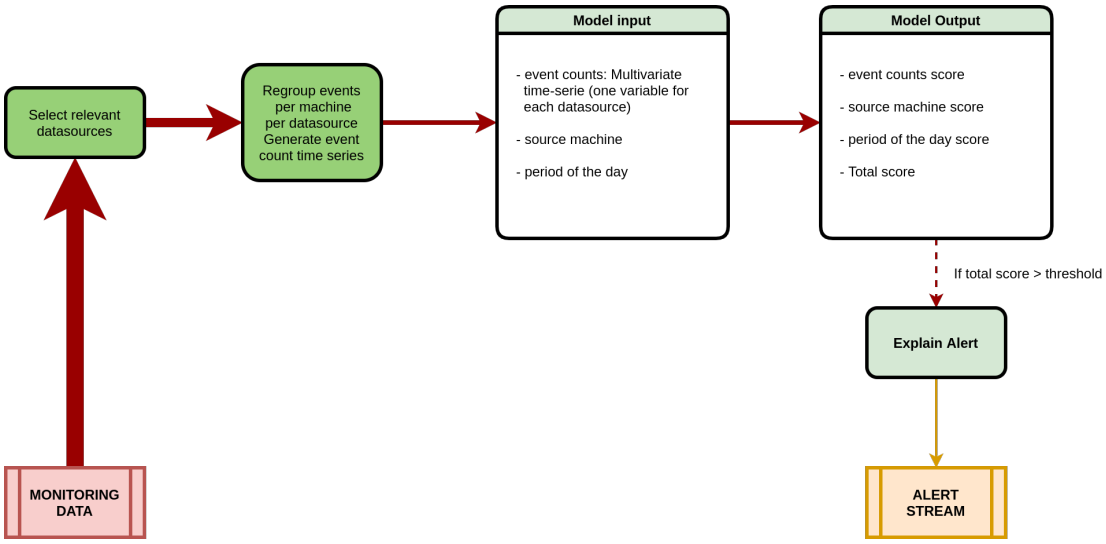


Figure 1.5 – Base ML for security analytics (already abstracted in some Commercial Off-The-Shelf)

The first use case (illustrated in Fig. 1.5) is to detect unusually high or low volumes of events. Indeed, the monitoring tools are configured to generate events upon detection of suspicious activity. An unusual increase of detected suspicious activity can be an indicator of an attack. Also, monitoring tools can stop functioning correctly (e.g., an attacker disables them), which in turn would cause a lower volume of events than what is expected. Here, using ML allows to automatically

set the high and low thresholds between which the volume of event is normal, while also taking into account the time (e.g., less events are expected during non-working hours), and the source of the events (e.g. firewall logs often generate more events than anti-viruses). This use-case is typically handled by SIEM tools featuring User and Entity Behaviour Analytics, and is a time-series anomaly detection problem. Our process simplifies the design of the analytics, especially for multivariate time-series with different scales and sampling rate (e.g., correlating the number of EDR alerts every 10min with the number of network connections per minute). Indeed, the DL model can find correlations between multiple inputs and handle the scaling of these inputs without human intervention. To facilitate investigation, it also provides indications regarding what caused the anomaly.

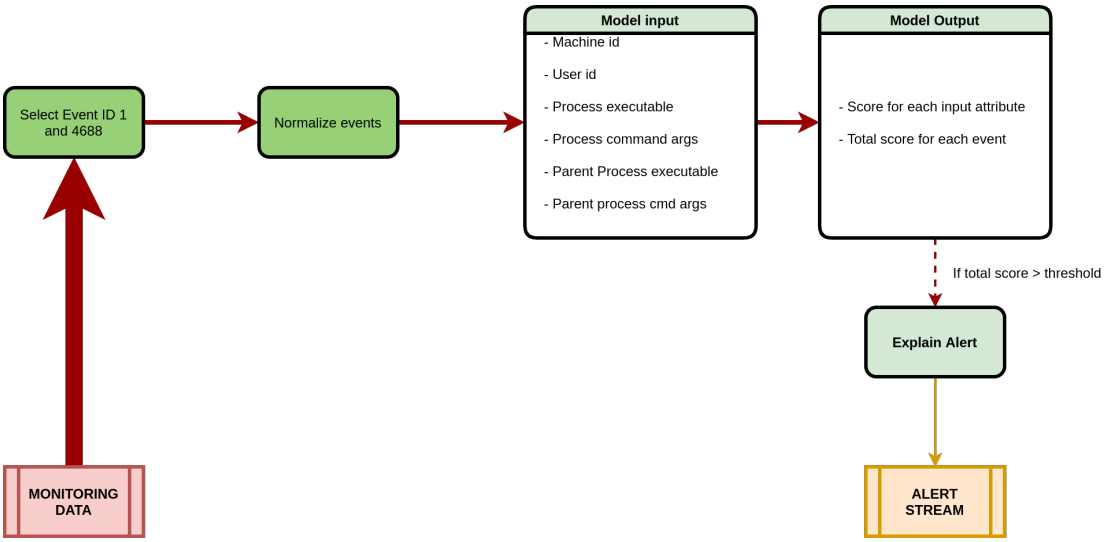


Figure 1.6 – Ability to easily design ML based analytics for all data sources (process execution here)

The second use case is to detect anomalies in any source of security events. Fig. 1.6 illustrates anomaly detection in process execution events (e.g., commands that are rarely or never used on machines, anomalous parent/child relations, etc.), but the same process can be applied to other data sources (e.g., netflow, firewall logs, opening of sockets by processes, application logs, etc.). Handling heterogeneous security events with ML algorithms requires careful design of the data transformation and enrichments processes, as categorical and text attributes (the majority for security events) cannot be processed as is by traditional ML algorithms. The DL models we employ automatically handles data transformation for

categorical, text and numerical variables. This limits the need for data scientists, as they are usually in charge of choosing and configuring the right data transformation methods.

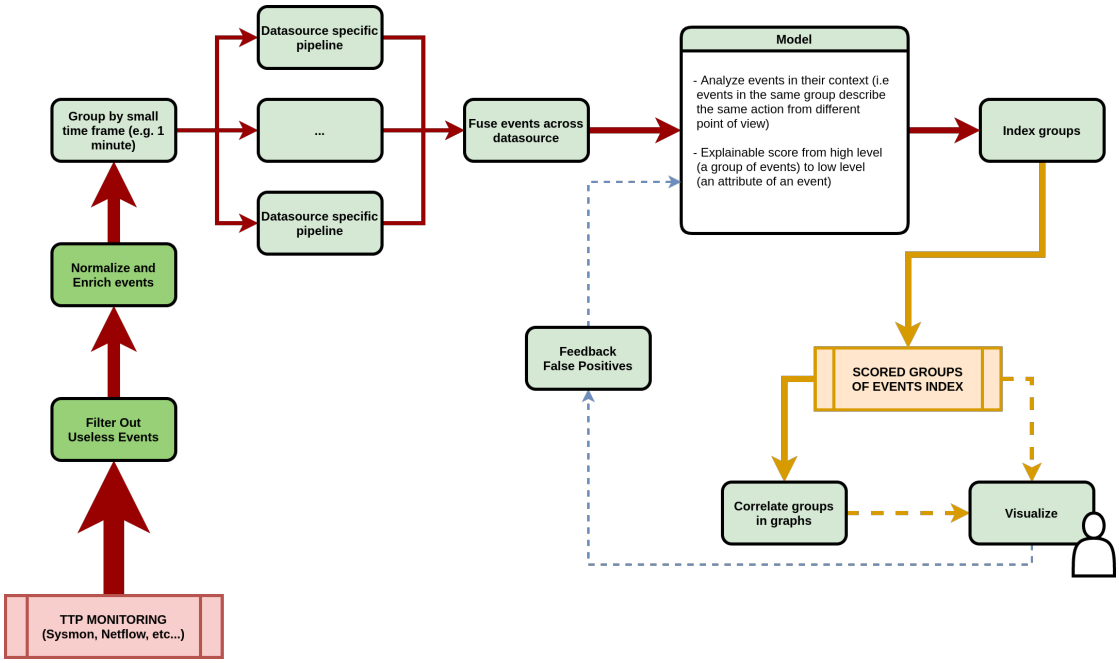


Figure 1.7 – Generalize ML usage for all datasources with contextualized alerting, visualization and human in the loop

The last use case consist in detecting contextualised anomaly by crossing information from multiple sources of events. To provide as much context as possible to each anomaly and facilitate investigation, we regroup events from multiple data sources that describes the same actions and detect anomalous groups of events (Fig. 1.7). Our approach also provides a way to find similar alerts which can be used to accelerate handling of recurrent false positives, or find previous incidents with the same characteristics. This method is designed to integrate with SOAR (Security Orchestration Automation and Response), and its results can be visualised with common security visualisation methods (e.g., graph visualisation). To handle massive amounts of security events, special care has been taken to accelerate processing with parallel and distributed computing. The objective is to improve situational awareness by allowing finer grained security monitoring strategies with manageable volumes of alerts. The anomaly detection models are continuously trained with the false positives found by analysts in an active learning fashion

(i.e., the number of elements to annotate is reduced to avoid burdening the operators). This helps mitigate frog-boiling attacks considering that only the anomalies that have been verified by a human expert can be used for re-training the models. Therefore, attacker will also need to bypass human verification to poison a model.

1.4 Summary

In this chapter, we introduced the concept of security monitoring and its complementarity with the IT system protection measures. Defining, enforcing and maintaining a perfect security policy that protects the system from all current and future threats is impossible. Instead, current best practices consist in enforcing security policies that do not impact the system's usability, and trace and record the system activity to detect illegitimate access or violation to the security policy through security monitoring.

For large systems, security monitoring is entrusted to Security Operation Centres (SOC), that collect security events generated by multiple sensors inside the system (e.g., network monitoring, endpoint detection tools, etc.). In a SOC, the SIEM normalises and correlates these events to detect and alert on suspicious behaviours using correlation rules. Human analysts investigate these alerts to determine whether the detected behaviours are legitimate or not. Illegitimate activity that triggered an alert are escalated as security incidents and investigated to determine the potential impact and causes.

Incident response is often performed by Computer Emergency Response Teams (CERT). In a CERT, incident responders collect additional data to extract evidence of attackers activity. A thorough investigation of this data is performed to remove the attackers from the system and prevent future breach. This data is also analysed to look for novel attack behaviours and produce Cyber Threat Intelligence (CTI).

In both SOC and CERT, security analysts need to analyse massive amounts of data to detect and investigate adversary behaviours. To relieve pressure on analysts and reduce the time to detect attackers activity, automated security data analysis procedures (a.k.a., security analytics) have been employed. In their current form, security analytics often consist of scripts that detect specific adversary TTP (Tactics, Techniques and Procedures).

To simplify the fine-tuning of the analytics and allow more advanced attack patterns detection (including novel attack detection), ML-based approaches have been proposed. In this thesis, we focus on unsupervised anomaly detection and clustering, due to their ability to detect attacks without prior knowledge and data. We find that the low adoption rate of these approaches in operational context can be attributed to four challenges:

— Configuring ML-based approaches require extensive knowledge in data science, which security analysts often lack;
— ML-based approaches are often viewed as black-boxes that provide results with high amount of false positives that are difficult for analysts to qualify and exploit;
— Their robustness against targeted attacks (i.e., mimicry and poisoning attacks) can open the detection system to additional evasion techniques;
— Datasets that capture the complex and evolving nature of security monitoring do not exist.

In this thesis, we propose methods to address these challenges. In particular, we propose an approach to allow security analysts with limited knowledge in data science to design and exploit security analytics that rely on anomaly detection and clustering in operational use cases. This approach puts the analysts at the centre of the analytics design process to limit the black-box effect, being able to take feedback from them. We also describe a method for generating representative datasets, and use it to produce a dataset that we use to assess our approach.

Chapter 2

Handling Heterogeneous Security Events

In this chapter, we describe how we can detect anomalies and perform clustering on heterogeneous security events. Our approach relies on neural networks auto-encoders. Their structure is automatically created based on the source of events to analyse. This source is described by the list of attributes to analyse and their type (categorical, numerical or string). The design is focused on minimizing the amount of data science knowledge that is required for the security analysts to adapt the methods to their specific needs.

2.1 Introduction

Security monitoring of information systems requires to log events happening during the execution of processes at system level, the exchange of data via the network or the warnings issued by applications. In addition to event logging, Intrusion Detection Systems can produce alerts that are likely to be the consequence of an attack. Due to the huge number of events produced, even if a monitoring strategy has been clearly defined, it is difficult for the analysts to detect what are important events from the security point view, i.e., what are the events that are symptomatic of an intrusion inside the system.

The current practices consist in collecting all security events in a SIEM (Security Information and Event Management) solution. This solution is able to correlate information included in multiple events in order to recognize known attack patterns. Despite the definition of highly accurate correlation processes [66], this treatment still requires to manually define static correlation rules. Thus, the effectiveness of the detection relies on the ability of the analysts to define a complete set of correct correlation rules for known attacks with low False Positive Rate.

This set of rules should be updated continuously to take into account the newly discovered threats, and remove old rules that detect obsolete threats (e.g., modifications to the monitored system security policy). This is a tremendous task, and it is insufficient to emphasize all attack steps, especially for emerging threats. As a consequence, a lot of anomalous events stay hidden to the analyst.

During the threat hunting process, analysts rely on prioritization tools to highlight the most anomalous events and identify misbehaving entities in the system. If necessary, a more thorough forensic analysis of these entities can be performed. After this analysis, they should be able to produce a set of Indicators of Compromise (IoC) and correlation rules.

As a way of prioritizing events, we propose an approach that associates an anomaly score to each attribute of an event [1]. These scores are then combined to provide a global anomaly score to the event. A higher score means that the event has a higher probability of being a consequence of an unusual behaviour. This approach relies on the use of Artificial Intelligence mechanisms, more specifically, neural networks auto-encoders (see Sec. 2.2).

We focus on the ability of our approach to be applicable to most types of security events (i.e., network, system and application events). We aim at automating and abstracting as much as possible the complex task of feature engineering that is required to transform attributes into compliant inputs for the chosen algorithms (e.g., for strings, choice between feature hashing, one-hot encoding, TF-IDF, etc.). In fact, security analysts often lack the required knowledge to perform this task, and data science expertise is very rare within SOC and CERTS. Instead, the method described in this chapter only requires analysts to identify events attributes as being a numerical, categorical or string variable. This makes it easier to adapt to new category of security event.

The approach also provides a way for auto-encoders to compute a cluster identifier for each event. This identifier is used to easily and accurately regroup similar events. This clustering reduces the volume of redundant information presented to analysts.

By combining the anomaly detection and the clustering capability with the relative ease of configuration of our approach, we aim to facilitate the adoption

1. Attributes are the fields of an event. Connection duration, source IP address, number of bytes received are examples of attributes for a network event. See Sec. 1.1.2

of ML-based approaches by security analysts. Specifically, we think anomaly detection can accelerate the detection engineering process (especially the fine-tuning part). The combination of clustering and prioritisation via the provided anomaly score can also help threat hunter and incident responders explore the security data to find traces of novel attack patterns sooner, and investigate security incidents faster.

In Sec. 2.2, we provide basic concepts around neural networks and auto-encoders. The input pre-processing methods that are meant to transform heterogeneous attributes into vectors that can be processed by the neural networks are given in Sec. 2.3. Sec. 2.4 describes the building blocks of the auto-encoder, and how its structure is automatically constructed for each type events to analyse. Finally, Sec. 2.5 focuses on the use of such a model by security analysts.

2.2 Auto-encoders neural networks

This section covers the basics about how a neural network works. It introduces the basics of deep learning, which heavily rely on choosing the adequate structure of neural network according to the modelled data. The notion of constraints, that allows guiding the training of a network, is presented. We focus more specifically on the subtype of neural networks that our approach is based on, the auto-encoders.

2.2.1 Introduction to neural networks

A neural network is a composition of functions with trainable parameters. These functions are called the neurons of the network [98] and the most simple form of neuron consists of the weighted sum of the neurons input and an optional term (called bias). A non-linear function, called activation function, is applied to this sum. The weights of the neurons and the bias term constitute the trainable parameters of the network. Generally, neurons are organized in successive layers, where each neuron in a layer takes as input the output of the neurons of the previous layer. A neural network is trained to minimize an objective function, which often contains at least a measure of the divergence between the expected output and the one effectively predicted by the network. To solve this minimisation problem, the most used algorithms derive from the gradient-descent optimisation

method [100], which requires the gradient of the objective function for each parameter of the network. Therefore, in this document, we generalize the neural network definition to a composition of differentiable functions, where parameters are trained using a variant of the gradient back-propagation method, which relies on the chain rule (i.e., differentiation of composition of function) to compute the gradient of the objective function for each parameter of the network. Specifically, in a feed forward neural network (i.e., organised in successive layers), the computed gradient of layer $n+1$ is used to compute the gradient of layer n, hence, gradient is *back-propagated* through the layers, starting from the last one. Commonly, batches of elements from the training data are presented successively to the network, and the weights are updated with each batch. This is especially useful for large datasets because each elements of a batch can be handled in parallel.

$$f_{(W,b)}(X) = \sigma(\sum_i (w_i x_i) + b) \tag{2.1}$$

Figure 2.1 – Equation of a simple neuron. With $W = (w_0, w_1, ...w_n)$ the weights of the neuron, b the bias, σ the activation function and $X = (x_0, x_1, ...x_n)$ the inputs.

Structures for neural networks

Historically, neural networks have been used to map vectors from high dimensional space (e.g., value of the pixels of an image, complex set of extracted features, etc.) into more easily comprehensible values (e.g., a class, a small vector, etc.). With the rise of deep learning, researchers moved from using only fully connected layers of formal neurons (Eq. 2.1) to incorporating layers specifically adapted to the structure of the data provided as an input to the model (e.g. 2-D convolution for images, recurrent networks for text, etc.). For tasks that require the network to output in a high dimensional space (e.g., image or text generation), the structure of the network is often organised in two parts, an encoder which maps inputs into a latent representation, and a decoder which takes the latent representation as an input and outputs value into the expected vector space. The auto-encoder is a special Encoder-Decoder model for which the output space is the same as the input space.

Constraints on neural network parameters

While the gradient back-propagation method combined with the huge number of parameters prevents from predicting exactly what and how the network will learn, it is possible to guide the training of the network. This is done by adding constraints to the desired parameters in the form of terms added to the objective function. For example, if the output of a layer is constrained by the term defined in equation 2.2, the layer will tend to output smaller values to minimize this penalty.

$$\lambda \sum_i x_i^2 \tag{2.2}$$

Figure 2.2 – L_2 regularization. λ is the weight of the penalty and x_i are the inputs

2.2.2 Basics on auto-encoders

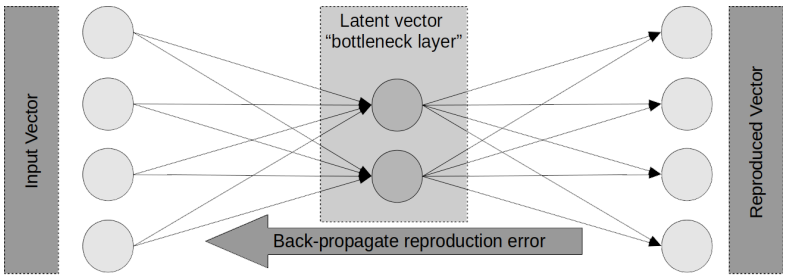


Figure 2.3 – Overview of an autoencoder

Auto-encoders (Fig. 2.3) are a particular structure of neural networks that are trained in an unsupervised way (i.e., without the expected output value in the training dataset) and made of an encoder that maps input vectors to a low dimension representation (also called latent space) and a decoder that reconstructs the original input vector from the latent space.

The input vectors are vectors containing numerical values (integers, floats or vectors of integers or floats). For anomaly scoring, the auto-encoder is first trained on normal data to compress and decompress the input vectors with as little loss of information as possible (i.e., an approximation of the identity function). Then, an inference phase takes as an input events produced during the monitoring process,

to which the auto-encoders estimated identity function is applied. As an output we obtain a result that can slightly differ from the input. We name this difference "reproduction error", and we use this difference to estimate the deviation of the input from the inputs that were learnt. More specifically, since the auto-encoder is biased towards normal events, the reproduction error is higher for anomalous events.

During the security monitoring process, an information system produces a huge number of heterogeneous events. These events can for example be extracted from the operating system (e.g., system calls), from network (e.g., connections, protocols, network IDS alerts, etc.), or from specific applications (e.g., web server requests logs). Moreover, different types of operating systems can be used (e.g., Windows and Linux systems), generating different formats of events. Finally, within a single source of events, the variable attributes have heterogeneous type that can be considered as either numerical variables (e.g., a file size, a duration, etc.), categorical variables (e.g., a protocol identifier, a file type, etc.), or string variables (e.g., content of a script, HTTP User-Agent, etc.).

Considering that an event that is the consequence of a normal behaviour on one system might be a sign of an intruder in another, our objective is to create normal behaviour models for each system. Models need to be able to process heterogeneous attributes and be adaptable to the various security events that are available for each system. Using these models, an anomaly score is computed for each event.

In addition to the anomaly score computation, an auto-encoder is able to produce a representation of its inputs, called latent representation, in a vector space that has the desired properties for commonly used clustering algorithms (e.g., DBSCAN, K-Means, etc.). By exploiting this latent representation, it is possible to provide a cluster identifier to each event, later used to regroup similar events in clusters (2.4.5), and thus to lower the volume of information analysts have to analyse (i.e., at most a few elements for each cluster instead of all of them).

Usually, security analysts are in charge of refining the detection capabilities for the monitored system. Therefore, for the approach to be exploitable in an operational use case, its configuration should be possible with the knowledge and skills that are already available within security analysts teams.

Due to their ability to handle heterogeneous attributes, we choose to adapt neural network auto-encoders to compute an anomaly score (2.4.6) for security

events. This includes the use of generic blocks (Sec. 2.4) and pre-processing functions (Sec. 2.3) for numerical, categorical and string variables. Using these generic components, it is possible to limit the configuration step to the definition of the attributes to analyse and their type (which is in line with security analysts capabilities).

2.3 Pre-processing Model Inputs

This section describes the various pre-processing steps that are applied to the inputs of our auto-encoders. These functions are meant to be generic and adaptable automatically on the training data. When choosing the attributes to analyse and specifying their types (categorical, numerical or string), the right transformation function will be implicitly chosen, and its parameters adapted.

2.3.1 Input Transformation Algorithm

As explained in subsection 2.2.2, the auto-encoder takes as an input a vector of numerical values. The goal of the input transformation process is to transform the vector containing the value of the attributes of an event into a vector suitable as an input for the auto-encoder. This transformation process is presented in Alg. 1.

There are three possible types of variables that can be taken as an input to our auto-encoders. The first type, like any machine learning algorithm, is numerical variables. The second one, the categorical variables, are variables whose values belong to a finite set and cannot be mathematically ordered (e.g., username "Bob" is neither superior or inferior to username "Alice"). Finally, raw strings are considered as sequences of categorical values.

2.3.2 Normalising Numerical Attributes

For numerical variables (integers and floats), taking raw values as an input is not the most efficient way of learning the normal distribution of numerical attributes [54]. We propose as a remediation to normalise the numerical values from a set of floats into a reduced interval (e.g., the set $[0, 1]$). This transformation is produced by the **ScaleValue** function in Alg. 1.

Algorithm 1 Event Attribute Vector Transformation

```
function TransformCategory(attribute, KnownPatterns, KnownCategories)
    for all pattern in KnownPatterns do
        if matchPattern(pattern, attribute) then
            attribute ← pattern
            break
        end if
    end for
    if attribute in KnownCategories then
        return value ← KnownCategories[attribute]
    else
        return value ← KnownCategories[DefaultValue]
    end if
    return value
end function
function TransformString(attribute, vocabulary, variant)
    tokens ← TokenizeString(attribute, vocabulary)
    switch variant do
        case GPU
            return tokens
        case CPU
            return GetTokensLogCount(tokens)
        case CPULowLatency
            return size(tokens) / size(attribute)
end function
function Transform(event, params)
    Vector ← ∅
    for all i in Size(event) do
        attribute ← event[i]
        p ← params[i]
        switch attribute.type do
            case categorical
                Vector[i] ← TransformCategory(attribute, p.patterns, p.categories)
            case string
                tokens ← TransformString(attribute, p.vocabulary, p.variant)
            case number
                Vector[i] ← ScaleValue(attribute, p.q90)
    end for
    return Vector
end function
```

In the context of anomaly detection, outliers with extreme values can reside in normal data. To limit their impact on the normalisation process, we find the 90^{th} percentile Q_{90} inside the training dataset (i.e., 90% of the normal values are below Q_{90}). The transformation then consists in dividing the input by Q_{90}.

If the values of the attribute grow exponentially, as suggested by Kaastra et al [54], the result of the transformation will be the logarithm of the initial value divided by the logarithm of Q_{90} (Eq. 2.3).

$$f(x) = \frac{log(x)}{log(Q_{90})} \tag{2.3}$$

2.3.3 Normalising Categorical Attributes

To handle categorical variables, we draw inspiration from *word2vec* [79], a commonly admitted standard for Natural Language Processing (NLP). The goal of this technique is to map each word in a continuous vector space (i.e., vectors of floats) based on its context (other words appearing in the same sentences). This mapping is generally called an embedding. It allows to treat natural language and perform, for example, the recognition of semantics of the words in sentences. We use the following analogy: an event is a sentence and its attributes are the words composing it.

The neural network will optimize the embedding function based on the other attributes of the given event, that will represent the context of the transformed attribute. This context allows to determine if the category of an attribute is normal in a given context.

In practice, a categorical embedding layer of a neural network takes as an input a category identifier (i.e., an integer) that represents a vector filled with as much 0 as the total number of categories, except for a 1 for the corresponding identifier. When the total number of categories is large, a proportionally large number of parameters needs to be optimized. To reduce the induced computational complexity, we propose the use of regular expressions to map every string matching the same pattern to the same category identifier. For example, if an application prefixes the name of its temporary files with a random combination of 8 lower case letters and numbers, it can be interesting to consider all the files in the application folder that matches this pattern as the same category. We can map them to the

same identifier using a regular expression like `^\/tmp\/app\/[0-9a-f]{8}\.tmp$`. However, as usual, the regex should be carefully chosen to prevent an attacker from bypassing them.

For a given value of an attribute, if the category is known (or if it matches a predefined regex), it returns the corresponding identifier. In case the category was never encountered before (frequent in the context of anomaly detection), it returns an integer corresponding to the category "Unknown". This corresponds to the function **TransformCategory** in Alg. 1.

2.3.4 Normalising Raw String Attributes

To capture the complete information of some attributes in security events, it is interesting to handle them as natural language. As an example, command line arguments of processes have their own strict syntax (changing a single character may lead to different results), as well as a specific semantic (different combinations of arguments lead to different results) [36]. With our auto-encoders, a raw array of integers can be passed as an input to model these syntax and semantic.

Initially, we implemented the very simple approach of transforming each character of the string as its UTF-8 code (i.e., an integer between 0 and 255). However, the deep learning algorithms used to model the syntax and semantic of raw strings require a lot of computing power, especially during the training phase, with a quadratic time complexity w.r.t. the size of the sequence. In the NLP field, this problem is partially addressed by decomposing sentences into tokens. Current state of the art [104, 64, 63] consists in identifying frequent sequences of characters and using them as tokens. For example, in English, 'us' and 'ing' are quite frequent and therefore, the word 'using' could be transformed into a two tokens word instead of five characters.

Drawing inspiration from [104] and pattern mining algorithms, we propose a simple algorithm to find frequent sub-strings inside string attributes, and build a vocabulary out of these patterns. Due to the potentially high number of events in the datasets, we choose an approach easily implementable in a map-reduce fashion.

1. **MAP**: Decompose each element into patterns by iteratively and greedily aggregating bi-grams (set of two successive characters). For example 'secure'

is decomposed into [(se), (ec), (cu), (ur), (re), (secu), (cure), (secure)] and 'security' into [(se), (ec), (cu), (ur), (ri), (it), (ty), (secu), (curi), (urit), (rity), (securi), (curity), (security)]

2. **REDUCE**: Regroup patterns and count them. Using the same example: the corpus ['secure', 'security'] becomes [((se), 2), ((ec), 2), ((cu), 2), ((ur), 2), ((ri), 1), ((re), 1), ((it), 1), ((ty), 1), ((secu), 2), ((curi), 1), ((cure), 1), ((urit), 1), ((rity), 1), ((securi), 1), ((curity), 1), ((secure), 1), ((security), 1)]
3. **FILTER**: Keep only the patterns that have the best potential to compress the data. This potential is calculated by multiplying the frequency of the pattern by its size minus 1. For example, [((se), 1.0), ((ec), 1.0), ((cu), 1.0), ((ur), 1.0), ((ri), 0.5), ((re), 0.5), ((it), 0.5), ((ty), 0.5), ((secu), 3.0), ((curi), 1.5), ((cure), 1.5), ((urit), 1.5), ((rity), 1.5), ((securi), 2.5), ((curity), 2.5), ((secure), 2.5), ((security), 3.5)]

Finally, to avoid having unknown tokens, every character in the UTF-8 code is added to the vocabulary. This way, even if no frequent sub-strings can be found in the string, it can still be encoded, albeit with no gain in the size of the string.

This algorithm is not designed to be an accurate sub-sequence mining algorithm. For instance, it will not find some frequent patterns, especially odd-sized ones, and cannot take a gap constraint into account. However, it's both easy to implement and reasonably scalable. It only requires a map-reduce-filter operation for each partition of the data in parallel, followed by a reduce-filter to aggregate the results.

2.3.5 Alternatives to String Attributes normalisation

The tokenization that is explained in Sec. 2.3.4, is useful to reduce the computing power required and ease the learning process for raw strings within our auto-encoders (compared to only UTF-8 encoding). However, in some cases (e.g., no GPU available), it might be necessary to reduce the required computing resources even further. To do so we can re-purpose the tokenization in two different ways. The first one consists in using an histogram of frequency of tokens. More specifically, we construct this histogram X using Eq. 2.4, with x_i the i^{th} value of the histogram and c_i the number of occurrence of the i^{th} token in the string to trans-

form. Using the logarithm instead of the raw count can limit the risk of attackers evading detection by padding their malicious payloads with normal data [59], as it requires orders of magnitude more frequent tokens to reduce the value of the unexpected tokens in the histogram.

$$X = (x_0, x_1, ...x_n) \mid x_i = log(1 + c_i) \tag{2.4}$$

The second alternative relies on a desired property of the token identification algorithm, that selects tokens based on their ability to compress the strings. Hence, the compression ratio (i.e., the number of tokens in the original string divided by the number of characters in the compressed form) should be higher for normal data than for anomalous data. In fact, normal data should be similar to the data the frequent tokens have been mined-on, and thus contain more of them. On the other hand, anomalous data likely contains unknown patterns, and thus should be less compressed. The computed ratio can then be used similarly to a numerical variable by the auto-encoder.

The first alternative is preferred as it is more robust to evasion and loses far less information since one still knows which tokens are present in the string and their proportion. However, the compression ratio approach further reduces the required computational resources, at the cost of being easy to evade (e.g., by padding the malicious input with multiple, large and frequent tokens).

2.4 Auto-encoder structure

In this section we explain the core of our approach, i.e., how we generate an auto-encoder structure based on the selected attributes to analyse. We detail the block-based architecture and the design choices we made to allow the auto-encoder to cluster events and provide a normalised anomaly score for all attributes in the events.

2.4.1 Overview

Given the diversity of inputs, each attribute needs to be processed differently by the auto-encoder. For the embedding part, the first layers are organized into independent blocks, one block for each input attribute. The encoded representations

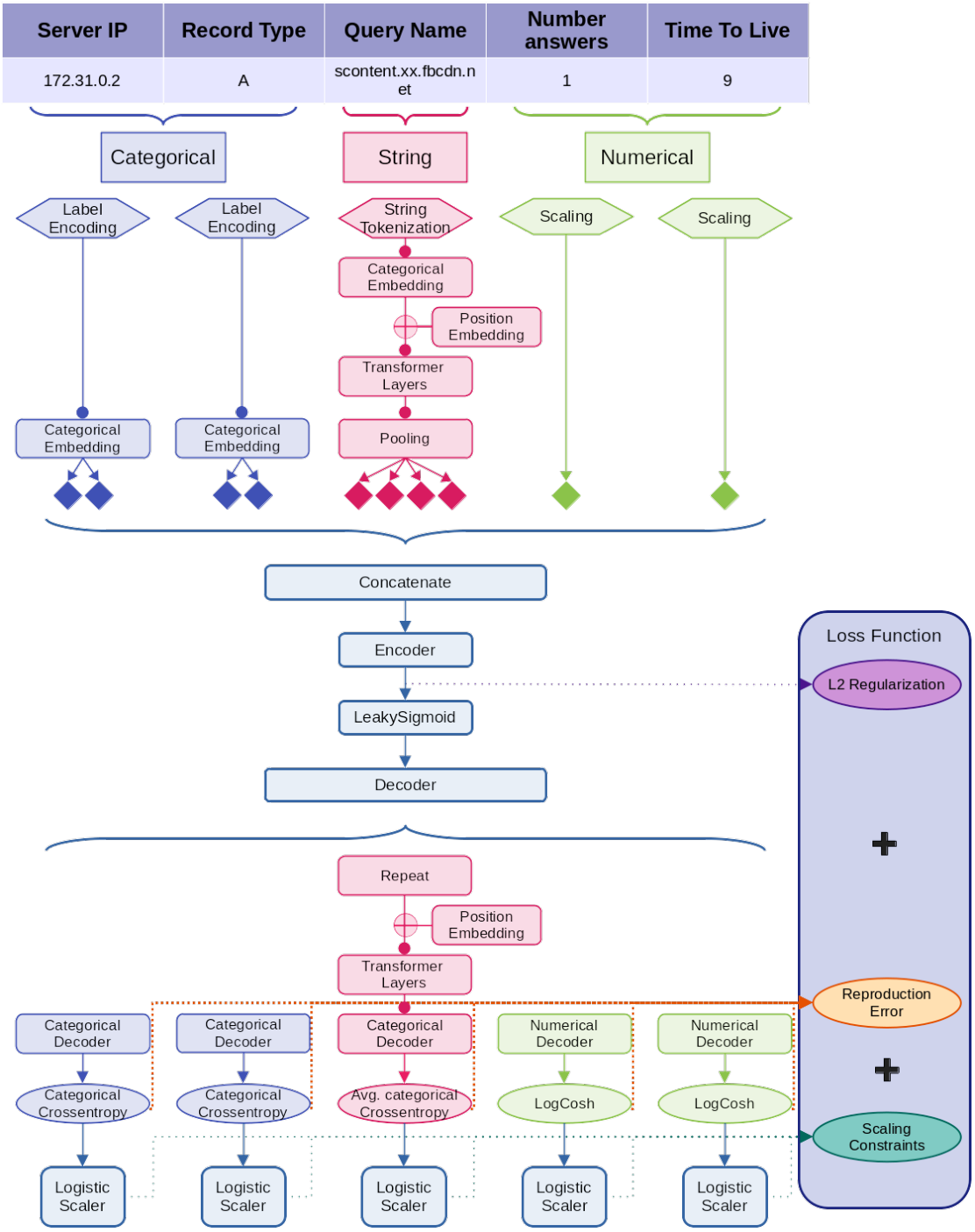


Figure 2.4 – Example network structure with a DNS event

of each attribute (in the form of floating point vectors), provided by these blocks, are then horizontally concatenated before being processed by the shared layers of the encoder. Symmetrically, for the decoding part, the encoder output first passes through shared layers of fully connected neurons, and the last shared layer output is used as an input to every reconstruction blocks. We implemented three types of blocks (Fig 2.4), one for each type of variables (categorical, numerical or strings), and the function used to compute the reconstruction error. This reconstruction error is also used as the objective function to minimize during training. As such, for all blocks, this function needs to be differentiable.

2.4.2 Blocks for numerical variables

Originally, neural networks, like most machine learning algorithms, are meant to process vectors of floating point numbers. Therefore, numerical variables are easy to process in our case. Specifically, the input and output blocks are simple neurons with the Leaky ReLU (Eq. 2.5) activation function.

$$g_\alpha(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{otherwise. } \alpha \leq 1 \text{ (ususally 0.1)} \end{cases} \tag{2.5}$$

Figure 2.5 – LeakyReLU activation function, with α leak rate.

To compute the reconstruction error, we use the log of the hyperbolic cosine, or logcosh for short (Eq. 2.6). For low error values (i.e., between 0 and 1) it behaves similarly to the traditionally used squared error, which means training is more and more precise when the reconstruction error gets closer to 0 (i.e., perfect reconstruction). For higher reconstruction error, logcosh is similar to the absolute value of the error. This is particularly useful in the context of anomaly detection, because anomalous events are likely to be present in the training data and generate larger error than normal events. Using the squared error would result in giving much more importance to these anomalous data points, thus reducing robustness of the method to noise in the training data.

$$E(x, \hat{x}) = log(\frac{e^{x-\hat{x}} + e^{\hat{x}-x}}{2}) \tag{2.6}$$

Figure 2.6 – LogCosh loss function. x is the original value, and $\hat{x}$ the estimated reconstructed value

2.4.3 Blocks for categorical variables

As explained in section 2.3.3, we use a method similar to *word2vec* [79]. Its implementation within a neural network consists in a layer with as much input as there are possible categories, which takes sparse vectors as an input, i.e., vectors filled with 0 for every components, except for the i^{th} which is 1 (with i the category identifier). This is often called a categorical embedding layer and it outputs a dense floating point vector of lower dimension called embedding dimension.

Symmetrically, the output block has as much output as there are categories and is trained to output a floating point vector with a higher value for the i^{th} component. As usual with neural networks, the softmax function (Eq. 2.7) is then used on this output vector to compute a vector whose components sum to 1. This is similar to a probability distribution, which allows us to use the categorical cross-entropy function (Eq. 2.8) as the reproduction error.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{n} e^{x_j}} \tag{2.7}$$

Figure 2.7 – Softmax value of the i^{th} component for vector X = $\{x_0, x_1, ..., x_n\}$

$$E(x, \hat{x}) = -\sum_{i=0}^{n} x_i \log(\hat{x}_i) \tag{2.8}$$

Figure 2.8 – The categorical cross-entropy loss function

2.4.4 Blocks for string variables

The blocks for strings are more complex. The chosen structure is inspired by state of the art NLP techniques. It combines a first layer of embedding (to transform a sequence of token identifiers into sequence of float vectors), a set of layers

that performs most of the processing meant to find inter-dependencies between the tokens to learn the semantic and syntax of the attribute, and a final layer of pooling to transform a sequence of vectors into a single vector.

Syntax and semantic modelling

Two core concepts have been employed in NLP to model the semantics and syntax behind a language to perform various tasks (e.g., translation, summary, named entity recognition, etc.). Historically, Recurrent Neural Networks (RNN) have been employed for these tasks, and today the Long Short-Term Memory (LSTM) neural networks [49] and Gated Recurrent Units (GRU) neural networks [21] are still popular. In recent years, the attention mechanism have been mixed with these techniques [8, 20, 76, 56]. It improved the models quality by highlighting the interdependencies between the steps of the sequence (i.e., tokens, words or characters). Lately, methods that solely rely on the multi-head attention mechanism have shown to outperform RNN and mixed approaches. Multi-head attention consists in performing multiple attention operations (called attention heads) that attend to different parts of the sequence, and combining their results. This provides the ability to cross information from more tokens in a string, and is the core component behind the Transformer neural network structure [114]. A Transformer layer (shown in Fig. 2.9), is made of a multi-head attention block followed by a feed forward network, consisting in two fully connected layers of formal neurons with the ReLU (Eq. 2.9) activation function. To mitigate the vanishing gradient problem [48], a residual connection is added to these two parts (Add & normalise on Fig. 2.9). It consists in adding the input of each part to its output and normalising the result before passing it as an input to the next part. In our case, we choose to use the Transformer structure to model string attributes.

$$g(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.9}$$

Token embedding

Both RNN and Transformers are designed to process sequences of floating point vectors. Therefore we need to convert each token identifier into a float vector. To do so, tokens can be considered as categorical variables, and thus we can use the

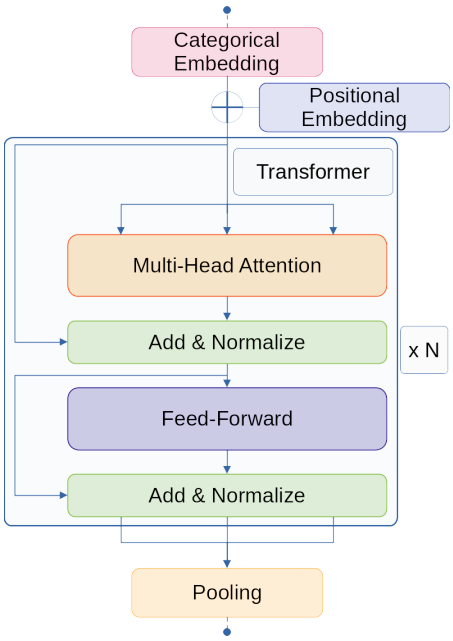


Figure 2.9 – Overview of a Transformer structure within the string encoding block.

method described in Sec. 2.4.3.

In the case of Transformer models, the attention mechanism do not take the order of the tokens into account. Therefore, to improve the model's performance, adding position information to the embedding (called positional embedding) is a common practice [114]. In the end, the input of the Transformer component is the sum of the categorical and positional embeddings of the tokens.

As a sequence of tokens can be considered as a sequence of categorical variable, the reproduction error is simply the average cross-entropy (used for categorical attributes, see Sec. 2.4.3) over the whole sequence.

Pooling

The Transformer part of the string processing block outputs sequences of vectors. However, to be able to concatenate the encoded representation of all the attributes, it is required that each encoding block outputs vectors. To do so, we considered two different approaches.

The first one, called average pooling, consists in averaging all the vectors in the sequence to end-up with a single vector. While this operation is straightforward to implement, it tends to dilute the information contained in each token, especially

for long sequences.

To avoid this problem, we drew inspiration from [38] for the second approach. Specifically, we used the attention weight computed during a multi-head attention operation to perform a weighted average instead. This way, the pooling function can accentuate the importance of certain tokens when reducing them to a single vector. This operation is called Attention Pooling.

For the decoding block, we need to go back from a single vector to a sequence of vectors. To do so, we first duplicate the vector (as much time as there are tokens in the string). Then, similarly to the input block, we add positional embedding to this sequence of vectors. This sequence can then be processed by the Transformer component of the decoding block.

Low resources alternatives

As mentioned in Sec. 2.3.4, Transformers and RNNs require large computational resources. Since the available resources are sometimes not sufficient to handle these types of models, we proposed to handle strings using the token log-count histogram or simply the compression ratio offered by the tokenization. For the latter, once transformed, the variable can be handled as a numerical attribute.

On the other hand, using the histogram requires a few modifications. In fact the encoding block becomes a feed forward network of two fully connected layers of neurons. This network has as much input as there are possible tokens and a lower amount of outputs (similarly to an embedding function). Symmetrically, the output block has as much outputs as there are possible tokens. The reconstruction error is computed by substracting the cosine similarity to 1 (Eq. 2.10). This function is commonly used in the data science domain as a measure of divergence between high-dimension vectors (in our case, the number of tokens can be large).

$$E(X, \hat{X}) = 1 - \frac{X \cdot \hat{X}}{\|X\|\|\hat{X}\|} \tag{2.10}$$

2.4.5 Latent clustering block

In the cybersecurity domain, regrouping similar events can ease the analysis process (i.e., analysing a few groups of events instead of all events one by one). In addition to its interesting properties for anomaly detection, the auto-encoder

can also provide a low dimension representation of its inputs thanks to its latent layer. This latent layer can be used for event clustering. Recent work [34] obtained good clustering performance by combining auto-encoders with Gaussian mixture model. In the case of security events, we found that the dominance of categorical variables leads to many clusters with a standard deviation close to 0, which is not an ideal case for Gaussian models due to the division by the standard deviation in the Gaussian equation. By outputting vectors of bits (either 0 or 1), the approach we propose better fits the discrete nature of the variables found in security events.

The first attempt to incorporate this property in the latent layer had it divided into 2 blocks of N components. We design the network such that the left-hand side expresses, for each of the N components, the likelihood of it being 0, while the right-hand side expresses the likelihood of being 1. The output vector converges to this structure due to the back-propagation property of the neural network. Hence, if the value of the left-hand side is higher than the value of the right-hand side, the corresponding component in the latent space will be close to 0. Otherwise, the component value in the latent space will be 1. We chose to use the Soft-ArgMax function (Eq. 2.11, where β controls the steepness) so that the output of the latent layer is either close to 0 or close to 1, while still being able to compute a gradient (required for neural networks).

$$f(x_0, x_1) = \frac{e^{\beta x_1}}{e^{\beta x_0} + e^{\beta x_1}} \tag{2.11}$$

However, while the gradient exists with this function, it is close to zero for much of the input range. In fact, as shown in Fig. 2.10, unless x_0 (the value of the left-hand side) and x_1 (the value of the right-hand side) are close in value, the gradient of the function is almost null. This is also known as the vanishing gradient problem, that causes the increments to the weights to be so close to 0, and thus, can prevent the encoding blocks to effectively learn the structure of normal data (and also perform useful clustering). Instead, we use a single block with a LeakySigmoid activation function, shown in Eq. 2.12, with $\alpha > 0$, to control the leak rate and $\lambda > 0$ for the steepness. Its inspiration comes from the sigmoid function, whose output ranges between 0 and 1, but is notoriously causing the vanishing gradient problem [48] and the LeakyReLU function (Eq. 2.5), that compensates the null gradient of the ReLU function (Eq. 2.9) by multiplying its input by a given leak rate (e.g., 0.1) when this input would cause a null gradient (i.e.,

$x < 0$ for ReLU). This way, the gradient is at least equal to the leak rate instead of 0 (Fig. 2.11). Additionally, we add L_2 regularization to the latent output (before applying LeakySigmoid), to encourage the encoder to output values closer to 0, i.e., in the range where the gradient of the Sigmoid is higher.

$$f(x) = \alpha x + \frac{x}{1 + e^{-\lambda x}} \tag{2.12}$$

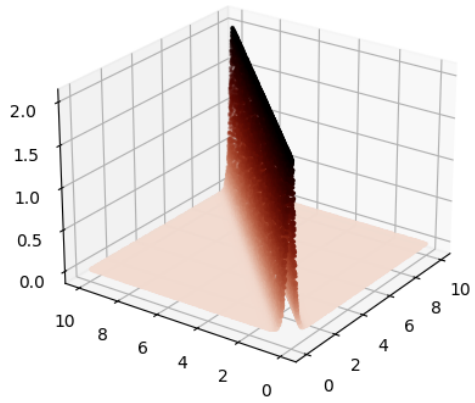


Figure 2.10 – Partial derivative value of softargmax(x_1, x_0) w.r.t. x_1

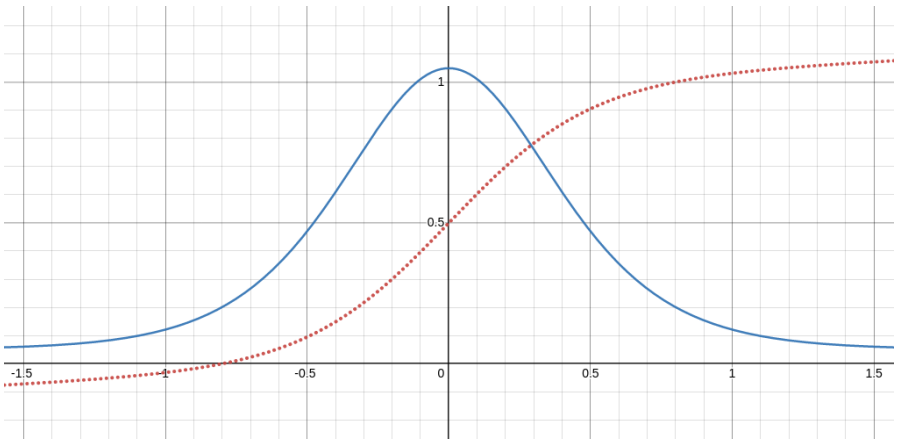


Figure 2.11 – LeakySigmoid (red dotted curve) and its gradient (blue curve), with the leak rate $\alpha = 0.05$ and $\lambda = 4$

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{2.13}$$

$$CDF(x) = \frac{1 + erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)}{2} \tag{2.14}$$

Figure 2.12 – Cumulative Distribution Function at value x for as Gaussian distribution parametrized by (μ, σ). The error function is denoted erf

In the end, if we round the output of the latent layer, it becomes a vector of N bits. These N bits define a cluster identifier, that is later used to regroup the events with equal identifiers. Alternatively, it is possible to use the floating point value as an input to a typical clustering algorithm (e.g., DBSCAN, K-Means, etc.).

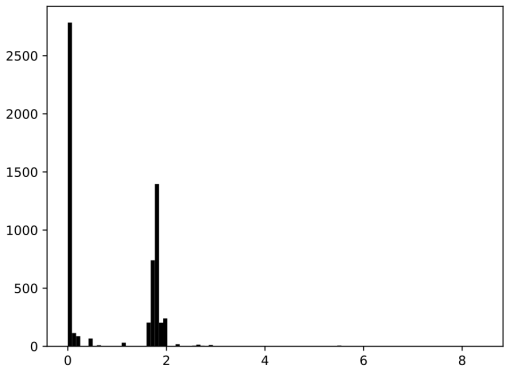
2.4.6 Normalising per attribute anomaly score and computing event score

Due to the diversity of attributes and types of attributes, the reproduction error for one attribute is not directly comparable to the reproduction error of another attribute. As an example, if error E_0 for attribute a_0 ranges from 0 to 1 and error E_1 for attribute a_1 ranges from 1 to 1.5, then the comparison $E_0 < E_1$ is not an indication of anything. To provide hindsight of what attribute might have caused the anomaly, we need to be able to compare anomaly score between attributes.

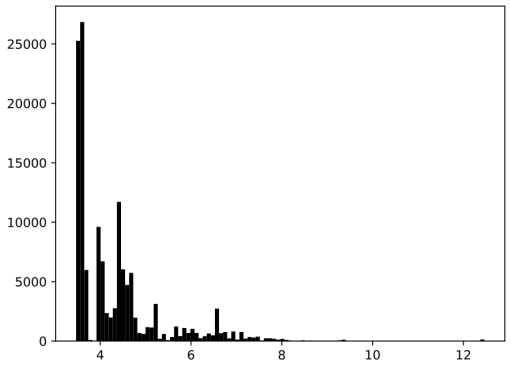
During our experiments on multiple datasets and types of security events, we found that for most attributes, distribution of the reproduction error on normal data can be approximated by a Gaussian mixture model (Fig. 2.13). The parameters of such a model are the mean μ, the standard deviation σ and the probability ϕ for each of the n Gaussians. We originally computed the anomaly score of the attribute using the Cumulative Distribution Function (CDF) of the Gaussian (μ_0, σ_0) with the highest μ (i.e., the distribution of the least normal events). Indeed, the CDF outputs value between 0 and 1, with 0 implying the attribute has a normal value, and 1 implying that it is likely anomalous.

However, fitting a GMM directly within neural network is not straightforward. Instead, the Expectation-Maximization algorithm [29] is the favoured approach to find the parameters. Doing this requires an additional step after the training of the

neural network weights. Moreover, in cases where the variance of the loss is close to 0, this method provides unsatisfactory results (i.e., high score for all events or low score even for anomalies).



(a) Cross-entropy loss for executable name in a process event



(b) Cross-entropy loss for user-agent in a web server *access.log* event

Figure 2.13 – Distribution of the reproduction error for attributes of process execution and web server *access.log* events.

Instead of the Gaussian mixture, we therefore propose to use a logistic distribution (see Fig. 2.14). The objective is to have a score that is close to 0 for most of the normal events, while still being close to 1 when the attribute is anomalous. To do so, we train the parameters of the distribution accordingly, directly within the network, by adding terms to the loss function (Fig. 2.15). The CDF equation for the logistic regression is given in Eq. 2.15, with λ that controls the steepness of the curve and μ the symmetry point $f(\mu) = 0.5$. To that effect, we find μ (the point at which the logistic CDF is 0.5) such as it approximates the highest value on the (supposedly normal) data. To do so, we constraint the parameters of the logistic distribution (Eq. 2.17). Specifically, a constraint is added to the μ parameter of the attribute scoring function (Eq. 2.16). For this constraint, the lower α is, the closer to the maximal input value μ will be, and $\overline{X}$ is the average value of the reproduction error for a batch of data. In addition to the constraint on μ, another one is added to avoid λ being negative (which would cause the score to be higher for normal values than anomalous ones).

The score of an event is the weighted mean of the score of all its attributes. The weight w_i of each attribute x_i is computed by a small neural network that takes as an input the encoded representation of the event (latent representation).

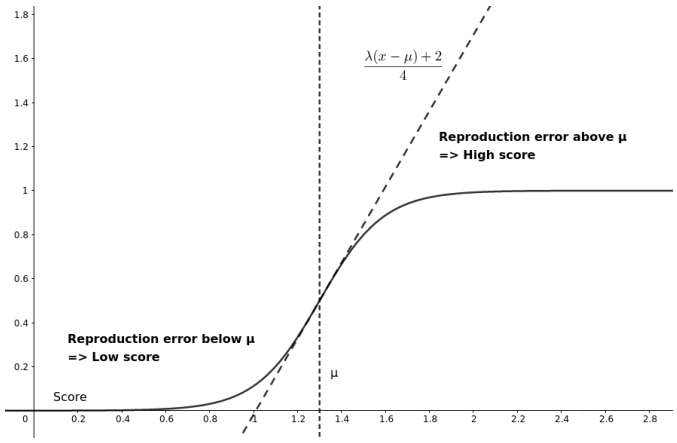


Figure 2.14 – Example of the per attribute scoring function

Minimizing the standard deviation of the weight vector is added as an objective to discourage the weighting neural network to amplify or attenuate excessively some attributes.

$$f_{\lambda,\mu}(x) = \frac{1}{1 + e^{-\lambda(x-\mu)}} \tag{2.15}$$

$$E_{\mu}(X) = \alpha * g\left(\mu - \overline{X}\right) + g\left(\overline{X} - \mu\right) \tag{2.16}$$

$$E(X) = E_{\mu}(X) + \overline{f_{\lambda,\mu}(X)} + g(-\lambda) \tag{2.17}$$

Figure 2.15 – Constraints for the scoring function

2.5 Configuring, training and using the model

This sections focuses on the usage of our approach, from an analyst point of view. Specifically, it describes how analysts would configure models for their needs, how they would train them, and more importantly, in which case and how to use these models.

2.5.1 Model configuration approach

Our goal is that our model is adapted by the analysts to suit their own need. Therefore, we aim for our configuration process to be as similar as possible to a detection engineering process (see 1.2.2), which analysts are familiar with.

Essentially, detection rules are defined following four major steps. First, the sources of security events to be analysed need to be chosen by the analyst. The second step is to select the list of attributes from these sources. At this point analysts can start to implement the detection logic, i.e., the conditions that needs to be met in order to consider an event as illegitimate. Finally, the analysts need to fine tune the detection logic because it is likely to trigger false positives at first. This is the longest step of the process. It is done by allow-listing some events, users, machines, etc. or making the initial pattern matching more specific. For example, if analysts want to detect malicious command lines executed from a document with macros, they can analyse the events recording the execution of processes (e.g., Event ID 1 with Sysmon, Event ID 4688 from Windows audit logs, etc.), and from this source, they need to select at least the process and its parent names and command line arguments, and optionally the host and user that executed these processes (e.g., to filter out the false positive caused by departments which rely on macros). They can then detect the execution of command line interpreters with the text processing software as parent process. Finally, they can fine tune this rule to account for legitimate use cases that may trigger alerts.

The process described in this chapter also implies the definition of the source of data and its attributes. Indeed, when they design detection capabilities for a traditional detection system, analysts implicitly define which attributes to look at. Whereas, with our proposed system, the definition of these attributes is explicit. The proposed process also requires an additional step of defining the type of each attributes (i.e., categorical, string or numerical). This task can be made faster when using well defined event data models (e.g., Elastic Common Schema [2], Splunk CIM [3], etc.). Indeed, while these data models will not take into account specific enrichments performed by the analysts, they provide the type of most of the common attributes found within security events. Furthermore, the extra time

2. ECS documentation: `https://www.elastic.co/guide/en/ecs/current/index.html`
3. Splunk CIM documentation: `https://docs.splunk.com/Documentation/CIM/5.0.1/User/Overview`

```
{
    "process_exec": {
        "process.exe": "categorical",
        "process.cmd_line": "string",
        "parent.exe": "categorical",
        "parent.cmd_line": "string",
        "username": "categorical"
    }
}
```

Figure 2.16 – Example configuration provided by the analyst for a model that analyses executed processes.

requested to chose the attributes and define their types are mostly compensated during the fine tuning part that is mostly done by the model itself during the training phase. Finally, one model configuration can be applied to multiple detection use cases, instead of a single one when designing rules. As an example, for the use case described in the previous paragraph, and more generally for anomalous process detection, we could use the configuration described in Fig 2.16.

2.5.2 Training the system on baseline data

With our method, there are two sets of parameters that need to be adapted on data before the model is ready to be used. Specifically, before training the networks weights, the transformation functions need to be adapted on the data, by finding the known values for categorical attributes, mining the tokens for strings, and finding the 90^{th} percentile for the numerical variables. This is akin to the baselining of the system that is performed when a SOC is deployed. Our method is designed to be used as a way of automating this process.

The challenge for anomaly detection is that the monitoring data of a production system may be tainted by previous intrusion attempt. Indeed, our method is more precise when trained on normal activity only, as an adversary behaviour occurring regularly might end up being considered as normal. This is a limitation of anomaly detection-based approach. We try to limit its impact by using more robust methods (e.g., using reconstruction error function that are less sensitive to outlier like *logcosh*). Also, while having completely clean data for training is unlikely, malicious behaviours traces in the training data would only prevent the

detection of these specific behaviours, and not of novel ones (e.g., new attacker, stealthy adversary starting the last phases of their attack, etc.).

2.5.3 Using the model on live monitoring data

Once trained, a model constructed with the described approach can expose two capabilities: clustering events and computing anomaly scores. Depending on the use case, these capabilities can be exploited differently. Specifically, we envisioned these models to be used either as a detection mean, generating alerts based on detected anomalies, or as a support for Exploratory Data Analysis (EDA), by clustering and prioritising events.

When designing detection rules, a significant amount of the detection engineer's time is spent on fine tuning the filters to avoid false positives caused by the specificities of a system. For example, some departments may require the use of macros that execute commands, which can also be misinterpreted as malware execution. In this case, the specific users and the macros they rely on for their work should be allow-listed. This fine-tuning can take a lot of time and is prone to error (i.e., white-listing adversary behaviours). Our model can be trained on the output provided by a wider-spectrum rule (compared to a fine-tuned one) to automatically learn the usual false positives caused by the rule for a specific system. Doing so requires setting up a threshold for the anomaly score of the events. Above this threshold, an event can be considered anomalous, and thus more likely to be caused by an adversary action. An example of defining such a threshold is provided in chapter 4.

Our approach can also be applied to automate parts of the hunting process. In this case, the anomaly score can be used to prioritise events (i.e., the higher the score, the higher the priority), and similar events can be clustered to reduce the amount of time the analysts spend analysing them. The model can also be used as a way to enrich the events indexed in the SIEM. Specifically, the anomaly score and clustering identifier can be used to create hunting dashboards (e.g., plotting the cumulated anomaly score through time, identifying assets with the highest number of anomalies, etc.), and filter data using the indexer's search capabilities (e.g., returning only the events with a higher score than x for a given attribute, filtering out a set of cluster identifiers, etc.).

2.6 Summary

In this chapter, we proposed a method that relies on neural networks auto-encoders to compute anomaly scores for heterogeneous events. It can be used to prioritise investigations during incident response or threat hunting operations. We proposed an original approach to exploit the latent space of the auto-encoders to provide clustering capabilities. So as to reduce the volume of information that is presented to the analysts by regrouping similar events together.

We drew inspiration from state-of-the-art deep learning techniques to handle the most common attribute types found inside security events (numerical, categorical and string attributes). These techniques simplify the design of the feature extraction and transformation processes that are required by any machine learning-based approach. This allows to quickly create specific models for multiple sources of security events. While other machine learning methods for anomaly detection have already been proposed, they mostly need specific feature engineering for each new type of events, which requires knowledge in data science that is rarely available among Security Operation Center (SOC) analysts.

We envision two major use cases for these tools. First, we aim at reducing the required time for detection engineer to built new detection logic by reducing the time it takes to fine tune the detection rules to white-list legitimate and frequent behaviours that trigger alerts. Instead, the model is in charge of automatically performing this fine-tuning step. The second use case is focused on threat hunting and incident investigation. In these contexts, analysts explore large amount of security data to uncover potentially unknown attacker behaviours. The clustering capability as well as the ability to prioritise events based on the anomaly score can reduce the investigation time. In turn, analysts can perform more thorough analysis, which can help them uncover traces of advanced attacker behaviours.

In the next chapter, we focus on the ability to continuously update models without introducing attack traces. It requires to include the human analyst at the core of the ML-based system. As such, this system needs to output contextualised detection, which provides as much elements as possible for the analysts to understand and qualify anomalous behaviours.

Chapter 3

Adaptable and maintainable analytics for security operators

This chapter details how we combine detection rules, scripts, data fusion and machine learning models to build security analytics. It presents a scalable algorithm to perform automatic event fusion from pivot variables identified by analysts. We build upon the work described in chapter 2 to propose a dynamic auto-encoder structure that can process fused events. An active learning approach is proposed to continuously train the models in analytics while providing robustness against frog-boiling attacks (i.e., poisoning continuously learning anomaly detection models). To avoid increasing the burden on analysts, special care is taken to limit the volume of information presented to them, and provide as much context as possible to improve the explainability of the provided results.

3.1 Introduction

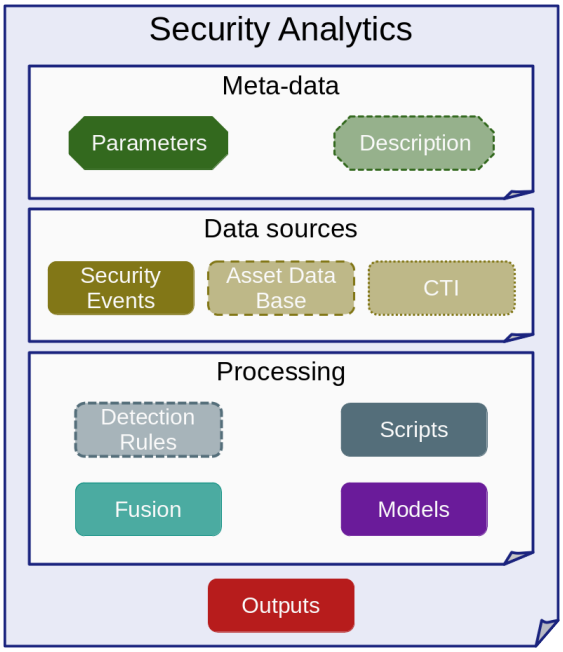
This section introduces our vision of security analytics, which combine detection rules, scripts, data fusion methods and models to process heterogeneous security data from multiple sources. We focus on simplifying the design process for analysts by abstracting the data science parts. We also aim at simplifying the maintenance process, which is required to adapt to novel behaviours, and rely on active continuous learning of models with the human analysts in the loop.

3.1.1 Designing security analytics

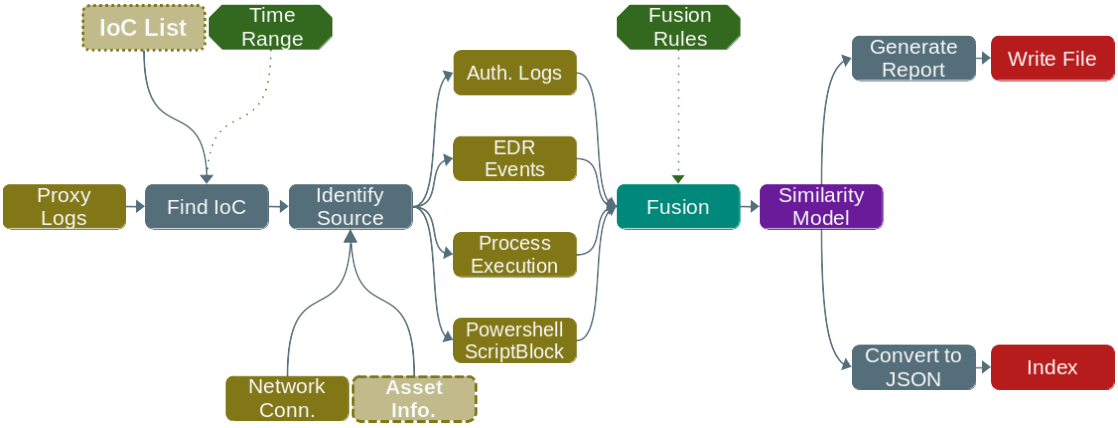
SOC analysts perform intrusion detection of monitored systems thanks to automated real-time detection systems that recognize known attack patterns often relying on sets of correlation rules. Moreover, the threat hunting analysts uncover unknown attack methodologies by exploring events in the search of potentially suspicious activities. The number of false alarms raised by the real-time detection systems should be kept to a strict minimum in order for incident responders to react as quickly as possible to known attacks (see Sec. 1.1.3). Threat hunters, on the other hand, can dedicate more time to perform in-depth analysis of these incidents to find novel attack patterns. They can also investigate events in the search of emerging threats behaviours. Thus, due to the huge amount of data that needs to be analysed and pieced together to investigate anomalous (and potentially malicious) patterns both incident responders and the threat hunting teams can greatly benefit from automation tools.

In recent years, multiple anomaly detection methods have been applied to security monitoring. However, multiple limitations still slows the adoption of these methods that are mainly based on machine learning. While novel deep learning advances can simplify the design process by being more flexible regarding the input data [36], the results they provide may still be hard to interpret. This lack of explainability can be lowered by presenting contextualized events to the analysts [80]. This prevents analysts from understanding the mechanics and hinders their ability to qualify alerts coming from these methods. We want to implicate analysts and detection engineers directly in the process of designing analytics, in order for them to exploit data science capabilities more efficiently.

Recently, multiple approaches have been applied to endpoint monitoring with interesting results such as information flow tracking [44, 16] and event causality [122]. However, in the context of a SOC, it is unlikely to have access to the level of detail necessary to find causal links between events, and we need to rely on approximations instead. Instead, we focus on event fusion method that enables analysts to automatically reconstruct the context around security events and can be configured by analysts according to the data they have access to.



(a) Overview of the analytics structure.



(b) Example use case of an analytics: threat hunting from IoC.

Figure 3.1 – Analytics structure and use cases.

To accommodate the evolving needs (and capacities) of security operators, we aim at providing analysts with building blocks to create, improve and exploit security analytics (i.e., automated data processing procedures to support security analysts) for their operations. These analytics should be able to process, at scale, multiple sources of heterogeneous data, and take into account how the results they provide could be visualised [92]. Fig. 3.1a provides an high level view of the structure of an analytic. Specifically, an analytic is called using a set of parame-

ters (e.g., a time frame to analyse, a list of machines to investigate, etc.). Using these parameters, the analytics retrieve data from multiple data sources from the various tools that store security data, i.e., security events from the SIEM, asset information from the asset base, and Cyber Threat Intelligence data from the CTI or threat sharing platform (e.g., MISP [1], OpenCTI [2], etc.). Expert knowledge can be provided in the form of detection rules (e.g., using the SIGMA format [3]), or as more flexible scripts. We aim at being able to aggregate data from all the sources using fusion methods and mix the use of expert system with anomaly detection and clustering models.

In Fig. 3.1b, we provide as an example a high level view of an analytics meant to accelerate hunting based on URL that can be Indicators of Compromise (IoC). When doing so manually, threat hunters often find false positives either due to bad IoC (e.g., error in the CTI feed, insufficient precision of the monitoring data, etc.), or because the IoC describes legitimate infrastructure that has been compromised but only provides its original service to our system. Therefore, they need to investigate each match by analysing the logs of the machine from which the connection is originated. To accelerate the process, the analytics can retrieve automatically the logs for each match, fuse them to reconstruct the context around each of these match and regroup similar matches to facilitate the analysis by the human expert.

3.1.2 Maintaining security analytics

The security domain has a strong specificity that requires to handle continuous learning (i.e., updating models with novel behaviours) with extra care. Namely, attackers will actively seek to evade the defensive measures [116]. For anomaly detection models, continuous learning exposes another attack vector, commonly called the frog-boiling attack (see details in Sec. 1.3.3). This attack consists in progressively presenting to the model inputs that are increasingly similar to the whole attack. As the model is retrained regularly on novel data, at some point, the attacker's activity will be considered normal.

For initial access, this type of attack is unlikely to be effective, as attackers have limited knowledge of the target system, which prevents them from generating

1. `https://www.misp-project.org/`
2. `https://www.opencti.io`
3. `https://github.com/SigmaHQ/sigma`

actions that seem normal enough not to trigger an alarm, and they need to gain persistence rather quickly (which may be less difficult to detect) considering that the complete implementation of the attack may take time. However, for advanced attackers that can inject a backdoor into the target system (e.g., via a supply chain attack like the SunBurst malware [119]), this attack is plausible, as it can limit the visibility of the trail left by attackers on the target system.

To mitigate this attack vector, we need to limit the number of untrusted data points that are used to incrementally train the model. Considering that the approach is meant to detect novel threats, finding an automated method to verify if the data points are symptomatic of unwanted behaviours is not an option (i.e., if such a method existed, it would probably be easier to use it directly instead of using ML models). Performing such a verification is however very similar to what analysts do when they qualify alerts. In fact, among all the scored meta-event sets, some can be selected (e.g., the top N with the highest score, or the ones with a score higher than a predefined threshold), and then presented to analysts for them to annotate (e.g., unwanted or false positive). Only verified false positives would be inserted into the model, which adds a layer of complexity for the attackers.

3.1.3 Overview of the approach

In this chapter, we build upon the approach detailed in Chapter 2 to propose a method to detect traces of anomalous and potentially malicious activity by analysing security events coming from multiple sources (e.g., process auditing, network probes, web proxies, etc.). Our method does not require specific low-level information from logs, and can therefore be configured for various monitoring strategies (i.e., ability to configure it for various sources of events, and various levels of visibility).

The first step of this method is to regroup events into sets that describe the same action (e.g., a network connection seen as an opened socket by the endpoint monitoring tool, and as a network flow by network probes). In case the action triggers an alert, this provides context to the analysts, which allow them, in theory, to qualify alerts quicker. To analyse these sets, we use an auto-encoder model, derived from the approach described in chapter 2, a specific type of deep neural network that is particularly suited for anomaly detection. We design the model so that it can analyse any attribute contained in security events, and link information

from all the events within each set to detect anomalies based on the context (e.g., reading `/etc/password` at start-up is normal for a web server, but reading the same file when answering a request might not be). The model can learn incrementally to adapt to the evolution of the normal behaviour of a system. To account for changes in the monitoring strategy, the structure of the model can be updated without requiring a complete retraining (which is computationally intensive).

Due to the large volumes of events that need to be analysed, whenever possible, we choose algorithms that can benefit from parallel and distributed computing. We give a particular attention to integrate easily into SOC analysts habits and capabilities. Specifically, the configuration step does not require advanced knowledge in data science, and instead focuses on available expertise from SOC analysts.

3.2 The need for event aggregation

In this section, we detail why we focus on event aggregation, and especially event fusion. We aim at proposing an efficient way for analysts to investigate and understand results provided by analytics. Based on our experience with graph-based contextualisation methods, we conclude that these method require to reduce the number of nodes to be effective, as visualising more than a few tens of nodes prevents analysts from focusing on important events, and graph processing solutions in general are costly to scale with the number of nodes. We propose to do so via event fusion, and on pivoting automatically around data sources.

3.2.1 Introduction to alert correlation

Alert correlation methods are already part of SOC tools. Valeur [112] described extensively the correlation process (see Sec. 1.1.3), which can be summarized in six major steps, the **normalisation**, the **enrichment**, the **fusion**, which consists in fusing similar alerts and correlating multiple steps of the attack to detect known patterns and the **verification** which focuses on filtering out irrelevant alerts (e.g., known false positives) and prioritising the others. The method described in Chapter 2 shares similarities with this step (i.e., it can be used to filter false positives and prioritise events). The step of **aggregation** consists in regrouping various steps of the attack (detected during monitoring) in order to finally perform a **global**

analysis of this attack on the monitored system.

In our context, we analyse both events and alerts (i.e., every behaviour, not only suspicious ones). This allows the detection of novel attacks, but also implies that the volume of information to process is greatly more consequent than what is processed by an alert correlation system. In this section we do not focus on the normalisation and enrichment of events as these are already part of SOC and CERT processes, and extensive public resources are available for these (e.g., the Elastic Common Schema [37] and MITRE ATT&CK® data sources [83]). Instead, we investigate methods that can provide context to alerts in order for analysts to recognize and understand quickly the behaviours that lead to the alerts. Consequently, we consider improvements to the fusion, verification and aggregation steps that support the global analysis.

Current trend in the literature consists in exploiting network graphs to provide this context. Based on operational feedback (Sec. 3.2.2) and experiments (Sec. 3.2.3), we identify limitations that should be mitigated before these approaches can be employed in operational use cases.

3.2.2 Challenges of graphs for security events investigation

In section 1.2.3, we reviewed methods that employs graphs to visualise and investigate security events and alerts. Specifically, using information flow tracking and provenance graph can help reconstruct the various steps that constitutes a complete attack [51, 16]. Formal methods also exist to build graphs of security events from audit logs based on causal dependencies between them [122]. However, for our use case, they have four major limitations:

1. Such a graph quickly becomes too big (i.e., more than a few tens of nodes) to efficiently visualise even for short time ranges (i.e. above one hour). In fact, in typical IT systems, some servers are contacted regularly by almost all the machines (e.g., an Active Directory), which is translated into causal dependencies between all the machines, as a malicious modification to the listening processes of these could have an impact on all the others. This quickly creates an exploding number of dependencies between events, which implies a large dependency graph.
2. They are only used to investigate alerts, as the detection process still relies

on a complete and complex set of detection rules.

3. They require a consequent modification to the monitoring system, as they rely on low level events that generate massive amounts of data. This is incompatible with operational IT systems, as SOCs and CERTs usually cannot modify the monitoring tools deployed on the system, and even if they could, it would be extremely costly to collect and store these events as long as the legislation and contracts requires it (i.e., from 6 months to multiple years, depending on the sensitivity of the system).

4. Adapting such a system to novel sources of data is complex, as the methods employed to build the graphs are strongly correlated with the data sources employed (i.e., they require specific attributes in specific types of events).

HOLMES [80] addresses the first limitation using a higher level graph to represent long scenarios. In fact, instead of representing one event by one node in the graph, a node corresponds to a composition of multiple events that can be attributed to an adversary technique. It also relaxes slightly the constraint on the detection rule set by using anomaly detection to filter frequent false positives. However, this higher level graph requires the definition of specific rules to match adversary techniques and still relies on low level events and specific informations that are not available in our case.

Instead of causal dependencies between events, it is possible to construct graph from identified pivot variables within the events [91, 70]. This is more aligned with our objectives, as these pivot variables are often identified intuitively by incident responders and threat hunters to investigate security events. Furthermore, these graphs can be used to detect suspicious patterns by combining community detection with ML-based methods for edge weighting (supervised regression in [91], anomaly detection in [70]). However, such a graph scales poorly with high number of events (i.e. starting from a few millions).

In summary, to take advantage of a graph representation, we need to reduce the number of nodes to present to the analysts. We also need this method to be adaptable by the analysts to the available data.

3.2.3 Experimenting with graphs and anomaly detection

To simplify the investigation by analysts of the results provided by the method described in chapter 2, we investigated a way to combine a graph representations with events anomaly scoring and clustering. The core assumptions are that an isolated anomaly is rarely the consequence of a successful attack, as attackers will perform multiple actions before reaching their goals, and that advanced attackers can perform seemingly normal actions, that generates events called *weak signals*, between the steps of their attacks (e.g., creating files on a network share to exchange informations with another compromised machine). The chosen approach had the following properties:

— The constructed graph is directed and acyclic (as a provenance or causal dependency graph). We aim at approximating, to varying degrees of fidelity (based on the available data), the causal dependencies between events, and according to Lamport's happened-before relation [65], an event cannot precede its cause (hence the acyclic nature of the graph).
— Across (and within) data sources, pivot variables (provided by the analysts) are identified to build the edges between events. For example a process creation event will be linked to the last event recorded that implicates the parent process (based on the process identifier).
— An edge is removed if the combined anomaly score of the previous nodes in the graph is too low. The combination function is optimized to limit the number of edges on baseline data (i.e., supposed without attack).

As shown in Fig. 3.2, this approach can quickly create graphs that are too big to understand. In fact, even if the graphs presents less than a percent of the total events, there are tens of thousands of said events per machines per day. In a typical IT system that has more than a thousand machines, the graph will not provide valuable insights. This is especially true if we try to visualise more than a few minutes of activity, as the number of dependencies between events will start to explode.

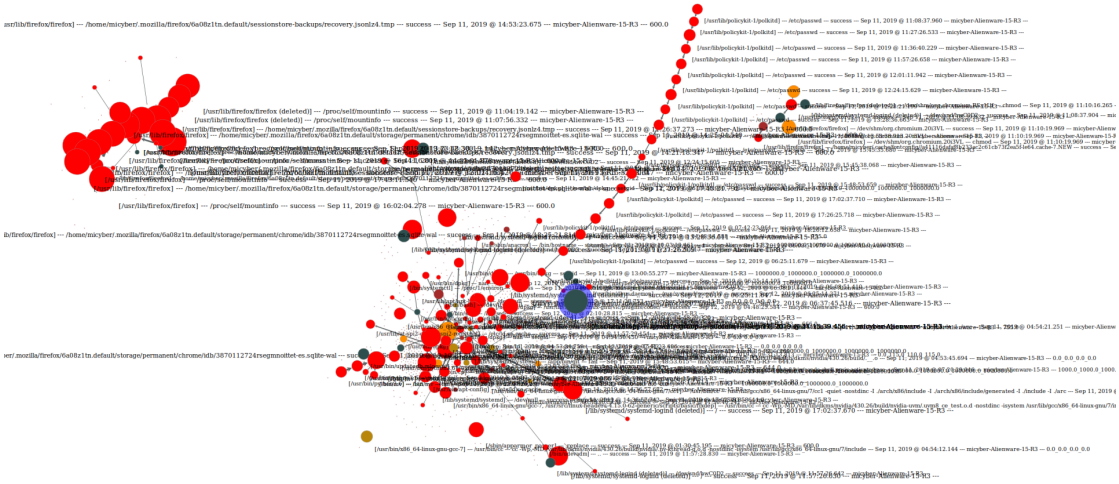


Figure 3.2 – False positive sub-graph constructed during our experiment (in this case, caused by a software update).

However, by only showing the most anomalous events, and aggregating directly linked similar events (i.e., with the same cluster id), we found that for short periods of time, it can help understand attackers activities. Fig. 3.3 shows the graph with nodes corresponding to aggregated events distributed horizontally by time windows. Fig. 3.4 shows a timeline view, which highlights the most anomalous events per time window.

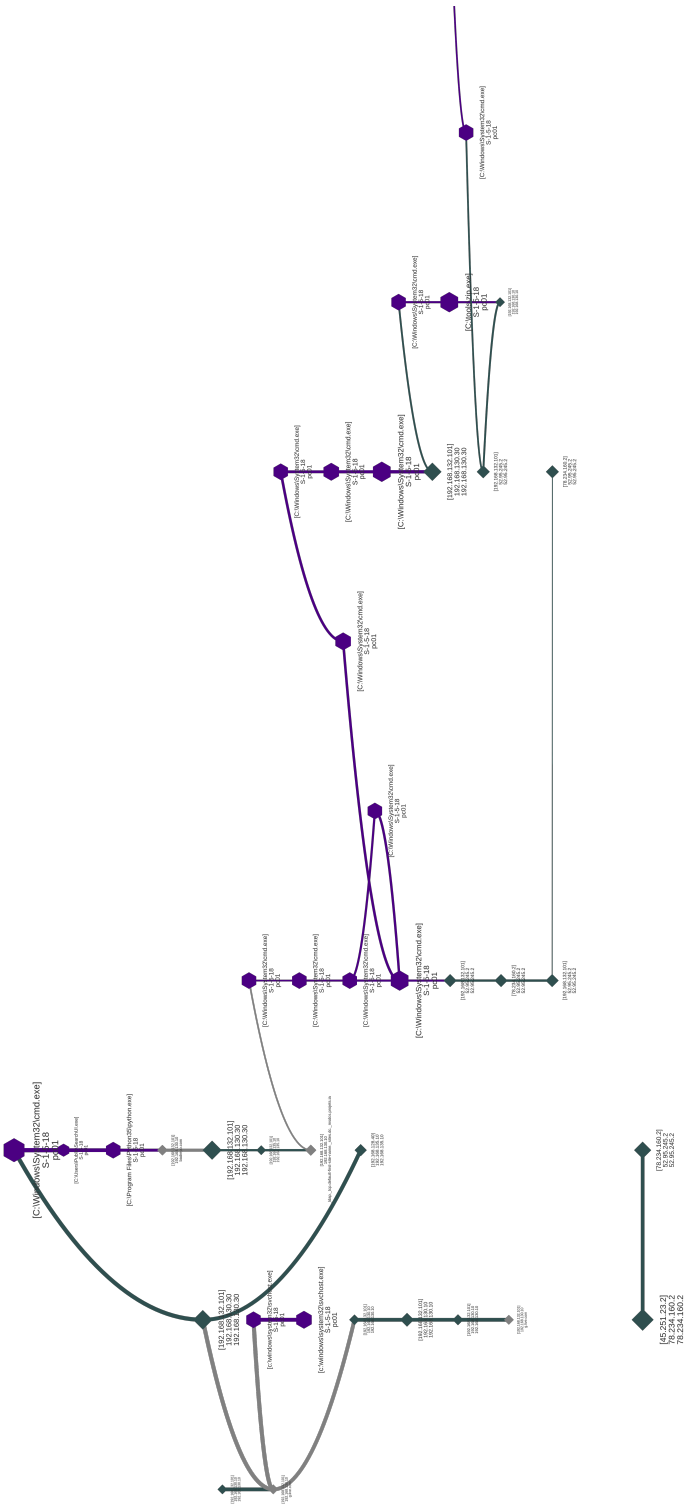


Figure 3.3 – Filtered representation a few steps of an attack, happening in a short period of time (a few minutes), as a DAG.

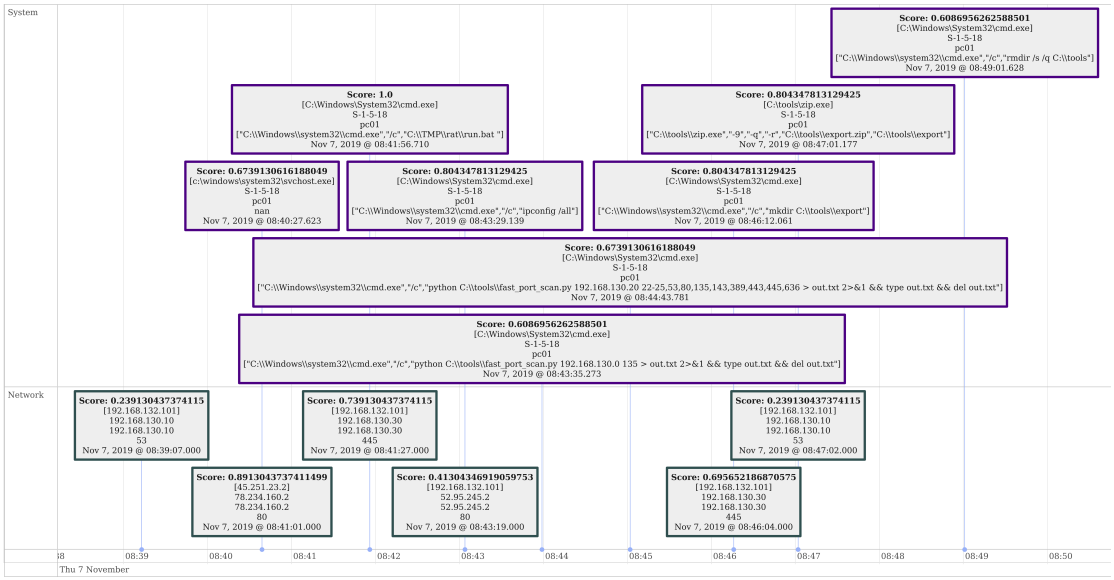


Figure 3.4 – Filtered representation a few steps of an attack, happening in a short period of time (a few minutes), as a timeline.

Finally, with the approach for anomaly scoring described in chapter 2, we had to manage and coordinate one model for each data source. When filtering and refining the visualisation using the anomaly score (i.e., only showing the most anomalous ones), this required the use of mechanisms to normalise the score from one source to the other.

3.2.4 Focus on event fusion and pivoting around security data

Our experiments with graph-based investigation of anomalous events showed that:

— For longer periods (i.e., an hour or more), too much nodes are present in the graph to be efficiently visualised.

— Within enterprise IT systems, some servers are regularly contacted by most of the machines (e.g., AD, file servers, Intranet portal, etc.), which generates spurious correlations between events. This is due to the insufficient precision of the approximation of causality that is attainable with COTS.

— One action performed by a human (legitimate or not) and its consequences can be recorded as tens of events in a short period of time, which further

reduces the efficiency of the graph as an event visualisation tool.
— Managing and synchronising multiple models to solve the same problem is not trivial.

As highlighted by Milajerdi et al. [80], to highlight multiple steps of a long-spanning attack, adding an abstraction layer between events and nodes (i.e., one node should be composed of more than one event) provides significant readability improvements. In their case, they associate certain combination of events, linked by causal dependencies, with adversary techniques, and display a graph whose nodes represent these techniques. Furthermore, graph analysis methods (e.g., graph databases, community detection, etc.) are expensive to scale with higher number of nodes and edges.

In our case, we cannot rely on having access to the level of information required to find causal dependencies. Moreover, we want analysts to be able to use our methods to find novel attack patterns, including novel TTPs. Therefore, to abstract nodes in the graph, we cannot rely solely on existing knowledge. Instead, we focus on regrouping events that are the consequence of the same actions performed by human. We also aim at designing a way for analysts to automate and scale the frequent task of pivoting around data.

3.3 Aggregating events

This section describes heuristics used to group events in sets of meta-events. First, we introduce the definition of a meta-event in the context of security monitoring. Second, we present the method we use which consists in regrouping events by time windows and then aggregate related events within these time windows using logic. Finally, we describe the algorithm that is used to aggregate events within a time window. Specifically, this algorithm extracts pivot variables from the events by combining relevant attributes (configured in the form of rules), and then use a reduction operator to group events into sets of meta-events.

3.3.1 Definition of a meta-event

Our objective is to propose an event aggregation method that can handle large volumes of events by taking advantage of parallel computing. Solutions for auto-

matically identifying which events should be aggregated has already been studied in previous work [101, 117]. In our case, we adapt a model that is already common for alert fusion, by aggregating events that are close in time and using simple logic rules to separate unrelated events within these time windows.

Defining rules is similar to identifying pivot variables when hunting for adversary behaviours, which is a core concept known by analysts to explore security data and follow the trails left by attackers.

Among the events collected when monitoring an information system, finding duplicated events is highly probable. This can be attributed mostly to four facts.

1. The same action being performed several times in a short period of time (e.g., a process spawning a pool of child processes). Due to the asynchronous nature of modern computing, it is unlikely to have the exact time-stamp of each events.
2. The sensor that registers the events or the chosen attributes of these events might not provide enough information to distinguish two similar but different actions (e.g., multiple threads connecting to the same service in parallel will generate socket events that can be hard to distinguish if the source port of the connection is not considered).
3. Redundant sensors or multiple implementations of these sensors (e.g., two different IDSes) for high availability.
4. For performance reason, the log shipping method of a lot of modern monitoring solutions often follow the "at least once" delivery policy by default rather than the "exactly once" policy.

For this reason, we chose to consider a meta-event as a group of events pertaining to the same data source during a predefined period of time and with equal attributes [4]. The proposed system aims to leverage knowledge of SOC analysts. The definition of a meta-event therefore depends on three choices, namely, the data sources, the size of the time window inside which events can be grouped together, and the list of attributes that needs to be equal for events to be grouped. As long as these choices remains in the hands of analysts, the performance of our system depends on assignments made. For the sake of this work, we define time

4. Timestamp excepted

window, data sources and event attributes based on expert knowledge provided by SOC analysts.

3.3.2 Linking meta-events

When monitoring an information system, the various sensors will record events differently (e.g., an endpoint audit mechanism may attribute network socket opening to processes, while network probes will focus on analysing the network protocol). To extract as much information as possible from available logs, we regroup all the events related to the same action and jointly analyse them. This is known as the alert fusion step of the correlation process [112], and we extend it to take both audit events and IDS alerts as inputs (instead of only alerts).

To leverage the available knowledge of current SOC analysts, we chose an approach that is similar to what is actually done for alert fusion: slicing the time into small windows, and then using logic in the form of rules to separate events describing different actions. Fig. 3.5 gives an overview of the complete process.

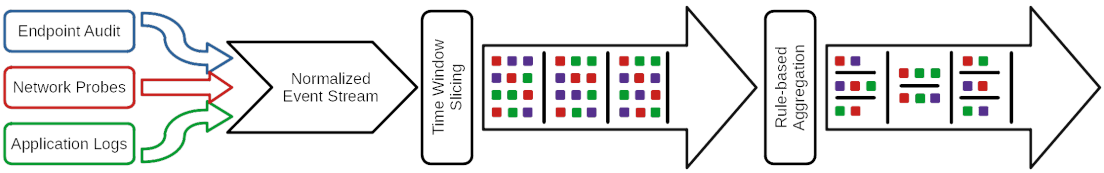


Figure 3.5 – Overview of the fusion steps

Group by time proximity

While the asynchronous nature of modern computing may render the strict ordering of the events via their time-stamps highly improbable, the various sensors deployed inside the monitored system are likely to react to the same action within a short period of time (at least if the whole system is synchronized using the same time server, or if the log shipping system is in charge of setting the time-stamps). Therefore, the first step we perform is to regroup events appearing inside the same time window. The size of the window is configurable as, depending on the tools available, the time between the first event and the last event matching the same action can vary from milliseconds to a few minutes (e.g., for NIDS that trigger

alerts at the end of the connections). Each slice of time can be handled completely in parallel.

Correlation of meta-events using rules

Linking security events based on causal dependencies can ease the investigation process [122]. Indeed, each action inside an IS, such as attacks, is observed from various sensors. Each one records different pieces of information about the root action. We hence aim to uncover the root action through event correlation of given groups of events. This correlation is based on two assumptions:

1. Regardless of the data source, two events coming from the same origin (e.g., same process, same host, same user, etc.) and within a short period of time are more likely to share a common cause. Indeed two correlated events can be either causally linked, caused by the same cause or a coincidence. The probability of a coincidence is lower when analysing finer grained events (e.g., endpoint monitoring, syscall auditing, etc.).
2. Some event attributes are common to multiple data sources [68, 91], which enables us to regroup events from heterogeneous sensors. For instance, an endpoint auditing mechanism can log sockets opened by processes which can be linked with NIDS logs using the source IP address, and other endpoint logs using the process identifier. For some tools, some attributes are specifically designed to link events together (e.g., the zeek network probe [89] events have attributes that are meant to link each analysers results to the original network flow).

Meta-events with attributes indicating the same origin are therefore pieced together. The list of such attributes is called a correlation rule. From a threat hunting analyst perspective, defining such a correlation rule is similar to explicitly specifying pivot variables.

A rule is composed of one or more sub-rules. Each sub-rule applies to a list of data sources (e.g., network flow and process sockets, NIDS and proxy logs, etc.), and has a list of attributes that should all be strictly equal from one event to another. Additionally, a sub-rule can contain Boolean filters, to specify which events should be correlated. For example, HTTP(S) proxies do not necessarily record the source port of the connections. Therefore, it can be interesting to fuse connec-

tions recorded by a network probe with proxy events only if these connections are outgoing HTTP(S) ones.

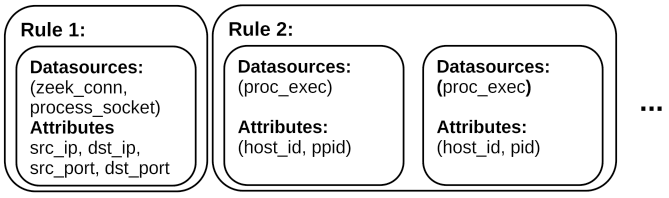


Figure 3.6 – Example of a a set of rules

In the example given in Fig. 3.6, rule 1 permits the aggregation of network connections as seen by endpoints monitoring solutions and network probes (i.e., network flows that has the same source and destination IPs and ports). Rule 2 aggregates parent and child processes execution.

3.3.3 Scalable event fusion algorithm

To analyse large volumes of events, we aim at taking advantage of parallel and distributed computing whenever possible. To do so we chose to adhere to the map-reduce standard when designing our meta-event regrouping algorithm. Therefore, we can either use functions that can be applied independently and return one result for each value (map), or commutative functions (i.e., $f(a, b) = f(b, a)$) that combines two values into a single one, i.e., $f(a, b) = a \bigoplus b = c$ with $a, b, c \ni E$, where E is the whole set of events (reduce).

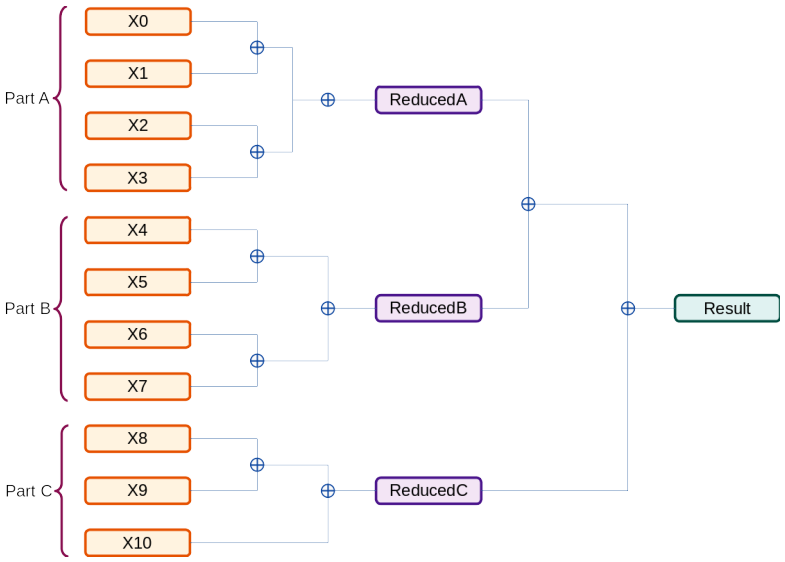


Figure 3.7 – Example of parallel processing of a reduce operation.

Reduce operators can be processed in parallel. In fact, as the order of the operands doesn't matter and the function outputs inside the same space as its inputs, we can group all values by pair, process each pair in parallel, and repeat the process until there is only one result left. However, to ensure the result is available, each worker (e.g., threads, processes, nodes, etc.) needs to be synchronized after they perform an operation. This synchronisation has a non-trivial cost, and therefore, the traditional scheme to process a reduce operation in parallel (Fig 2), is to partition the input data. The partitions are then processed in parallel and the result of each partition is finally aggregated.

In the distributed scenario (i.e., workers cannot access the same memory), each time a worker needs data that resides in another worker, it needs to be serialized and sent via network. If this operation is done too frequently, it can quickly negate the benefits of parallel processing (i.e., spend more time than if processed sequentially).

Unified data structure (MAP)

Before aggregating them into sets of meta-events, each event is structured into a common format, i.e., normalised to comply with the reduce operator expected inputs. This operator combines two lists of meta-event sets into one. As a reminder, a set is a collection of one or more (potentially infinite) distinct elements. We use

the notation `{elem1, elem2, ...}` to represent sets.

A meta-event set is, therefore, a group of one or more distinct meta-events, from one or more data sources. It is composed of a **timestamp** which is the average value of all the events that it contains, the value of the attributes of all the meta-events that compose it (**meta-events data**), organised by data source, and the **pivot** variables that are used to aggregate the events.

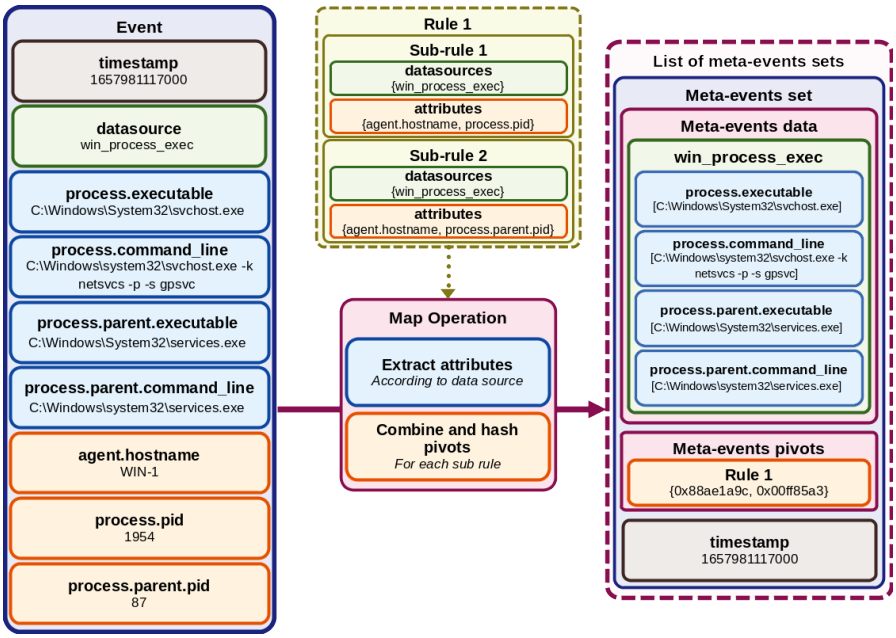


Figure 3.8 – Overview of the map operation with a process execution event.

The map operation consists in extracting the attributes of the input event and organising them into the desired structure, effectively creating, for each event, a list of meta-events sets composed of a single event. To create the pivot variables, for each applicable sub-rule in the rule set, the corresponding attributes are extracted, combined and hashed with any algorithm that can be used to create a hash map (e.g., xxHash [22], MurmurHash [50], etc.). These hash values are then organised by rule identifier, as sets of values. Fig 3.8 illustrates the behaviour of this map operation.

Regrouping events into sets of meta-events (REDUCE)

Algo. 2 describes the reduce operator that we use to aggregate events. It takes as input two lists of meta-event sets ($MEGL_1$ and $MEGL_2$) and outputs a single one.

Algorithm 2 Reduce operation for meta-event grouping

```
function MetaEventGroupListReduceOp(MEGL_1, MEGL_2)
    result ← MEGL_1
    for all MEG in MEGL_2 do
        FirstMatchId ← −1
        IdsToRemove = ∅
        for i < result.size do
            if MatchCorrelationRules(MEG.attr, result[i].attr) then
                u ← MergeMEG(MEG, result[i])
                if FirstMatchId < 0 then
                    result[i] ← u
                    FirstMatchId ← i
                else
                    result[FirstMatchId] ← MergeMEG(result[FirstMatchId], u)
                    IdsToRemove.append(i)
                end if
            end if
        end for
        result.DeleteIndices(IdsToRemove)
        if FirstMatchId < 0 then
            result.append(MEG)
        end if
    end for
    return result
end function
```

The *MatchCorrelationRules* function consists in checking if there is an intersection between any of the sets of pivot variables (one set for each applicable fusion rule) for two meta-event sets. If so, the two sets can be merged (*MergeMEG*), by combining the list of events within each meta-event sets, and removing duplicated values (i.e., events with the same values for all the analysed attributes).

A note on scalability

In the best case scenario where all the events can be regrouped into one meta-event set, the inner and outer for loops of the reduction operator always have one iteration, resulting in a $O(n)$ time complexity (with n the number of events) if processed sequentially. In the worst case, each event belongs to its own meta-event set, and thus, the time complexity rises to $O(n^2)$. On average, we found that the aggregation achieves one to two order of magnitude reduction (an average of 10 to 100 events per meta-event set), which leads to a $O(nlog(n))$ time complexity.

Also, while the number of elements in the list is reduced, the actual size of the list in memory is only slightly reduced (i.e., only duplicated events and common pivot variable values are pruned). Therefore, the algorithm cannot scale indefinitely with more computing node. Our own testing resulted in diminishing returns after more than 10 processing workers, but the true scalability depends on multiple factors (the implementation, the monitored system, the chosen rules, etc.). On the other hand, each time window can be processed completely independently, and thus scale indefinitely, at the cost of additional latency. For example if it takes at most 2 minutes to process a 1 minute time window, 2 workers would not be saturated, but results would be delayed. For large volumes of events (more than a million events per time window), scalability could be improved by organising events into independent sets with (almost) no interdependencies (e.g., logs from each sub-networks, each endpoints processed independently, etc.).

3.4 Dynamic auto-encoder for anomaly scoring and clustering

Grouping together events in sets of meta-events offers a better understanding of observed phenomenon. We go further by prioritizing sets of meta-events with

neural network-based anomaly detection system by computing an anomaly score for each set and the meta-events that compose it.

3.4.1 Model inputs

As described in Sec. 3.3, the inputs of the model are sets of meta-events, which are then transformed into numerical vectors (using method descrined in Sec 2.3). Each meta-event is composed of one or more identical events appearing in the same period of time that are extracted from a data source. Most sets only contain events from some of the data sources, and the number of meta-events for each data source from one set to another is likely to be different (Fig. 3.9). Moreover each data source have multiple attributes of potentially different types (categorical, numerical or string attributes).

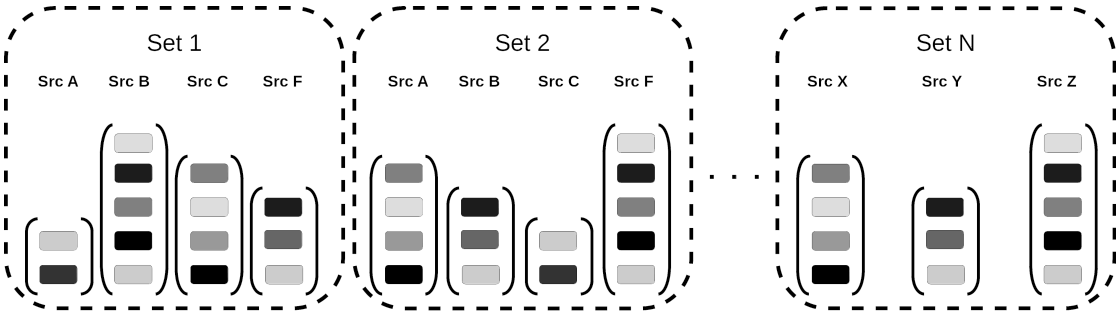


Figure 3.9 – Schematic view of the model inputs

3.4.2 Block-based architecture to handle heterogeneous attributes

We build upon the work described in Chapter 2 to handle the heterogeneous attributes inside the meta-events. In fact, using neural networks makes it possible to integrate embedding functions for categorical and string attributes (numerical features are handled by any machine learning algorithms), and let the network adapt the parameters of these functions directly during training. To improve readability, we summarize here the description of the blocks we employ to build the auto-encoder's structure (detailed in Chapter 2).

For categorical values, we use a *word2vec* inspired embedding function which maps the integer indices of the categories to a continuous vector space where

categories appearing in the same context are close.

Strings are treated as natural language using Transformer networks that exploits multi-head attention, which showed good results in Natural Language Processing. In the context of security events, they are effective at capturing syntax and semantic of string attributes and compress them into smaller vectors.

Once each attribute of a meta-event is transformed into a continuous float vector, the vectors are concatenated and processed by fully connected layers of neurons. This allows the network to find correlations between the input attributes and in turn compress the input data into a low dimension representation. From this low dimension, the network reverses the encoding process to reconstruct as precisely as possible the input attributes.

3.4.3 Dynamic structure with coherent encoded representation

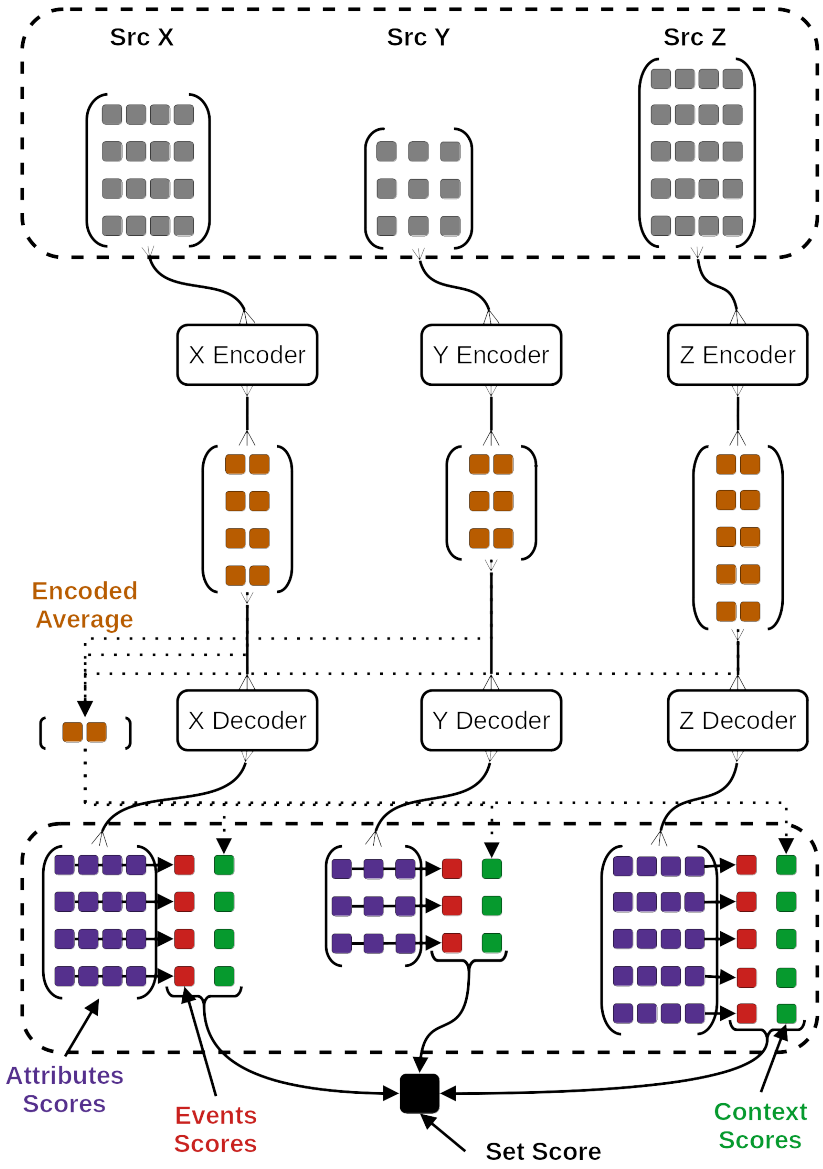


Figure 3.10 – Overview of the anomaly scoring for a set of meta-events

For each data source, an encoder and a decoder are initialized at the beginning of the training phase. One of the limitations of the solution described in Chapter 2 is that the compressed representations of the events are not comparable from one source to another. Besides, each source of events needs its own model, which can

be difficult to manage once the number of sources grows too high. We propose to alleviate these limitations using a dynamic network structure combined with a penalty added to the objective function. It encourages the encoder to encode meta-events appearing frequently in the same sets into close vectors in the encoded space. Fig. 3.10 provides a functional overview of the model.

For each set of meta-events, we select the encoder-decoder pairs corresponding to each data source in the set and build an auto-encoder out of them. In addition to the reconstruction error (i.e., the difference between the original input and the output of the network), we compute the distance between the encoded representation of each meta-event of the set and the average value of these encoded representations for the whole set. We derive the **context score** from this distance, and minimizing this score is added as an objective for the encoder, which encourages it to compute an encoded representation that is close for meta-events that frequently happen in the same context (e.g., an HTTP request is often accompanied by a DNS query).

3.4.4 Computing anomaly score

As described in Sec. 2.4.6, the score of each attribute of an event is computed from the reconstruction error of these attributes. The parameters of a logistic distribution are optimized directly within the neural network, by adding constraints 2.15 to the loss function. Thanks to this, each attribute has a score between 0 (normal) and 1 (anomalous). The context score is weighted similarly to an attribute score.

To compute the score of a meta-event, a small part of the neural network is dedicated to weighting each attribute, to lower the importance of frequently anomalous attributes (e.g., a completely random field), and increase the significance of anomalies on more predictable attributes.

The score of a set is the weighted mean of the scores of all the meta-events that constitutes it. The weight q_i for a meta-event is $q_i = \frac{s_i}{\sum_j s_j}$ with s_i the score of the i^{th} meta-event in the set. Using this method to compute the weight increases the importance of a small number of highly anomalous meta-events in the set. This can improve the robustness against some mimicry attacks, where attackers dilute their behaviours inside a high number of seemingly normal events (e.g., a thousand connection to frequently accessed websites for one communication with

the Command and Control server).

3.5 Handling concept drift

We take into account the evolution of behaviours through time which requires training models continuously. To avoid integrating adversary behaviours within the model (which prevents detection of said behaviours), we propose an active learning setup, where human analysts provide feedback in the form of annotated false positives. By combining clustering and the history of annotated alerts, we limit the number of alerts to be verified by analysts, and reduce alert fatigue by limiting redundant alerts.

3.5.1 Overview

Normal behaviour of an IT system is bound to evolve with time, as new behaviours appear and old one cease to manifest. Indeed, new users are added, old ones are removed, software are updated, etc. In data science, this phenomenon is called concept drift and is handled by updating the model. In the context of security monitoring this phenomenon can manifest in different ways. First, correlations between variables evolves (e.g., a known user started using a known command) and/or the correlation rules used for regrouping events into sets of meta-events (data sources do not change, only the composition of the sets) need to be changed. In this case, only the model weights needs to be updated, and as neural networks learn incrementally by nature, it is only a matter of performing a few training steps on data containing the new behaviours.

The second manifestation of concept drift is a modification of the input space which requires a modification of the pre-processing functions parameters (e.g., a new user, a new software deployed, etc.). Finally, in some cases the data sources schemas must be updated (e.g., new type of sensor deployed, modification of the analysed attributes, etc.). By combining the neural networks capacity to learn incrementally and the design in functional blocks (for each attribute of each data source), we can handle concept drift without requiring a costly complete retraining of the model. This can be assimilated to transfer learning, which consists in adapting for task a model that has originally been trained for another task.

Kirkpatrick et al.[57] proposed the Elastic Weight Consolidation (EWC) algorithm in order to attenuate catastrophic forgetting of neural networks, i.e., the network completely forgets older normal behaviours too quickly. EWC consists in adding penalties and constraints on the network's weight during training to avoid modifying weights that are essential to solve the previously learned tasks. Authors have also proposed the use of small sample of previous data either as a small knowledge base constituted during training and used during inference (episodic memory) [77, 107], or simply by selecting a sample of past experiences when training on new data (experience replay) [96]. We chose this second approach as it is simpler (computationally speaking) than episodic memory, and it also applicable to the adjusting of the transformation function parameters (EWC is only useful for network weights).

3.5.2 Active learning setup

In the Machine Learning domain, active learning is a special case of semi-supervised learning, which consists in asking a human expert to provide feedback to the model to improve the latter. Usually, this consists in selecting a restricted set of specific points (e.g., points close to the decision boundaries) and ask the expert to annotate them with their label (i.e., the class they belong to). In this section, we present an adaptation of this process to security alerts annotation for continuously training anomaly detection models.

Annotation process

While the annotating alerts from rule-based and anomaly-based detections seem similar, as anomalies are not necessarily symptomatic of malicious activity, anomaly detection can lead to higher false positive rates, especially if targeted at novel threat detection, where finer grained events need to be analysed. This means we need to ensure we do not saturate analysts capacities to qualify alerts. Also, repetitively annotating similar alerts is known to cause the "alert fatigue" problem. This prevents analysts from detecting subtle differences in alerts, and cause annotation errors. Therefore, we need to reduce the volume of anomalies that are presented to analysts, avoid redundant alerts, and simplify the alert understanding. This last objective is handled by the contextualised nature of the

meta-event sets that are analysed (i.e., all the events describing the same actions are provided).

To reach the other objectives, we combine two methods. First, meta-event sets are clustered by taking advantage of their encoded representation (Sec. 2.4.5). While the cluster identifier obtained by considering the encoded representation as a vector of bits (representing an integer) is easy to use and provides good results for clustering events, for meta-events, we had better results with density-based clustering methods (e.g., OPTICS [6], DBSCAN [39], HDBSCAN [78], etc.). In fact, due to varying number of events within meta-events, similar meta-events often end up having a different (but close) latent representation. Density-based clustering methods are better suited to regroup these types of similar meta-events, and thus further reduce the number of clusters.

Then, we drop some of the clusters to keep only the most anomalous ones to analyse. This is done by setting a threshold for the anomaly score, above which a set is considered relevant for human analysis. Usually, this threshold is adapted by the analysts, depending mostly on the volume of alert they can investigate, but also on their tolerance to false negatives (i.e., anomalies that do not generate alerts) as a higher threshold will potentially miss more anomalies, and thus analysts need to arbitrate whether or not they want to reduce false positives rate at the expense of raising false negatives rate.

We store all the previously annotated alerts inside a database. This database is used to verify new alerts automatically (without an annotation required from the analysts). This way, if a similar alert as already been annotated as a false positive, it is not presented again to analysts and can also be considered a false positive. On the other hand, if it has already been annotated as an unwanted behaviour, its priority can be increased.

Once the number of unitary elements (i.e., clusters of alerts) has been reduced to an acceptable level, they can be presented to analysts that will investigate them and qualify them (i.e., define if they are false positives or unwanted behaviour). This task can greatly benefit from visualisation methods as they can focus analysts eyes on the relevant informations. In our case, we do not aim at defining such visualisation methods as they are an entire research subject by themselves [23], and instead focus on providing metrics (i.e., the various anomaly scores) and additional data (i.e., cluster identifier and links with the original events) that can support

these visualisations.

Generating incremental training dataset

When the number of false positives inside the annotated alert database is sufficient to justify a incremental training step (i.e., at least a few hundreds of meta-events, as deep neural networks learn more efficiently from larger training datasets), we can use them to perform an incremental training of the model.

However, as explained in Sec. 3.5.1, to avoid forgetting old behaviours too quickly, we create an incremental training dataset composed of both false positives and historic normal behaviours from baseline database. This database is initially composed of the initial training data of the model, and is incremented with new data using the approach:

— For all new meta-event set that is considered normal (i.e., with an anomaly score that is low enough to not generate an alert), if it belongs to a cluster inside the baseline data, it is added to the database, otherwise it is discarded.

— All newly annotated false positives are reinserted inside the baseline.

To perform an effective frog-boiling attack, in addition to generating inputs that are below the alerting threshold, attackers would need to ensure these inputs are sufficiently similar to baseline to be considered normal. While not impossible, this significantly increases the complexity of such an attack.

To save compute time, and permit forgetting obsolete behaviours, we avoid retraining with the whole baseline dataset, and instead take a sample from it. To select this sample, we first organise the baseline dataset into clusters. For each of the cluster, we consider the time t_i corresponding to the latest meta-event set contained within this cluster. When drawing a meta-event set from the baseline, we randomly select a cluster to pick from. The probability of picking from a specific cluster is given in 3.1. Essentially, clusters consisting in elements that haven't been seen in a while are less likely to be selected than more recent ones. The sample size is given as either 1000, to constitute a sufficiently large training dataset, or as twice the number of false positives to be incorporated in the model, whichever is the highest, to ensure novel behaviours are not overrepresented inside the incremental training dataset, which could cause bias toward new behaviours, and cause false positives from older ones.

$$w_i = \frac{t_i - t_0}{t_n - t_0} \tag{3.1}$$

The complete active learning loop, from the new meta-event sets to analyse to the generation of an incremental training dataset, is summarized in Fig. 3.11.

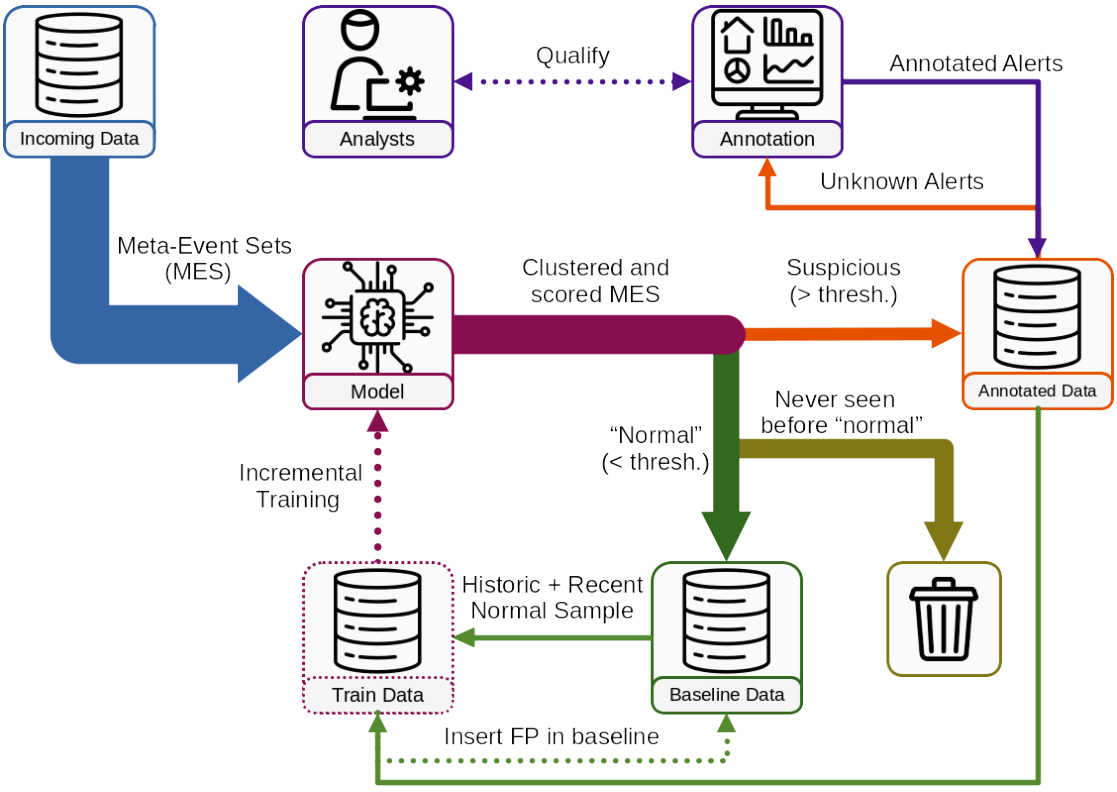


Figure 3.11 – Overview of the active learning loop.

3.5.3 Updating input transformation parameters

For numerical attributes

In this case, we use the same method described in section 2.3.2 to find the scaling factor but for the incremental training dataset (composed of both new data and the sample of historic data). As the new factor, we take the average between the previous value and the newly calculated one. This avoid changing the scaling factor too brutally in case the new data is very different from the historic data.

For categorical attributes

New categories are likely to appear as time goes by. This is especially true if the attributes describe low level entities (e.g., an executable file path in the local PATH of a user). As new categories are mapped to the "unknown" category by default, it is possible that the majorities of categories appear as "unknown" after some time. Also, changing the number of integer index of the categorical transformation function requires changing the embedding layer structure (the number of parameters is proportional to the number of possible index). To avoid having to change the network's structure too frequently, we propose to initialize the categorical embedding with more potential index than there actually is inside the training data. This way, new categories can be added to the categorical-to-integer mappings without requiring network structure changes.

For string attributes

Similarly to the categorical attributes, changing the total number of tokens of the tokenizer requires changing the structure of the network. Therefore, we chose not to change the total number of tokens, and simply replace the least frequent old patterns with the new more frequent ones.

3.5.4 Updating network structure

When a change in the network structure is required, it is still interesting to keep as much of the original network as possible. In fact, this form of transfer learning is an effective way to accelerate training by only requiring some parts of the network to be updated.

The methodology we use to build the network structure is abstracted via functional blocks. Indeed, the network is composed of pairs of encoder-decoder (one pair for each data source), each encoder is composed of an embedding block for each attribute of the event source and a fully connected sub-network, and symmetrically, each decoder is composed of a fully connected sub-network and a reconstruction block for each attribute. Adding or removing a data source is simply a matter of adding or removing the encoder-decoder pair.

When modifying the attributes of a data source, it is also required to modify the first layer of the fully connected sub-network of the encoder. In fact, as the

effective operation rely on a product of matrices, the number of neurons of the aforementioned layer must always be equal to the sum of the number of outputs of each embedding blocks. For the same reason, modifying the number of categories of a categorical attribute requires to modify the embedding block and the reconstruction block of this attribute.

3.6 Summary

In this chapter we presented a set of methods to build security analytics that combine expert knowledge (in the form of detection rules and scripts) with contextualised anomaly detection and clustering models, enabled by a scalable event fusion algorithm and auto-encoders with dynamic structures.

To maintain these analytics, we propose an active learning approach. Analysts are asked to provide feedback similarly to what they do when qualifying alerts as either false positives or true incidents. Doing so, we reduce the risk of an attacker progressively poisoning the model by performing actions that are not anomalous enough to trigger alarms and slowly evolve towards the desired adversary actions (i.e., frog boiling attack). We focus on reducing the number of redundant alerts presented to the analysts by using the clustering capabilities of the model.

In the next chapter, we will present our method to build more representative datasets. We will also provide experimental results on a custom dataset, with an implementation of our approach.

Chapter 4

Assessment

In this chapter, we describe the dataset we use to assess our approach, and the method we used to create it. Specifically, we designed a realistic user simulation method for lab environments that do not interfere or leave traces inside the collected security events. This chapter also describes the implementation of our approach as well as the results we obtained on our dataset.

4.1 Generating realistic user activity for dataset collection

Datasets that are representative of real environments are required to assess detection methods. However, for security monitoring, collecting these datasets remains an open challenge. Simulating user activity is one of the biggest problem that needs to be solved to generate a dataset for security monitoring method assessment. In this section, we propose an approach that instruments machines with external agents in lab environments in order to simulate user activity at the system level, without leaving traces of the agent within collected events.

4.1.1 Introduction

As explained by Xosanavongsa [121], assessing a method that analyses heterogeneous security events requires a dataset with multiple difficult-to-obtain properties. First, such a dataset requires to collect logs from heterogeneous sources (i.e., network, system and applications events). In our case, we focus on operational use-cases and therefore, we should be agnostic from the monitoring sensors as they can be any COTS that are already deployed in production environments (with only slight modifications to the configurations). Second, to highlight the complex nature of advanced adversaries activity, the dataset should contain attacks performed in

multiple steps, on multiple machines, at all the levels of the IT system. And finally, the biggest challenge is that the dataset should contain traces of activity related to normal user actions on the system. This is important to constitute a control group for any attack detection methods (i.e., to ensure False Positive Rate is contained to an acceptable level), and it is completely mandatory for anomaly-based detection methods which constitute baseline of legitimate activity on these traces.

To the best of our knowledge no publicly available datasets satisfies these criteria. Furthermore, considering the sensitive nature of security data which can contain personal information, regulations and contracts often prevents SOC from sharing production environments datasets for research purposes. Also, to verify the effectiveness of the proposed methods in various conditions, adversary emulation would need to be performed on the clients monitored systems (authorisation is unlikely to be granted).

To address one of the biggest challenges in the security analytics research domain [116, 3, 102], we need to be able to create datasets dynamically (i.e., ability to change the monitored environment, the monitoring strategy, the emulated adversaries, etc.). As having traces of the normal user activity is mandatory for such a dataset, in this section, we propose a method to automate the simulation at scale of these interactions on lab environments (e.g., a cyber range). As such a system is similar to user-behaviour simulation in sandboxes [41], we also consider its use for users simulation in detection laboratory (i.e., environments specifically designed to analyse adversary behaviours).

Automatic realistic life generation on IT systems is a complex subject that requires to solve multiple problems. First, instrumenting machines (to perform actions) should not leave detectable traces on them. In fact, to collect representative dataset for security monitoring, the security events resulting from simulated user activity should not be discernible from the ones resulting from real user activity. To integrate the life generation method to a detection laboratory, this also supposes that an observer (e.g., an attacker) cannot accurately and quickly determine that the interactions are not performed by humans using an automated test (reverse Turing test [113, 17, 1]). Second, pre-defined scenarios, which are required to control the nature of the interactions and life that is produced, do not take into account random behaviours and noise in the lab environment (e.g., position of a window has been changed, pop-ups, a software crash, etc.). Managing these

behaviours and noise requires recognition and adaptation in real-time (i.e., similar to the human user reaction time) in order to prevent the scenario from failing or being interrupted. Similarly, the sequence of actions that constitutes the simulated activity should be performed in a human-like timing. Finally, to simplify the implementation of life generation scenarios at the scale of an entire IT system, the human operators that control the simulation should be assisted during the creation of said scenarios.

The approach described here is inspired by the life generation methods employed in the BEEZH platform [42]. We aim at addressing multiple limitations. First, we gain in modularity (e.g., not bound by the virtualisation technology, adaptable to physical machines, etc.) by organising the design of the agent that interacts with the machines in multiple abstraction layers. The lowest abstraction layers exposes mouse, keyboard and screen interaction in a unified way, regardless of the actual interaction method. Adapting to various graphical environment is the role of the middle abstraction layer, and a higher level of abstraction is provided as a Python API to build activity scripts that are composed of unit actions (e.g., open a web, search for a list of keywords, etc.). We use image analysis methods (both deterministic and based on statistical models) to allow the agent to automatically adapt to minor changes in the environment (e.g., a window is moved, screen definition changes, etc.). We propose an action builder to assist human operator in creating and adapting action to their specific needs and environments. Finally, we describe preliminary work on scheduling and building large scale life simulation scenarios.

4.1.2 Related work

The simulation of user interaction with the machines of a lab environment is at the cross-roads of three domains. First, graphical user interface automation has been studied mainly for automated quality testing of software. We can distinguish two major approaches, namely pre-recorded action replay [123, 106, 84], that requires long configuration time, and full automation, that seeks viable inputs (i.e., mouse and keyboard) combinations [99, 41]. Both of these approaches rely on an agent installed on instrumented machines. This agent would be a strong indicator for an attacker, and would taint the event logs for our dataset generation use case, and it can therefore not be applied to our problem.

Second, to defeat malware performing reverse Turing tests, user simulation has been studied for sandbox environments. A recent approach, MORRIGU [81], instruments virtual machines through the VirtualBox API from a Windows host. Although malicious behaviours detection results are encouraging, these solutions are not designed to scale to more than a single sandbox environment, and therefore are not applicable to large scale IT system simulation.

Finally, as an extension to sandboxes and honeypots, user simulation has been studied to improve realism of honeynet platforms (i.e., controlled IT system designed to attract and detect attackers, as well as analysing their behaviour). The user simulation tool that best fits our requirements has been integrated to the BEEZH [42] honeynet platform. It controls the machines through the VNC server that is exposed by the hypervisor, and thus no agent is installed on the machines. This tool executes life scenarios that are composed of simple unit actions (e.g., open the web browser, look for some keywords in a search engine, etc.). In addition to BEEZH, to automate GUI element recognition, the open-source desker library [4] permits the use of image analysis methods derived from the computer vision domain. Using VNC allows a wide panel of interactions with the controlled machines. However, to ensure that the life generation does not interfere with the behaviour of the system, using VNC restricts the use-cases to instrumenting virtual machines. Indeed, installing a VNC server on a bare-metal machine would interfere with the normal network behaviour. This life generation method also requires the operator to manually adapt the scenarios to the specificities of the instrumented system (i.e., specific software versions, themes, etc.). Finally, the chosen image analysis method (Faster-RCNN [95]) requires large amounts of computing power, which is incompatible with our performance and scaling constraints.

Similarly to BEEZH, our life generation agent can only get feedback from the screen of the controlled machine. Most actions require to click on specific zones of interest on the screen. Faster-RCNN [95] is widely used for this. Nowadays, RetinaNet [72] and SSD [75] provide better results with shorter processing time. Speed can be further improved by using YOLO [94, 93], to the expanse of results quality.

4.1.3 Life generation agent design

The agent is in charge of translating user activity scenarios into low-level actions (keyboard, mouse and screen), and to perform theses actions on the instrumented machines. To simplify the configuration of the user simulation tool by human operators, we chose an approach in multiple abstraction layers (Fig. 4.1).

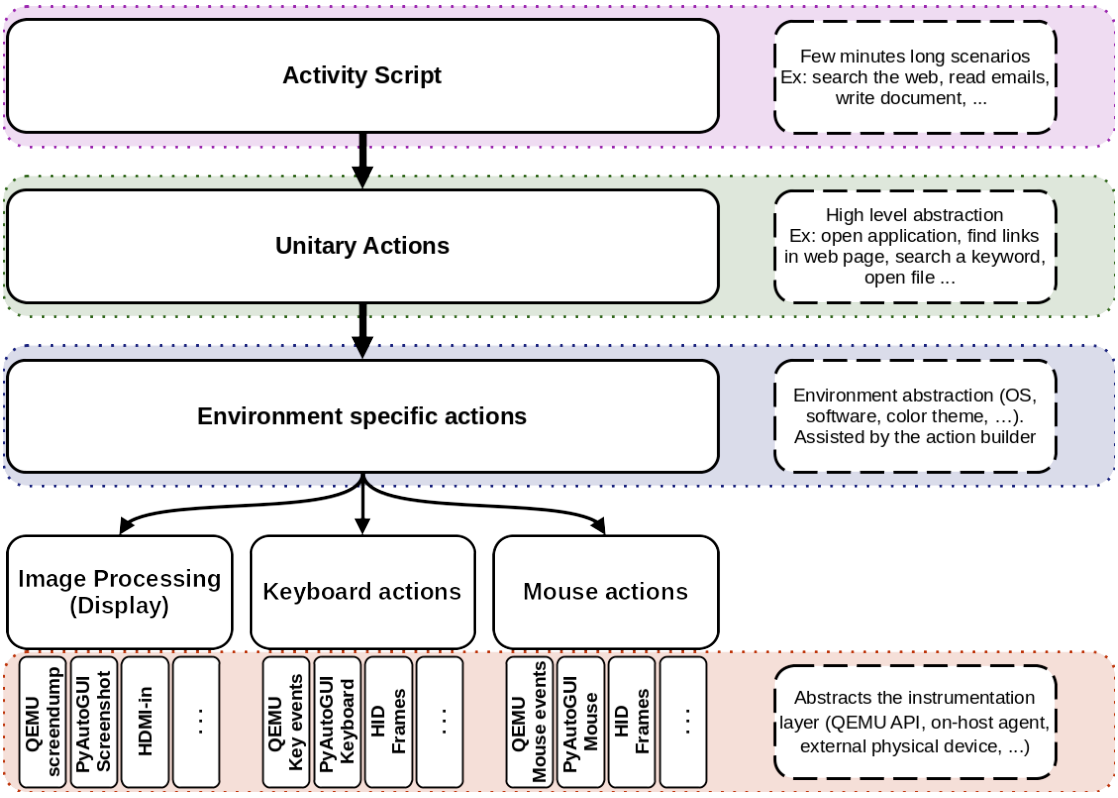


Figure 4.1 – Functional architecture of the agent

At the lowest level, a virtual user interacts with an instrumented machine through the mouse, keyboard and screen. We aim at adapting the agent to multiple virtualisation platforms (e.g. VMWare ESXi, Proxmox, etc.) as well as physical machines, without reconfiguring the life scenarios. We focused on three interaction methods, namely the Qemu monitor API [1], VNC, and as a fallback, through an agent directly installed on the instrumented machine. We also experimented with physical machine instrumentation by emulating USB keyboard and mouse according to the HID protocol [2], and video capture. However, additional work is

1. `https://qemu-project.gitlab.io/qemu/system/monitor.html`
2. `https://www.usb.org/hid`

required to ensure this method is reliable. This abstraction layer exposes basic interactions capabilities to the layer above, namely, moving mouse to specific coordinates, pressing and releasing mouse buttons and keyboard keys, and taking a screen shot.

We add random perturbations to these interactions in order for them to look natural to an observer. For mouse movement, these perturbations consist in moving the cursor gradually (i.e., not teleporting it directly to the desired location), adding momentum to this gradual movement (e.g., a long movement will result in the cursor getting a bit further than the desired point) and adding random deviations from the target direction. Until the desired destination is reached, the same process is repeated. For keyboard interactions, the perturbations consist in adding random delays between keystrokes.

The abstraction layer right above the interaction layer is meant to adapt to the instrumented machine's environment specificities. Indeed, while the image analysis methods that are employed (see Sec. 4.1.4) are almost insensitive to minor graphical variations (e.g., different screen resolutions, slightly different colours, etc.), some more important variations requires to target specific images on screen. For example, the Thunderbird and Outlook mail client have similar functionalities, but their user interface is completely different, which requires different interactions with the machine.

The final abstraction is used as an Application Programming Interface (API) for life simulation scenarios. It consists of unit actions that can be used for building longer scenarios (e.g., open web browser, browse a web page, identify links in it, etc.). We implemented the agent using the Python programming language, and therefore all the life generation scenarios are defined as Python scripts.

4.1.4 Analysing screen captures

To control the state of the machine in real-time, the agent only has access to the screen. Therefore, we use image analysis methods on screenshots to recognize area of interest, such as user interface buttons to click, links in a web page, etc. One of the biggest constraint we have is to have high computational efficiency, as we want to scale the method to large instrumented systems. For this reason, whenever possible, we limit the quantity of information to process (i.e., the size and the number of images), and the computational complexity of the techniques

we employ. In particular, we rely on three image analysis methods.

The first one is known as *template matching*. It consists in finding in an image (i.e., a screenshot), the points that are the most similar to the desired target. This method consumes little resources and is deterministic, but small variations (e.g., colours, resolution, shape, etc.) may render them ineffective. This technique is therefore dedicated to the detection of user interface elements that do not (or only slightly) vary for a given environment (e.g., start-up menu button, windows manipulation button, etc.).

In case *template matching* does not provide satisfactory result, we propose the use of deep learning algorithms that have demonstrated good performances for object detection. However, even the most efficient methods (e.g., SSD [75], YOLO [94], etc.) require large amount of computational resources, which can limit scalability as the number of instrumented machine increases. Moreover, there are no public datasets containing annotated areas of interest in screen captures. To collect and annotate such a dataset would require a considerable amount of time (even though transfer learning [110] can reduce the necessary amount of labelled information).

For these reasons, we chose an approach that exploits the geometry of the shapes that should be detected on screen (Fig. 4.2). Specifically, by using adaptive thresholding [71], it is reasonably simple to highlight objects in the foreground (e.g., icons, text, etc.) and the limits between the various zones on screen (e.g., windows borders). For a low computational cost, we can identify potential areas of interest with specific characteristics, by apply filtering rules on the zones identified thanks to adaptive thresholding (e.g., an icon is a small area almost as wide as high, a button often has a bigger width than height, etc.). However, as shown in Fig. 4.3, this method is prone to false positives. To filter those out, we use a Convolutional Neural Network (CNN), which is the standard deep learning structure for computer vision. Considering the method is only applied to some smaller areas of interest, the impact of the computational costs inherent to deep learning models is limited.

Finally, we use Optical Character Recognition (OCR) to navigate within the user interface (e.g., file and folders labels, menu elements, etc.) as well as detect links in web pages. The Python library tesseract [109] is used to analyse textual elements of the user interface, and we use the same method as desker [4] to detect links.

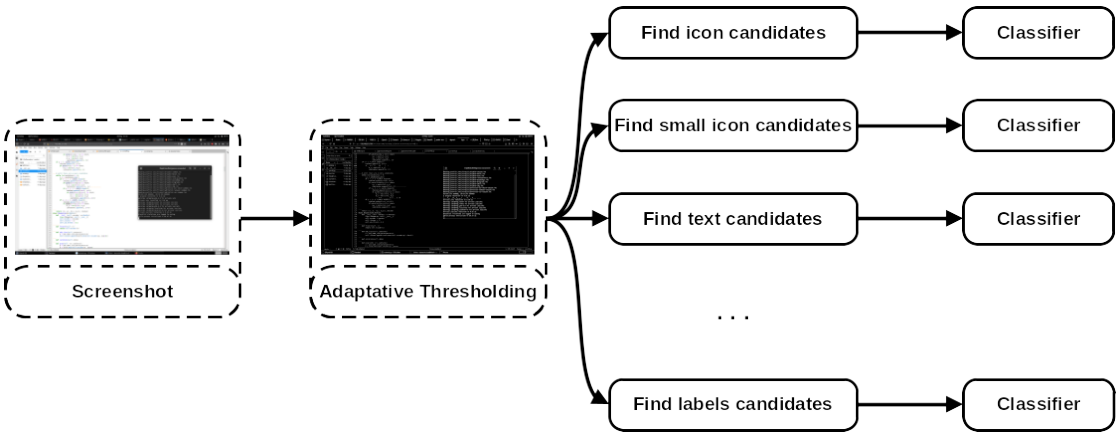


Figure 4.2 – Image analysis process.

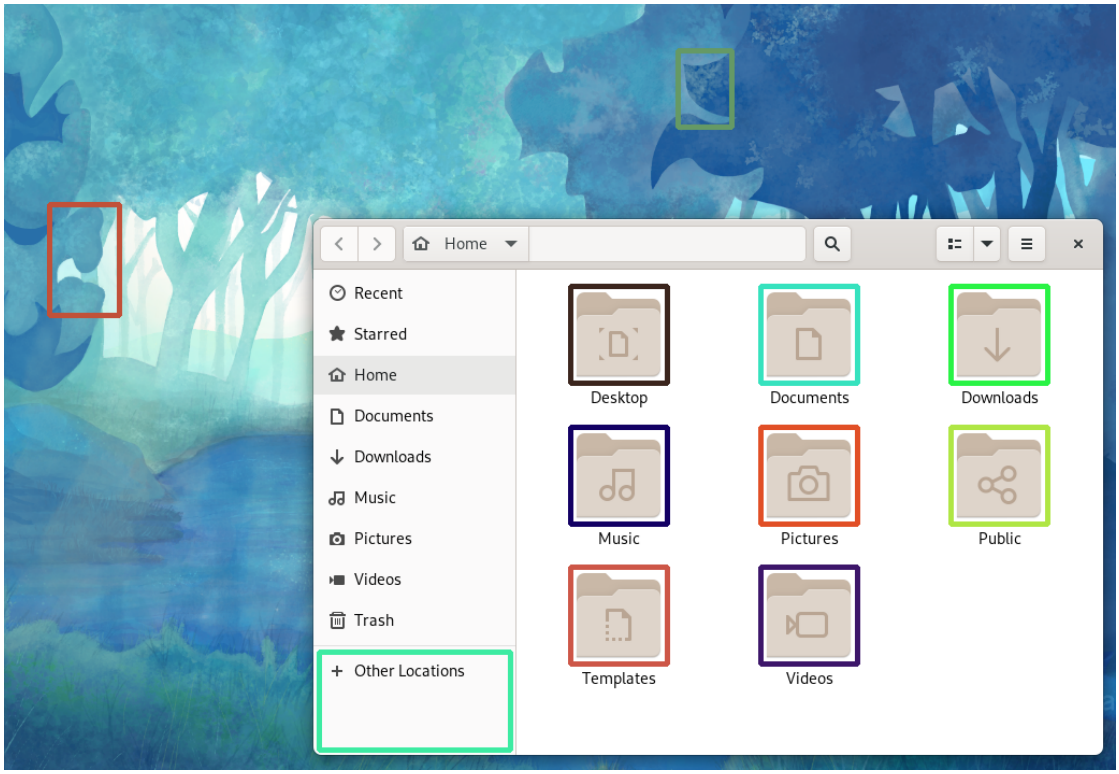


Figure 4.3 – Example of candidate area of interests detected.

4.1.5 Action creation assistance

The image analysis methods presented in the previous section allows the agent to manipulate and adapt to the environment that is instrumented. For instance, Firefox and Chrome web browsers, although having similar functionalities, have

different interfaces. Therefore, the agent will look for different visual markers to interact with these two browsers.

Using statistical methods (e.g., deep learning) allows, in theory, for a better adaptability of image analysis techniques. In practice, to reach sufficient generalisation capabilities, these methods needs to be presented with enough labelled data. For this reason, we propose two techniques to speed-up the creation of actions adapted to the specificities of environments, as well as the creation of datasets for training statistical models.

At its core, this action builder relies on recording of the mouse movements and clicks, the keys pressed and screen captures. These raw records are then aggregated, to split the records in small actions (e.g., mouse moved from A to B, then double click on B). Areas of interest on screen are extracted using two methods.

The first one consists in comparing screen captures when a mouse click is recorded and at the beginning of the preceding mouse movement. Most user interfaces highlights elements when they are hovered by the mouse cursor. Therefore, the comparison between the two captures around the coordinates that has been clicked will often show which GUI element was targeted. We automatically retrieve these elements, which can be used directly as targets for *template matching*. This mechanism greatly reduces the time needed to create new actions that the agent can perform, and also allows the operator to quickly adapt the existing actions to newly introduced changes in the interfaces (e.g., major software update).

The second method consists in re-employing the area of interest detection technique described in Sec. 4.1.4 (Fig. 4.2). This lowers the time required to annotate datasets for training neural Convolutional Neural Networks for detecting areas of interest. Indeed, the areas are preselected, and the human operator only has to label these areas (i.e., as false positive, or as a specific type of object).

4.1.6 System-wide life scenario

In the previous section, we described the inner working a single agent. In this section, we propose to widen the scope to multiple coordinated agents in order to simulate activity at the IT system level. This rely on a high level representation of user profiles and their interaction (Sec. 4.1.6), as well as scheduling agent activity in a coordinated manner (Sec. 4.1.7).

Virtual users representation

Based on the role they have in the company and the relations they cultivate with their colleagues, users will interact differently with the company's IT system. For example, two colleagues that are friends outside of work are more likely to discuss in an informal manner via email than two colleagues that do not know each other. Similarly, a software developer will use an Integrated Development Environment (IDE) more often than a manager.

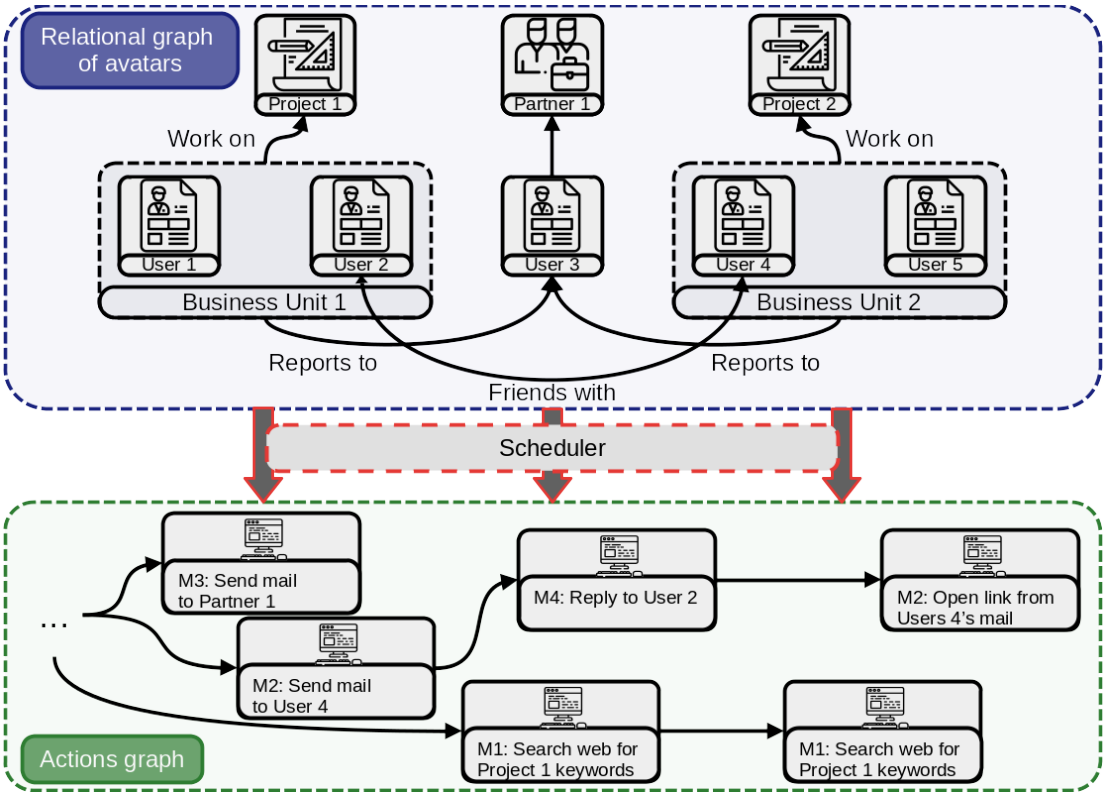


Figure 4.4 – Overview of the scheduler.

To simplify the set-up of cyber-training exercises and to ensure the generated datasets are coherent, it is interesting to create avatars for each simulated users. Each avatar is given a role in the company, a list of the relationships they have with other avatars, the projects they work one, etc. With a bit of formalism, it is possible to represent the avatars as a relational graph (Fig. 4.4), composed of the various avatars, the projects they work on, their work groups, and the list and nature of the relationships they have with other colleagues (e.g., friendly, business only, partnership, hierarchic, etc.).

In its current form, this graph is used as a graphical model to guide the operator when defining life scenarios. In the future, we envision the use of this graph as a way to automatically generate basic interactions between the avatars and the instrumented system.

4.1.7 Agent scheduling

Each agent (Sec. 4.1.3) of the life generation system corresponds to a single instrumented machine (and its virtual user). At the highest abstraction level, these agents execute life activity scripts. The scheduler's role is to coordinate these activities at the IT system scale. For example, a mail conversation between users corresponds to alternating mail reading and mail redaction activities.

In practice, a user A cannot read an email from a user B if this email has not been sent yet. To represent this logic succession of actions, as is common with many scheduling problems, we use a Directed Acyclic Graph (DAG). Each node of the graph is an activity (and its parameters) that will be performed by an agent, and each edge indicates a dependency between actions. Currently, the DAG is manually generated by the operator.

We implemented the scheduling using *redis*[3] as a message broker. Each agent retrieves the actions they should perform from its dedicated message queue and the scheduler sends work on these message queues whenever the actions can be performed by the agents. Each work order consists of an activity scenario (e.g., browse web) and its parameters (e.g., time of the activity, keywords to search, etc.).

4.1.8 Summary

In this section, we described the method we use to instrument machines on a laboratory environment. This method can simulate user activity at IT system scale. We focused on the simplicity for human operator to adapt and deploy the method for their needs. To do so, we use an agent that is divided in three abstraction layers, which abstracts, at its lowest level, the interaction method in the form of keyboard and mouse event as well as screen captures. To cope with the diversity of user interfaces in modern systems (e.g., different OS, software, etc.), the agent embeds

3. `https://redis.com`

image analysis methods based on both deterministic (e.g., *template matching*), and deep learning methods (e.g., Convolutional Neural Networks). This allows us to limit the demand on computation resources for the agent, while still taking advantage of the flexibility provided by the statistical deep learning models. We propose an action recorder to accelerate the configuration of the agent for new environment. This recorder automatically extracts targeted GUI elements, and facilitates collection and annotation of training data for statistical models.

Due to its ease of configuration and the realism of the interactions between agents and instrumented machines, this method can be used to perform large scale user simulation for building realistic datasets. As the agent interacts with the mouse, keyboard and screen of the machine, it can be completely externalised. Therefore, no agent will taint the collected security events on the machine, and actions will be performed as they would be by a human operator. We use this method to build an assessment dataset for our method.

The property of the simulation methods are also interesting for cyber-training platforms. Indeed, the realism provided by the interaction methods can, as in a production system, allow the attacker to blend in with the normal activity. Therefore, players need to focus on finding adversaries behaviours as they will have difficulties with white-listing user behaviours.

As a final note, we focused on the realism of the interactions, to the point they can defeat simple reverse Turing tests performed by malware. While a human would probably be able to detect the simulated nature of the interactions, it could prevent automated sandbox evasion that look for user interaction. Therefore, with some refinement (e.g., more realistic long term user activity scenarios, focusing specifically on sandbox evasion techniques, etc.), it could be used to simulate activity in detection laboratories that are used to analyse adversary behaviours.

4.2 Assessment dataset description

Assessing the performance of an anomaly-based security log analysis method requires the collection of enough logs to model the normal behaviour of the experimental monitored system. For confidentiality reasons, it is not possible to use production data, so the data needs to be collected from a lab environment and user interaction with this environment must be simulated. As accounting for ev-

ery possible cases of a real IT system is impossible, this simulation is bound to be biased. However, considering that each user inter-action is performed through the screen, keyboard and mouse, similarly to what a human operator can do, the collected logs will be similar to the ones collected on a production system. We designed a life generation scenario such as:

— Normal activity corresponds mostly to office tasks (e.g., web browsing, writing documents, sending emails, etc.) and is partially automated using method described in Sec. 4.1 (some interactions are directly performed by the operator);
— Administration tasks are performed manually and are likely to be treated as behavioural anomalies;
— All users have different behaviours;
— These behaviours evolve (e.g., users start conversing at some point), and the monitored system evolves through time (e.g., Sysmon is deployed on more machines after a few days), which forces anomaly detection methods to be trained continuously.

We played three attack scenarios, each one during a day, with one day of only normal activity with new behaviours between each scenario. These new behaviours are expected to cause false positives, and must therefore be taken into account to accurately detect the second and third attack scenarios. At the end of each scenario, all traces left by the attacker are completely removed. The attacker tooling is composed of publicly available threat emulation tools (e.g., PoshC2 [4], Mimikatz [5], etc.), as well as custom tools made specifically to evade detection by signature-base detection tools (i.e., Suricata and Windows Defender).

The dataset contains 4.2 million events, over a period of 7 working days. These events come from typical monitoring tools commonly found inside IT systems (i.e., Sysmon, Auditd, Windows Audit Logs, Windows Defender, Zeek IDS, Suricata IDS and Squid HTTP Proxy). We configure these tools to record a higher volume of information than what is usually deployed on IT systems (e.g., all available IDS rules, process monitoring with windows audit, socket monitoring with Sysmon and auditd, etc.). This enables us to test various levels of verbosity for assessing our method without regenerating a datasets. The dataset will be publicly available for

4. `https://github.com/nettitude/PoshC2`
5. `https://github.com/gentilkiwi/mimikatz`

anyone to reproduce the results or test new research with it.

4.2.1 Monitored system architecture

Considering that the monitoring tools we use are only found on mature systems, we need to ensure that the actions that are performed are representative of such a system (e.g., admin tasks should not be performed from user workstations). Therefore, the monitored system's architecture (Fig. 4.5) is intended to be reasonably secure (well separated infrastructure, administrator and user zones, controlled outgoing and incoming traffic, updated software and antivirus, etc.). Six simulated users perform various office tasks (documents writing, mail, web browsing, etc.) across Linux and Windows workstations while an administrator maintains the infrastructure of the system and sometimes connects to the users endpoints to install software or update configurations.

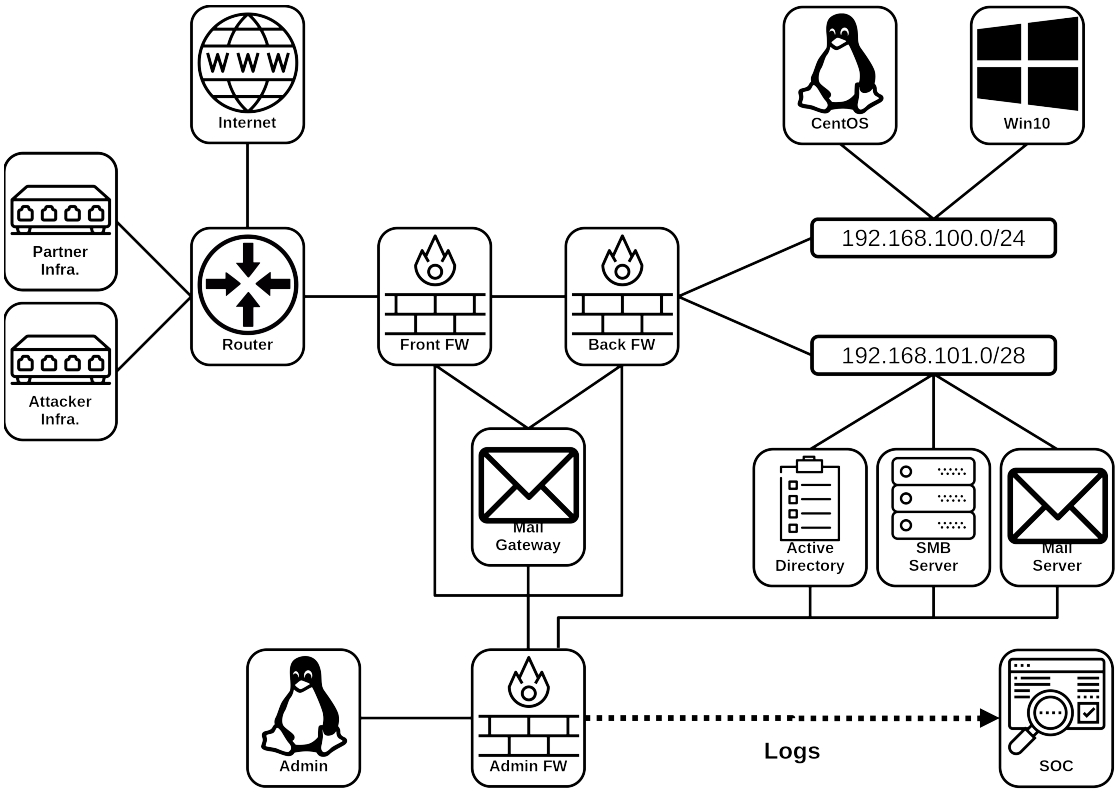


Figure 4.5 – High-level architecture of the lab environment.

4.2.2 Day by day scenario

Day 1-2 (14-15/06/2021): Baseline activity

The first two days of the dataset serves as a baseline of "normality" (i.e., without any attack traces) that is required for anomaly detection system training. The presence of attack traces inside the training data would prevent detection of said attack, but attacks employing different techniques would still be detected. During these days, in addition to the activity performed by the virtual users, the administrator connects via RDP (on Windows machines) and SSH (on Linux) to perform slight configuration changes.

Day 3 (16/06/2021): First attack scenario

The first attack scenario (day 3) emulates an attacker with custom tooling (to avoid detection by antivirus), but is noisy (multiple IDS alerts, a few antivirus alerts, more trial and errors, etc.). The approximate time-line of attacker actions is provided in Tab. 4.1.

Day 4 (17/06/2021): Novel legitimate behaviours

On the fourth day, the administrator deploys an email client (Mozilla Thunderbird) on a Centos machine. The user moves from using the web application to using the newly installed client. Two users that never exchanged mail before start to.

Day 5 (18/06/2021): Second attack scenario

The second attack scenario (day 5) reemploys the same tooling as the first scenario, but is more subtle (generates less IDS alerts, no antivirus alerts, actions are more precise, etc.). During the attack, the administrator diagnoses a problem on windows machines (and therefore generate unusual activity). The approximate time-line of attacker actions is provided in Tab. 4.2.

Day 6 (23/06/2021): New sensors and behaviours

On the sixth day, Sysmon is enabled on the remaining Windows endpoints (it was only activated on one before), and is expected to generate a large amount of

Time	Action
10:20	Phishing mail
10:45	Malicious attachment is opened
10:50	RAT is executed
11:00	Environment discovery
11:30	User access persistence
11:45	Additional malware is downloaded
12:00	Sub-net scan
12:10	Active directory enumerated
12:30	Spyware executed
12:40	Brute force local admin password
13:00	RDP tunnel through HTTP (UAC bypass)
13:00	Admin access persistence
13:05	C:\temp excluded from antivirus
13:10	Internal spearphishing
13:35	Files are stolen
14:00	Lateral movement through SSH
15:00	Dump Domain Admin hash
15:20	Pass the hash
15:45	Administrator Access on domain controller
16:15	Cleanup

Table 4.1 – First scenario timeline

Time	Action
10:00	Phishing mail
10:05	Malicious attachment is opened
10:05	RAT is executed
10:10	Environment discovery
10:15	User access persistence
10:30	Additional malware is downloaded
10:40	Active directory enumerated
11:25	Internal spearphishing
12:00	Brute force local admin password
12:00	RDP tunnel through HTTP (UAC bypass)
12:20	Admin access persistence
12:30	C:\temp excluded from antivirus
12:30	Dump Domain Admin hash
12:30	Files are stolen
13:00	Lateral movement through SSH
13:40	Cleanup

Table 4.2 – Second scenario timeline

false positives.

Day 7 (24/06/2021): Third attack scenario

The last scenario (day 7) is much more discreet than the previous ones and relies on new tools. The attacker limits its activity to a minimum to avoid generating too much events. This scenario aims at emulating a more advanced threat actor. The approximate time-line of attacker actions is provided in Tab. 4.3.

4.3 Implementation

In this section, we describe how we implemented the approach described in chapter 2 and 3. We also describe its integration within a laboratory environment that employs tools commonly used is SOC.

Time	Action
10:05	Phishing mail
10:10	Malicious attachment is opened
10:10	RAT is executed
10:15	Environment discovery
11:25	Additional malware is downloaded
12:00	User access persistence
12:00	Spyware executed
12:00	Passive sub-net scan
13:30	Active directory enumerated
13:40	Brute force local admin password
14:00	RDP tunnel through HTTP (UAC bypass)
14:00	Admin access persistence
14:05	C:\temp excluded from antivirus
14:10	Dump Domain Admin hash
14:40	Internal spearphishing
15:00	Files are stolen
15:10	Antivirus is disabled
15:20	Pass the hash
15:25	Administrator Access on domain controller
15:45	Lateral movement through SSH
17:00	Cleanup

Table 4.3 – Third scenario timeline

4.3.1 Experimental platform

To verify that our method could be integrated within operational system, we chose to implement a simplified SOC architecture (e.g., does not implement collaboration features, case management, etc.). This architecture (Fig. 4.6) revolves around the Elastic Stack, which is commonly used as a SIEM within SOC [6].

Log Shipping

To ship the monitoring logs to the SOC, we used the Elastic Beats [7], which already perform an initial normalisation of the events according to the Elastic Common Schema [37]. The tool we used are commonly deployed on production systems. On Linux machines, we used auditbeat [8] configured with auditd rules that monitors all the process executed, the network sockets opened as well as various sensitive actions [9]. For Windows endpoints and servers, we use winlogbeat [10] to ship Windows Event Logs generated by Windows audit system as well as Sysmon (configured similarly to auditd on Linux [11]). For any other log files (e.g., Suricata, Zeek, firewalls, applications, etc.), we use filebeat [12], which is shipped with many parser for commonly used log formats.

Big Data message bus

In a production system, such a monitoring strategy would generate massive amount of data. While the configuration of the tools would be adapted to balance visibility and volume (see Sec. 1.1.4), our approach is meant to accelerate log analysis and detect anomalous behaviour, and therefore, it could be used with more verbose monitoring strategies. Therefore, we chose to use Apache Kafka [13] as the entry point for all events in the SOC. In fact, this message bus has proven

6. https://www.elastic.co/security/siem
7. https://www.elastic.co/beats/
8. https://www.elastic.co/beats/auditbeat
9. Based on a modified version of the rules available here: https://github.com/Neo23x0/auditd
10. https://www.elastic.co/beats/winlogbeat
11. Based on a modified version of the rules available here: https://github.com/SwiftOnSecurity/sysmon-config
12. https://www.elastic.co/beats/filebeat
13. https://kafka.apache.org/

its effectiveness to handle massive amount of data within production Big Data architectures (e.g., 1M events per second, compared to 100k eps for large SOC).

The SIEM

To store and search security events, we use Elasticsearch [14], Logstash [15] and Kibana [16], as they are frequently used to build SIEMs. In this case, Logstash retrieves events from Kafka, enrich them with their corresponding data source (to simplify queries later), and index them inside Elasticsearch. Kibana is used as the Graphical User Interface to interact with Elasticsearch and investigate security events.

Scalable analytics platform

To implement our approach, we used Python, a language that is commonly known by security analysts and is one of the most popular among data scientists. To run it in an organised (and reproducible) way, we decided to use JupyterLab [17]. Notebooks are simple to use to prototype Python and can be organised with comments and used for data visualisation tasks. They can then be executed step by step to reproduce results. Recently, Jupyter notebooks started to get traction within the information security community, especially to automate parts of the threat hunting process (e.g., Mandiant published thiri-notebook [18]). In our case, analytics are implemented in Python and integrated within notebooks to test them with various parameters (e.g., change the time window for the fusion, the list of data sources, etc.).

To test the scalability of the method we described, we need a compute cluster. In the data science community, such a cluster is often backed by Spark [19]. However, deploying it requires to deploy Hadoop, which greatly complexifies the architecture, and is not adapted to small infrastructures (such as this lab environment). Furthermore, due to its strong roots in the Java ecosystem, using Spark might feel very unfamiliar to users used to Python (although possible using PySpark).

14. `https://www.elastic.co/elasticsearch/`
15. `https://www.elastic.co/logstash/`
16. `https://www.elastic.co/kibana/`
17. `https://jupyter.org/`
18. `https://github.com/mandiant/thiri-notebook`
19. `https://spark.apache.org/`

For this reason, we chose to use Dask [20] which interacts seamlessly with a Python environment, and can scale from a laptop to a thousand-nodes cluster.

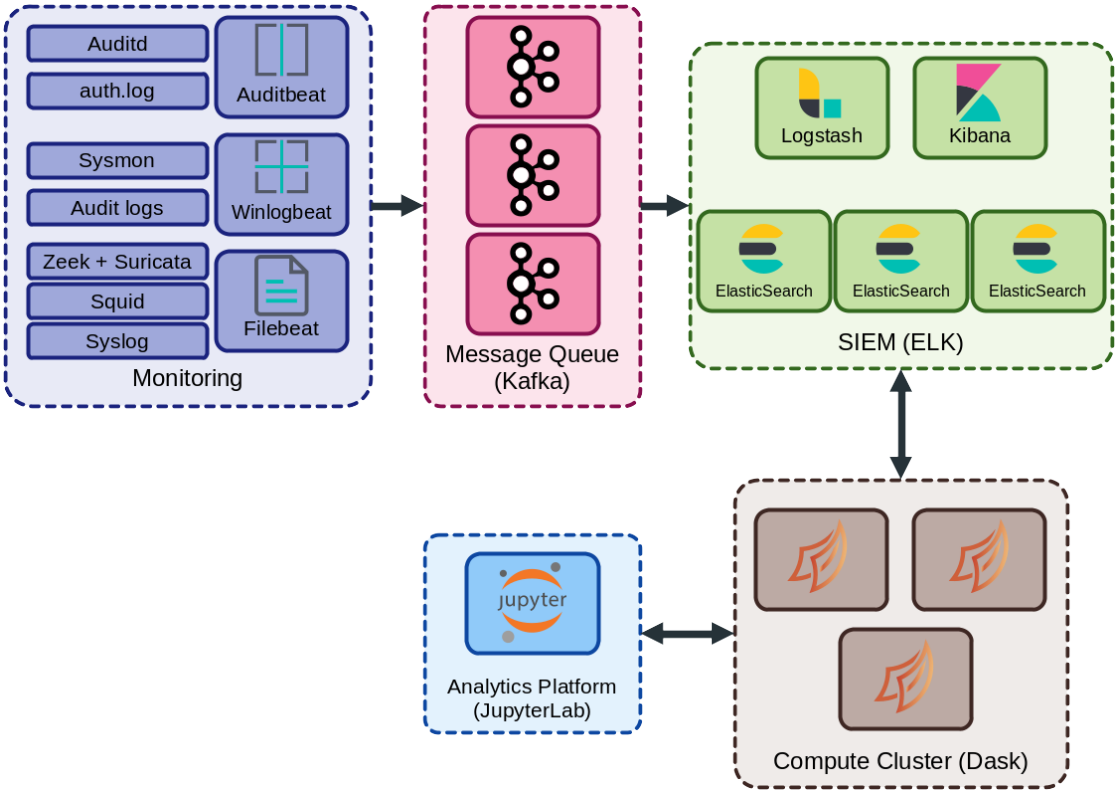


Figure 4.6 – SOC architecture employed to test our method.

4.3.2 Security analytics library

To be able to design multiple analytics based on our approach, we implemented an analytics design toolbox in the form of a Python library. To gain performance on critical operations some modules (e.g., the reduction operator of the fusion method, the tokenizer for string attributes transformation, etc.) are implemented in Cython [21], which allows to implement functions in a language that is similar to Python and is then translated in C and compiled into optimized machine code, leading to 10-100x better performance than pure Python. The toolbox is organised into four submodules described in this section.

20. `https://www.dask.org/`
21. `https://cython.org/`

Core

In this submodule the code for building analytics and managing them is provided. Through a pipeline logic (i.e., handling task order, executing them, etc.), this submodules calls the methods implemented in other submodules in an organized manner.

Extract Transform Load (ETL)

In terms of lines of code, this submodule is the biggest. It implements the code necessary to move data from the SIEM to the various processing task of the analytics, and to normalise, enrich and transform this data for it to be handled by the event fusion and the models. One of the biggest challenges handled by the ETL submodule is to ensure the data is formatted accordingly and to handle formatting errors gracefully, as debugging problems that only arises for a few events in a million takes a sizeable amount of time.

It is mostly implemented in Python using Pandas [22] to represent structured data in the form of data frames (arrays with heterogeneous column types). It also uses the dask library, which also handles Pandas data frames, to interface with the cluster and scale processing function. Some processing functions (e.g., string tokenizer) are implemented in Cython for better performances.

Aggregation

This submodule implements the map and reduce operators of the aggregation function for event fusion described in Sec. 3.3. These are then called by the ETL submodule that is in charge of scaling it thanks to the dask cluster. The code is mostly implemented in Cython and exposed in Python for the rest of the library to use. At the moment, we only applied limited optimizations to reduce the processing time on our dataset to a reasonable amount (a few minutes).

Models

We use the Pytorch [23] library to implement the neural network part described in chapter 2 and 3. It is a popular deep learning framework in the data science

22. `https://pandas.pydata.org/`
23. `https://pytorch.org/`

community and is particularly suited for building dynamic neural networks. It has a lot of built-in standard functions for neural networks (e.g., the categorical embedding we use, Transformer layers, most activation functions, etc.), and we couple it with Pytorch Lightning [24] to handle the training of the network. At the moment, we did not optimize the code to be as efficient as possible (especially on GPUs).

4.3.3 User interface

To verify that we could automate the data science parts based on a few inputs provided by the analysts, we implemented a basic user interface based on configuration files and Jupyter's *ipywidgets* library [25]. This interface is not meant for a security analysts and is rather used to automate the analytics creation to simplify the proposed method assessment with different parameters.

Data source configuration

Data source configuration is provided as a JSON file. Each data source is defined by the Elasticsearch index template in which to look for events, the list of fields to retrieve from events, and the list of attributes that can be analysed by the model. Additionally, it is possible to configure filters to retrieve only specific events. These filters are of type *must*, meaning all the conditions defined must be met for the event to be retrieved, *must_not* which is the inverse of *must* (if any of the conditions matches, the event is not considered), *exists*, to ensure the specified field is defined in the event, and *should*, denoting that at least one of the specified conditions should be met to consider the event. Fig 4.7 provides the configuration we use for the network flow data source (comming from Zeek's *conn.log*).

24. `https://www.pytorchlightning.ai/`
25. `https://github.com/jupyter-widgets/ipywidgets`

```
netflow = {
        "index": "monitoring-network-*",
        "fields": [
                "source.ip", "destination.ip", "network.transport",
                "destination.port", "source.bytes",
                "destination.bytes", "zeek.connection.state",
                "source.packets", "destination.packets",
                "zeek.connection.history", "source.port",
                "zeek.session_id"
        ],
        "attributes": [
                "source.ip", "destination.ip", "network.transport",
                "destination.port", "source.bytes",
                "destination.bytes", "zeek.connection.state",
                "source.packets", "destination.packets",
                "zeek.connection.history", "source.port"
        ],
        "must": [
                # datasource field must be "netflow"
                {"datasource": "netflow"}
        ],
        "must_not": [
                # ignore flows to and from 192.168.211.150
                # (log collector address)
                {"destination.ip": "192.168.211.150"},
                {"source.ip": "192.168.211.150"}
        ]
}
```

Figure 4.7 – Example data source configuration for retrieving network flow events from zeek *conn.log*.

In the future, for event data sources, we consider using the sigma rule format [26]. Indeed, they already provide abstraction to retrieve events from various sources

26. https://github.com/SigmaHQ/sigma

(called *logsources* in sigma rules), and from multiple SIEM. This would also simplify mixing rule-based detection and models in analytics. However, in our case, it would have required more integration work to implement, as it would have to modify the *logsources* mapping to take our enrichments to the ECS data model into account, and would not have provided much benefits as we only have one SIEM to interface with (Elastic), and we do not use rule-based detection.

Enrichment pipeline

This interface is meant to configure additional enrichment and normalisation tasks to the data sources. It allows renaming fields (if the chosen data model does not already normalise them), adding new fields with predefined values, applying extra enrichments operations, and provides a preview of the modifications on example data. The enrichment operations are applied in order of definition and can be of three different types:

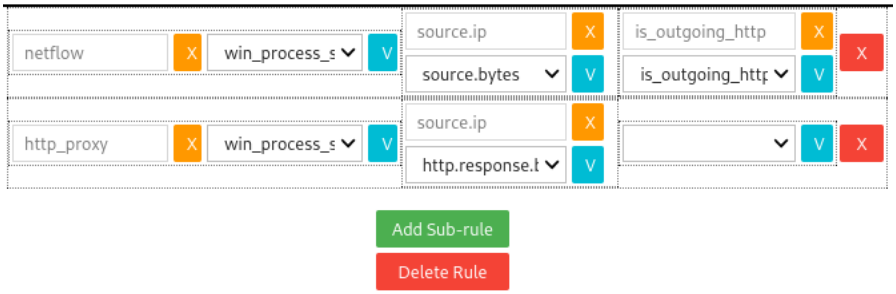
— **Map**: which creates (or modifies) an attribute based on an existing one (e.g., convert a specific string attribute in upper case);
— **Apply**: similar to *map*, but takes as input all the attributes in the event (not just a specific one);
— **Dataframe**: it takes the whole dataframe as input and returns a transformed one. This is useful for applying aggregation functions on the inputs (e.g., aggregating powershell script-block events to reconstruct the complete script).

Fig. 4.8 shows the interface for extra enrichment operations on the network flow data source. We use a similar interface to chose which (enriched and normalised) attributes will be processed by the model, and their type.

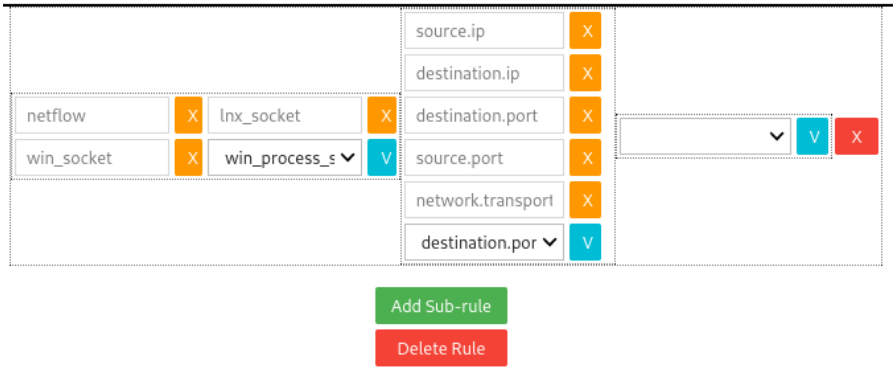
Figure 4.8 – Screenshot of the enrichment interface with a focus on network connection events.

Fusion rules

The rule creation interface (Fig 4.9) is designed to represent and edit the rules (described in Sec 3.3.2) in a more efficient way. For a given sub-rule, it automatically restricts the fields to those that are available in all the data sources that are listed in the sub-rule.



(a) Rule that fuses proxy events with network connections based on the source IP address only if they are outgoing HTTP connections.



(b) Rule that fuses network connections seen by network probe and endpoint monitoring tools (e.g., opened sockets).

Figure 4.9 – Screenshots of the interface to create fusion rules.

To give an overview to the reader of the possible links between data sources, we use a graph representation (Fig. 4.10) where each node is a data source and each edge indicates that at least one rule can be used to regroup events from the connected data sources. As expected, endpoint logs for each OS (Linux and Windows) form highly connected clusters (because the process identifier is found in all endpoint events), and event connected clusters are linked to each other via the events that record network connections from the various point of views (i.e., network probes, endpoint monitoring and application logs).

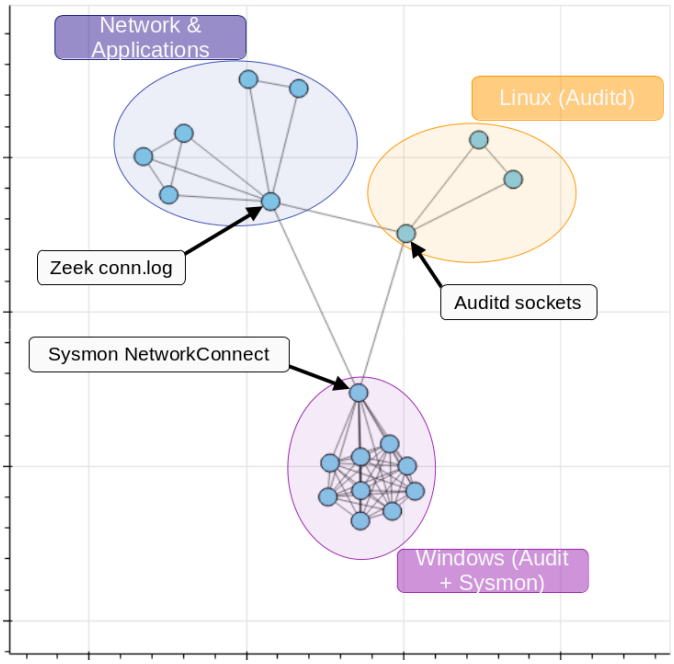


Figure 4.10 – Representation of the fusion rule set as a graph.

False positive annotation

The false positive annotation interface (Fig 4.11) is designed specifically to annotate the anomalies detected with out approach, and empirically assess the clustering capabilities. As such, it is not suitable for security analysts in a production environment. Doing so would require improved investigation capabilities (e.g., visualisations, dashboards, filtering, etc.).

Individual anomalies are grouped in clusters, with one tab for each cluster and a tab for anomalies that cannot be clustered (thanks to density based methods). Within each tab, meta event sets are ordered according to their time-stamp. For each set, the human expert can view a table containing each attribute of the event as well as the score associated to these attributes (and the global event score). To assess the clustering performance, the human expert is required to annotate each set individually as false positive or true positive. On our dataset, we did not find a case were a cluster contains both false positives and true positives, and sets belonging to the same cluster mostly contain the same information and are therefore faster to annotate in a batch.

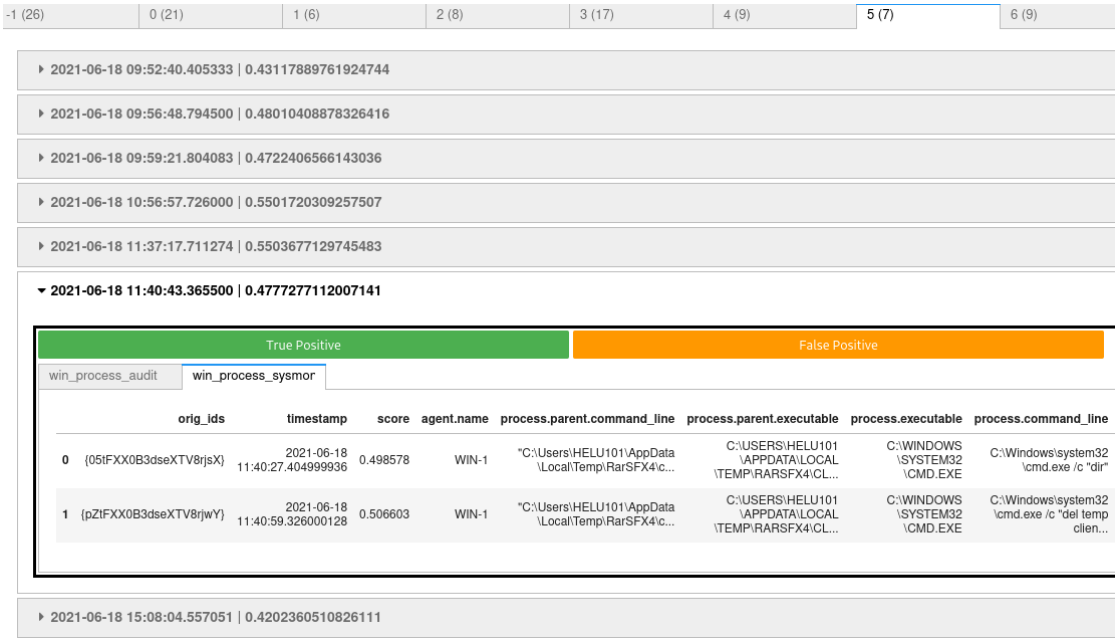


Figure 4.11 – Screenshot of the alert annotation interface. Here, the implanted malware executes a command retrieved from its command & control server.

In a production environment, some anomalies cannot be considered as either false positives (that should be integrated into the model) or true positives (which are escalated as security incidents). For instance some unwanted behaviours (e.g., an administrator not respecting the security policy) may be considered anomalous but do not require immediate case handling by the incident response team and are therefore not to be considered as true positives. The annotation data base could store these alerts to avoid presenting them repeatedly to analysts and handle them separately (e.g., identify why the administrator need to bypass the protocol, and how we can adapt it accordingly). However, in our dataset, we did not simulate this kind of behaviour, and thus, we did not take it into account for the annotation interface.

4.4 Experimental results

In this section, we used the dataset we collected to assess the anomaly detection performances, the clustering quality and the relevance of the annotation loop. These results show the pertinence of our approach.

4.4.1 Defining the assessment strategy

Commonly used quantitative metrics (e.g., False/True Positives Rates, F1 score, etc.) can provide valuable information regarding the performance of a detection method, but they only give a partial view of the usefulness of the detection system. For instance, missing 50 out of 100 connection attempts to a service in a brute force attack would not prevent analysts from finding out the source and the target of the attack, while increasing the False Negative Rate (FNR). However, missing a single event that characterizes an attack would not have a major impact on the FNR, while greatly reducing the quality of the detection.

Defining quantitative metrics to evaluate data science methods for cyber security is a challenge [92], and thus, we propose an alternative definition of a False Negative that is more inline with analysts practices. In fact, we know every actions that are performed by the attacker, but annotating individually the thousands of events that can be attributed to these actions would be time consuming and error prone. Instead, we start from a high threshold above which we consider a set of meta-events as an anomaly, and we lower it until we can find anomalies characterising every step of the attack (i.e., what is the machine impacted, the technique employed, etc.). Any set below this threshold that is a consequence of malicious activity will not be considered as a False Negative, as it will not add useful information to characterise the attack. Empirically, we find that a threshold of 0.4 is a good choice for the three scenario we have. However, this threshold is not optimal as multiple legitimate actions can have a score higher than it. Therefore, we gradually increase this threshold and we compute the precision (proportion of True Positive in all the anomalies), recall (proportion of the attack accurately detected) and F1 score (harmonic mean of precision and recall) to quantify the performance. For all these metrics, a value close to 0 implies a useless anomaly detection while a score of 1 is synonym of a good one. We consider the optimal threshold to be the one that maximizes the F1 score.

We mitigate the impact of a lucky (or unlucky) initialisation of the model by training 10 models with different initialisation of the parameters on the first two days of normal data. These data are randomly splitted for each model, with 90% used for training the model and 10% to control that the model is not over-fitting the training data (often called validation set). We test the performance of each of these models on the first scenario. After that, we keep only one model and use it

as the base for incremental learning and evaluating the performance on the other two scenarios.

4.4.2 Results

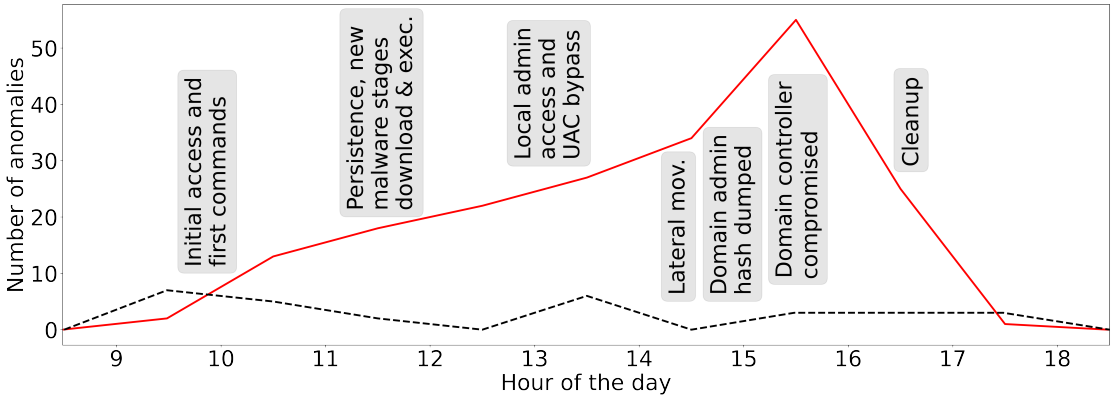


Figure 4.12 – Aggregated number of meta-event sets with a score above 0.5 per hour for scenario 1.

For each of the 10 randomly initialized models, we record the best F1 score on the first attack scenario. The average value of this score is 0.88 with the lowest value at 0.85 and highest at 0.91. The figure 4.12 has been generated using the best model (F1: 0.91, recall: 0.93 and precision: 0.88). The dashed black line shows the number of anomalies per hour detected during a single day without attack. On the opposite, red line shows number of anomalies per hour produced when an attack occurred. As expected, we can see that the number of anomalies increases as the attack progresses. The maximum number of anomalies is reached when most of the machines are compromised, and especially the domain controller. The cleanup phase is accompanied by a decreasing number of anomalies as machines progressively stop exhibiting attack behaviours. This scenario shows that our system can be used as intrusion detection system as it detects attack-related events and because it does not report more than fifty anomalies per hour from an initial number of events of approximately 20000 per hour.

For the second scenario, we use the false positives from the first scenario and the fourth day of the collected dataset as the input to incrementally update the anomaly detection model. We reach a F1 score of 0.97 with a recall of 0.99 and a precision of 0.95. Similarly, for the third scenario, we reach an F1 score of 0.92 with

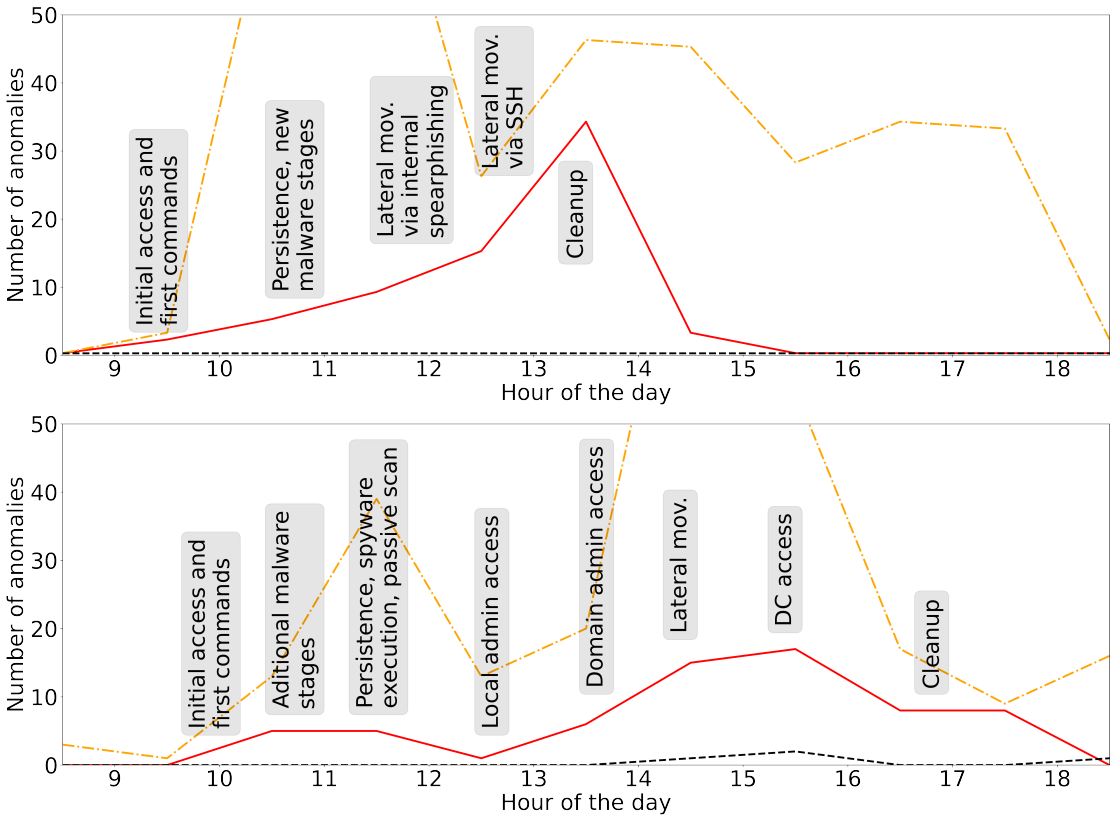


Figure 4.13 – Aggregated number of meta-event sets with a score above 0.5 per hour, for scenario 2 (top) and 3 (bottom)

a precision of 0.86 and a recall of 0.98. Without retraining (i.e., using the same model used for scenario 1), the best F1 score drops below 0.3 for both scenarios. To highlight the effectiveness of the retraining, in Figure 4.13, we add the plot (orange dot-dashed line) of the same normal day than the black dashed plot, but this time analysed by the outdated model.

Similarly to the first scenario, there is an increasing number of anomalies reported during the attack process of the scenario 2. We see lower numbers of anomalies, which is expected considering the attack a bit less noisy. As the third scenario is designed to be as discreet as possible, we see far less anomalies. Nevertheless, meta-events raised by our system are sufficient for an analyst to recognize the attack pattern. The orange line shows the necessity of the incremental learning process for our anomaly based detection system.

The latent representation provided by the model enables the use of standard clustering algorithms to cluster meta-event groups. Most of the time, new be-

haviours reappear regularly. Thus, the manifestation of these new behaviours in the log should form clusters of similar activity. In our case, we use the DBSCAN algorithm [39] to perform this clustering and we find that the discovered cluster permits to reduce the annotation time. Fig. 4.14 shows two anomalous meta-event groups that are found to be similar. The first one corresponds to the events generated when the attacker enumerated local users and groups and the second to the enumeration of domain users and groups. Preliminary results are encouraging, but a more in depth evaluation of the clustering performance is still required.

win_process_audit | win_process_sysmor

timestamp	score	agent.name	process.parent.command_line	process.parent.executable	process.executable	process.command_line
2021-06-23 08:22:00.956000000	0.505922	WIN-2	cmd.exe	C:\WINDOWS\SYSWOW64 \CMD.EXE	C:\WINDOWS \SYSWOW64 \WHOAMI.EXE	whoami /all
2021-06-23 08:22:09.728999936	0.506463	WIN-2	cmd.exe	C:\WINDOWS\SYSWOW64 \CMD.EXE	C:\WINDOWS \SYSWOW64 \NET.EXE	net localgroup
2021-06-23 08:22:09.763000064	0.520170	WIN-2	net localgroup	C:\WINDOWS\SYSWOW64 \NET.EXE	C:\WINDOWS \SYSWOW64 \NET1.EXE	C:\Windows \system32\net1 localgroup

win_process_audit | win_process_sysmor

timestamp	score	agent.name	process.parent.command_line	process.parent.executable	process.executable	process.command_line
2021-06-23 11:39:09.503000064	0.506828	WIN-2	cmd.exe	C:\WINDOWS\SYSWOW64 \CMD.EXE	C:\WINDOWS \SYSWOW64 \NET.EXE	net user /domain
2021-06-23 11:39:09.548999936	0.520004	WIN-2	net user /domain	C:\WINDOWS\SYSWOW64 \NET.EXE	C:\WINDOWS \SYSWOW64 \NET1.EXE	C:\Windows \system32\net1 user /domain
2021-06-23 11:39:16.097999872	0.507229	WIN-2	cmd.exe	C:\WINDOWS\SYSWOW64 \CMD.EXE	C:\WINDOWS \SYSWOW64 \NET.EXE	net group /domain
2021-06-23 11:39:16.116000000	0.524452	WIN-2	net group /domain	C:\WINDOWS\SYSWOW64 \NET.EXE	C:\WINDOWS \SYSWOW64 \NET1.EXE	C:\Windows \system32\net1 group /domain

Figure 4.14 – These two anomalous meta-event groups belong to the same cluster in the latent space.

4.5 Discussion

The experiment we performed shows that, with our approach, when designing security analytics that include machine learning models, it is possible to automate most of the data engineering and abstract the parts that require data science knowledge, while maintaining good detection capabilities. It also shows that the use of event fusion and clustering allows the implementation of an active learning loop, where analysts perform anomaly triage to provide feedback to the model,

while not overloading them with too much information that is difficult to make sense of.

Our method was integrated in a lab environment that relies on commonly used monitoring tools (from event generation to storage and analysis in the SIEM). Therefore, it could be exploited in a production environment without applying massive modification to the monitoring strategy and SOC architecture. Nevertheless, our method is designed to lower the trade-off between visibility and burden to the analysts. Therefore, to fully exploit the enhanced visibility that can be provided, it might be necessary to increase the scalability of the SOC architecture (e.g., using Big Data-inspired architectures and tools), to ensure it can sustain the increased volume of data.

While these results are encouraging, we envision four possible improvements paths that we describe in this section.

4.5.1 Visualisation

In this thesis, visualisation was not the main focus. However, we believe that it is an essential part for any method that wishes to assist human experts when analysing data. These visualisation should serve two purposes, namely, helping analysts navigate and pivot around data, as well as summarizing it to provide a better global understanding of the situation, while still allowing analysts to investigate low level events to get a finer understanding.

With our experiment using graph-based visualisation and analysis methods (Sec. 3.2), we concluded that abstracting data to reduce the number of nodes was required for these methods to provide tangible effects. For fine-grain analysis, e.g., investigating a short period of time (a few minutes), with a reduced set of machines (e.g., 1-5), we could use the event fusion method we proposed to effectively reduce the number of nodes in a Directed Acyclic Graph that approximates causal dependencies, the precision of the approximation depending on the visibility provided by the monitoring strategy. However, to provide the higher abstraction level that is required to detect long-lived, system-wide attacks, additional research should be done.

Graph representation is however not the only possibility and should likely be combined with other visualisations. In fact, most SIEMs expose to analysts the capabilities to design specific dashboards. The anomaly scores and cluster identifiers

are additional data that can be visualised with and enrich these dashboards.

4.5.2 Additional datasets

While the implementation of our approach has been done independently from the dataset to ensure we could configure it for various monitoring strategies, we only tested it on one dataset. While we focused on introducing realistic activity in the dataset, its realism is bound by the realism of the simulation. Moreover, it is not possible to simulate all the possible behaviours of users and therefore, any dataset can only be representative of the system it has been collected on (event for production environment). Finally, the size of the lab environment we used (approximately 20 machines) is small in comparison to some production environments (from a few hundreds to a few hundreds of thousands of machines).

We designed an approach to simulate user interaction with lab environments. By enriching the list of actions it can perform and improving the scheduling capabilities, we could generate multiple larger datasets from various monitored environments. Ideally, we would like to take feedback directly from security analysts, and replicate the behaviours that troubles them in day to day operations, in order to be able to design and test methods that fits their needs. In addition to that, generated datasets are free from personal and sensitive informations, and are therefore shareable with third parties.

4.5.3 Production use cases

The results show that the proposed system is able to monitor the attackers in detail and offer a reduced set of anomalies that needs to be investigated by the analyst. While our system has solely been tested in a simulated environment, multiple real-life applications have been envisioned. First, it can be used to analyze events coming from *detection labs*, i.e., controlled systems specifically designed to analyse attackers, where user activity is simulated to fool attackers. Second, considering large SOCs can receive 10M to 20M of events per hour (compared to ≈20000 for our dataset), using the proposed method would not alleviate alert fatigue. However, it can be integrated in current SOC practices which consist in analysing fine-grained events (e.g., Sysmon, netflow, etc.) only for reduced perimeters. Specifically, for known suspicious behaviours, analysts often design scripts meant to automati-

cally retrieve the context surrounding the detected behaviour. Incorporating our method in these scripts could allow faster handling of these investigations, which can reduce alert fatigue, and/or permit the use of wider spectrum detection rules (which can catch more advanced attacks). It is also possible to restrict our method to specific assets of the monitored environments (e.g., domain controllers, servers, etc.).

We consider that machine learning and data science is only (and is likely to still be in the foreseeable future) a tool to support human experts in their analysis by handling large amounts of data. Therefore, in production environments, security analysts should be implicated deeply in the design of security analytics.

The interface we implemented to assess our approach is not meant for usage by security analysts, but rather to verify that abstracting data science parts is still possible with reasonable performances. Thus, the interface should be revised to facilitate interactions with analysts. Furthermore, in addition to the models, all the code that enables a security analytics should be maintained and evolve with the needs of the analysts. In most IT software design, this maintenance problematic is handled by DevOps practices, which focuses on reducing the time between requirements expression, solution implementation and deployment. However, security analysts are neither developers nor data scientists, and therefore these practices should be adapted to suit there skills. We think additional studies are required to design the right interfaces for analysts.

4.5.4 Refinements

Even though these results are promising, some limitations can be highlighted. In fact, it still relies on parameters that needs to be defined by an expert. More precisely, an inadequate choice of attributes for the data sources, i.e., missing important attributes or adding useless ones, can lead to unsatisfactory results. For instance, with the current scoring system, a data source with more than 10 attributes will rarely cause alerts because most of the attributes would seem normal and dilute the score of anomalous attributes. Besides, the chosen time-window can have an impact on the quality of the fusion step. Indeed, during testing, we realised that a time-window below 120 seconds often fail can lead to a significant amount of missed fusion opportunity between data sources that have different ways of setting the timestamp (e.g., a network probe often emits an event at the end of a connec-

tion, while an endpoint monitoring would do so at its beginning). Symmetrically, we found that a time-window above 300 seconds would lead spurious correlations between meta-events.

Moreover, facilitating the use of the methods by security analysts (using visualisation and improving the analytics design process), as well as testing the approach with diverse datasets, will likely highlight more limitations. While we cannot predict what these limitations will be, some will likely need to be handled by data scientists as well as security analysts. Therefore, we consider it important that any implementation of the approach considers interaction with both data scientists and security analysts.

Conclusion

The purpose of this research is to lower the burden on security analysts by allowing them to leverage data science methods to automate more of their investigation procedures. Indeed, many aspects of security alerts investigation require analysts to skim through large amount of data. Often, legitimate behaviours repeatedly triggers false alarms, which can cause alert fatigue, as analysts are overloaded to the point they mechanically focus only on superficial elements, and potentially miss subtle variations that are the consequence of attackers behaviours.

To combat this phenomenon, we focus on anomaly detection, to highlight unusual behaviours, and clustering, to regroup similar alerts and avoid repeating the same alerts again and again. Considering that attackers behaviours are likely to be anomalous (i.e., different than the legitimate behaviours of the users), in addition to alert fatigue reduction, we explore the application of these methods to novel attack detection.

When designing our approach, we took into account the known attacks against anomaly detection models. For mimicry attacks (i.e., making an attack look legitimate), the risks of successful attack is lowered by the facts that the model is specific to each system (i.e., attackers can only perform more difficult black box attacks), and that multiple redundant sources of events can be analysed (i.e., an attack would need to look normal from all perspectives). For the frog boiling attack (i.e., poisoning of continuously trained anomaly detection models), the active learning setup forces attackers to also bypass human vigilance.

Key contributions

Designing analytics for heterogeneous security events

Security analysts, who have the knowledge and experience to design security analytics, often lack the data science expertise necessary to configure machine learning-based approaches. Also, machine learning models are often viewed as

black-boxes by security analysts, and consequently, they are reluctant in adopting approaches that rely on machine learning. Therefore, to improve analysts' understanding of the results, while benefiting from their expertise, the chosen approach puts them at the centre of the analytics design process. This process is adapted from one that is commonly used by threat hunters to design security analytics that help look for adversaries Tactics, Techniques and Procedures (TTP) [27]. It allows analysts to incorporate machine learning models for anomaly detection and clustering within these analytics.

A key focus of the proposed approach is the abstraction and automation of the configuration step of the machine learning models and the associated pre-processing functions (which are required to ensure the model can process the data). To that end, we rely on deep learning, namely neural network auto-encoders, to minimize the human involvement in the feature transformation step (in deep neural networks, the networks automatically adapt the transformation functions parameters). The auto-encoder is used to provide an anomaly score to each processed event and the attributes that compose it, as well as a vector (called latent representation) that is used to cluster similar events. The network structure is automatically constructed using generic building blocks for the most common attribute types that can be found in security events (i.e., categorical, numerical and string variables).

With the proposed approach, analysts can design analytics that combine traditional rule-based detection and scripts with anomaly detection and clustering models trained on historic monitoring data. A typical use case is to filter out false positives and regroup similar alerts generated by a detection rule. This way, analysts know what the analytics should detect (i.e., events that trigger the rules), and can rely on familiar detection capabilities (i.e., rules and scripts) to understand the results, while potentially lowering the number of false positives (as the frequent ones will be considered normal), and reducing the number of alerts (by clustering similar ones). For novel attack detection, based on previous experiences and known investigation procedures, analysts can configure models to analyse event sources which can provide visibility on attackers' activities (e.g., process execution, internal network communication to detect lateral movement, etc.). In this case, models can be used by threat hunters to prioritise the investigation of the most suspicious events.

Adapt to concept-drift using active learning

In an IT system, user behaviour is bound to evolve. As such, new legitimate activity can trigger alerts, and old activity can cease to manifest inside the monitoring data. In data science, this is called concept-drift, and is usually accounted for by training model continuously. However, an adversary can take advantage of continuous learning to poison models by progressively distilling seemingly benign traces of their attack until the whole attack is considered normal by the model (also called a frog-boiling attack). To prevent this, data that is reinserted inside the models should be carefully chosen (i.e., only false positives). We exploit the alert qualification that is already performed by analysts to select the data that should be used to retrain the models.

In the case of novel attack detection, as the analytics are used to analyse security events that are, for the vast majority (>99%), caused by normal user activity, the number of false alarms can be high. While clustering can help reduce the volume of information, for large scale systems, it might not be enough to reach an acceptable number of alerts. To further reduce this number and provide as much context as possible to accelerate the investigation process, we use event fusion to aggregate all the events that are caused by the same actions into sets. We propose a scalable algorithm for event fusion that the analysts can configure by providing the pivot variables (i.e., attributes that are common from one source of event to another) that they commonly use to navigate through security events when investigating alerts. We also propose to dynamically adapt the structure of the auto-encoder, based on the composition of the fused event set. This dynamic auto-encoder is also able to detect based on their context (e.g., events that might be independently normal, but should not come together).

Combined with clustering, this approach reduces the number of alerts by two orders of magnitude. As all the security events that described the actions that triggered the alert are presented to analysts (in addition to the score), they do not need to navigate through security events to reconstruct the context, and thus, the investigation time is reduced. This makes the approach suitable for an active learning setup, where the alerts qualifications resulting from the investigation are used as feedback for the model (i.e., false positives are incorporated inside the model).

Building realistic datasets

A major problem encountered by researchers when designing novel methods for security monitoring is the lack of representative public datasets. In fact, due to the sensitive nature of the information contained in monitoring data, practitioners are usually legally not allowed (both by contracts and regulations) to share production data. For datasets that are generated in lab environments, their representativeness is often limited by the quality of the activity that is generated. In particular, legitimate user activity is the biggest cause of false positives in production environments, but is usually limited in public datasets (e.g., only network activity, visible agents that taint the logs, etc.).

To overcome this challenge, we propose an approach for automated user activity generation in lab environment. It relies on an agent that interfaces with the machines through the mouse, keyboard and screen, and can therefore instrument machines from the outside (and thus, does not taint the logs). The agent is designed in multiple abstraction layers in order for operators to quickly design life generation scenarios and adapt them for the specificities of their environment (e.g., different software versions, heterogeneous OSes, etc.). The agents are coordinated using a scheduler that ensures agents performed the right action at the right time on the instrumented machines (e.g., a user cannot answer an email before receiving it).

We used this method to generate a datasets containing seven days of user activity and three complete attack scenarios. This dataset has been used to assess the methods we proposed in this thesis. It will be made available in order for researcher to reproduce the results and experiment with their own methods.

Perspectives

At the end of this research, we identify four major axis that could be further investigated. Specifically, the approach we propose consists in building blocks that are required but not sufficient to create a security analytics platform. As such, it lacks features that are necessary to security analysts (e.g., alert visualisation). Also, it has only been tested on a single environment, and three attack scenarios. While attacks targeting the models have been taken into account in the design, further evaluation of robustness against these attacks and potentially novel attacks

is necessary. Finally, further testing could help identify performance issues, both in computational and detection terms, which might require adaptations to the models.

Interaction with the operator

This thesis describes methods to incorporate machine learning models inside security analytics and maintain them, without requiring data science knowledge from the analysts. However, designing analytics in itself requires specific interactions with the human expert (i.e., a development environment), and therefore, the design of the user interfaces can also have a large impact on the ease of use by analysts.

In addition to that, we focused on producing the results, but we did not address the investigation and exploitation of these results by the analysts. To this end, research in data visualisation can be helpful. For instance, with the ability to regroup events into sets describing actions, the abstraction level could be enough to render graph-based visualisation efficient (i.e., number of node sufficiently low).

Diversify assessment use cases

At the end of this research, we assessed the performance of our approach on a single dataset containing three different multi-step attacks. While we focused on building a relatively realistic dataset, it cannot cover all the cases from all the possible IT systems. With the life generation capabilities, it becomes possible to dynamically create datasets. By taking feedback from security analysts, we could simulate the corner cases and pain points they identified to generate these datasets. Once sufficient testing on varied lab environments have been performed, the approach could be tested directly by security analysts, on production environments.

Evaluate robustness against targeted attacks

While we took into account both mimicry and frog-boiling attacks when designing our approach, we did not evaluate its robustness against any implementation of these attacks. Implementing these attacks in a lab environment under realistic constraints (e.g., attackers cannot modify the events in the SOC, they do not con-

trol the tools that generate security events, etc.) is not trivial, and would require dedicate research.

We evaluated our approach with an initial training performed on a dataset with only legitimate activity. In theory, having traces of attacks inside the training data would only prevent detection of these attacks. Evaluating and reducing the impact of tainted training data could also be investigated (e.g., exploring the initial training data to ensure it is clean).

Also, we did not explore novel attack vectors that could be exploited to evade the presented methods. For example, an attacker could try to saturate the computational resources to increase the time to analyse their activity, overload analysts with multiple very anomalous attacks while performing a more stealthy one, etc.

Refine approach performances

In this research, we did not optimize the approach for computational efficiency. However, deep learning comes with computational requirements that can be non-negligible, and it requires a sizeable amount of data, especially at training time. Further evaluation is required to determine whether or not this would cause a problem in an operational context. Also, approaches to lower the cost of the initial training of the models could be investigated (e.g., Transfer Learning [110], sharing initial training costs through Federated Learning [67], etc.).

All the previous axes could also lead to the identification of unforeseen limitations that might require modification of the approach.

PUBLICATIONS

[30] Alexandre Dey, Eric Totel, and Benjamin Costé, « DAEMON: Dynamic Auto-encoders for Contextualised Anomaly Detection Applied to Security MONitoring », *in*: *IFIP International Conference on ICT Systems Security and Privacy Protection*, Springer, 2022, pp. 53–69.

[31] Alexandre Dey, Eric Totel, and Sylvain Navers, « Heterogeneous security events prioritization using auto-encoders », *in*: *International Conference on Risks and Security of Internet and Systems*, Springer, 2020, pp. 164–180.

[32] Alexandre Dey et al., « Adversarial vs behavioural-based defensive AI with continual and joint learning: automated evaluation of robustness to deception, poisoning and concept drift », *in*: (2019).

[33] Alexandre Dey et al., « Realistic simulation of users for IT systems in cyber ranges », *in*: *arXiv preprint arXiv:2111.11785* (2021).

[40] Jean-Philippe Fauvelle, Alexandre Dey, and Sylvain Navers, « Protection of an information system by artificial intelligence: a three-phase approach based on behaviour analysis to detect a hostile scenario », *in*: *arXiv preprint arXiv:1812.00622* (2018).

BIBLIOGRAPHY

[1] Amir Afianian et al., « Malware Dynamic Analysis Evasion Techniques: A Survey », *in*: *arXiv:1811.01190 [cs]* (Nov. 2018), arXiv: 1811.01190, URL: http://arxiv.org/abs/1811.01190 (visited on 10/10/2021).

[2] Leman Akoglu et al., « Fast and reliable anomaly detection in categorical data », *in*: *Proceedings of the 21st ACM international conference on Information and knowledge management*, ACM, 2012, pp. 415–424.

[3] Idan Amit et al., « Machine learning in cyber-security-problems, challenges and data sets », *in*: *arXiv preprint arXiv:1812.07858* (2018).

[4] Amossys, *desker*, https://gitlab.com/d3sker/desker, 2021.

[5] Md Anjum, Shahrear Iqbal, Benoit Hamelin, et al., « ANUBIS: A Provenance Graph-Based Framework for Advanced Persistent Threat Detection », *in*: *arXiv preprint arXiv:2112.11032* (2021).

[6] Mihael Ankerst et al., « OPTICS: Ordering points to identify the clustering structure », *in*: *ACM Sigmod record* 28.*2* (1999), pp. 49–60.

[7] Eirini Anthi et al., « Adversarial attacks on machine learning cybersecurity defences in industrial control systems », *in*: *Journal of Information Security and Applications* 58 (2021), p. 102717.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, 2014, DOI: 10.48550/ARXIV.1409.0473, URL: https://arxiv.org/abs/1409.0473.

[9] Marco Barreno et al., « Can machine learning be secure? », *in*: *Proceedings of the 2006 ACM Symposium on Information, computer and communications security - ASIACCS 06*, ACM Press, 2006, p. 16, ISBN: 978-1-59593-272-3, DOI: 10.1145/1128817.1128824, URL: http://portal.acm.org/citation.cfm?doid=1128817.1128824.

[10] Adam Bates et al., « Trustworthy whole-system provenance for the linux kernel », *in*: *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 319–334.

[11] Richard E Bellman and Stuart E Dreyfus, *Applied dynamic programming*, Princeton university press, 2015.

[12] Christophe Bertero et al., « Experience report: Log mining using natural language processing and application to anomaly detection », *in*: *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, 2017, pp. 351–360.

[13] David Bianco, *The Pyramid of Pain*, Mar. 2013, URL: `http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html`.

[14] Battista Biggio, Blaine Nelson, and Pavel Laskov, « Poisoning attacks against support vector machines », *in*: *arXiv preprint arXiv:1206.6389* (2012).

[15] Markus M Breunig et al., « LOF: identifying density-based local outliers », *in*: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[16] Guillaume Brogi and Valérie Viet Triem Tong, « Terminaptor: Highlighting advanced persistent threats through information flow tracking », *in*: *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, IEEE, 2016, pp. 1–5.

[17] Alexei Bulazel and Bülent Yener, « A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion: PC, Mobile, and Web », *in*: *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*, ROOTS, Vienna, Austria: Association for Computing Machinery, Nov. 2017, pp. 1–21, ISBN: 9781450353212, DOI: `10.1145/3150376.3150378`, URL: `https://doi.org/10.1145/3150376.3150378` (visited on 10/10/2021).

[18] Eric Chan-Tin et al., « The Frog-Boiling Attack: Limitations of Anomaly Detection for Secure Network Coordinate Systems », *in*: *Security and Privacy in Communication Networks*, ed. by Yan Chen, Tassos D. Dimitriou, and JianyingEditors Zhou, vol. 19, Springer Berlin Heidelberg, 2009, pp. 448–458, ISBN: 978-3-642-05283-5, DOI: `10.1007/978-3-642-05284-2_26`, URL: `http://link.springer.com/10.1007/978-3-642-05284-2_26`.

[19] Xinyun Chen et al., « Targeted backdoor attacks on deep learning systems using data poisoning », *in*: *arXiv preprint arXiv:1712.05526* (2017).

[20] Kyunghyun Cho, Aaron Courville, and Yoshua Bengio, « Describing Multimedia Content Using Attention-Based Encoder-Decoder Networks », *in*: *IEEE Transactions on Multimedia* 17.*11* (Nov. 2015), pp. 1875–1886, DOI: 10.1109/tmm.2015.2477044, URL: https://doi.org/10.1109%2Ftmm.2015.2477044.

[21] Kyunghyun Cho et al., « Learning phrase representations using RNN encoder-decoder for statistical machine translation », *in*: *arXiv preprint arXiv:1406.1078* (2014).

[22] Yann Collet, *xxHash: Extremely fast hash algorithm*, 2016.

[23] Damien Crémilleux, « Visualization for information system security monitoring », PhD thesis, CentraleSupélec, 2019.

[24] *CVE*, URL: https://www.cve.org/.

[25] Laurens D'hooge et al., « In-depth comparative evaluation of supervised machine learning approaches for detection of cybersecurity threats », *in*: *4th International Conference on Internet of Things, Big Data and Security (IoTBDS)*, 2019, pp. 125–136.

[26] Prithviraj Dasgupta and Joseph Collins, « A survey of game theoretic approaches for adversarial machine learning in cybersecurity tasks », *in*: *AI Magazine* 40.*2* (2019), pp. 31–43.

[27] Roman Daszczyszak et al., *TTP-Based Hunting*, tech. rep., MITRE CORP MCLEAN VA MCLEAN, 2019.

[28] Biplob Debnath et al., « Loglens: A real-time log analysis system », *in*: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2018, pp. 1052–1062.

[29] Arthur P Dempster, Nan M Laird, and Donald B Rubin, « Maximum likelihood from incomplete data via the EM algorithm », *in*: *Journal of the Royal Statistical Society: Series B (Methodological)* 39 (1977), pp. 1–22.

[30] Alexandre Dey, Eric Totel, and Benjamin Costé, « DAEMON: Dynamic Auto-encoders for Contextualised Anomaly Detection Applied to Security MONitoring », *in*: *IFIP International Conference on ICT Systems Security and Privacy Protection*, Springer, 2022, pp. 53–69.

[31] Alexandre Dey, Eric Totel, and Sylvain Navers, « Heterogeneous security events prioritization using auto-encoders », *in*: *International Conference on Risks and Security of Internet and Systems*, Springer, 2020, pp. 164–180.

[32] Alexandre Dey et al., « Adversarial vs behavioural-based defensive AI with continual and joint learning: automated evaluation of robustness to deception, poisoning and concept drift », *in*: (2019).

[33] Alexandre Dey et al., « Realistic simulation of users for IT systems in cyber ranges », *in*: *arXiv preprint arXiv:2111.11785* (2021).

[34] Nat Dilokthanakul et al., « Deep unsupervised clustering with gaussian mixture variational autoencoders », *in*: *arXiv preprint arXiv:1611.02648* (2016).

[35] Zhiguo Ding and Minrui Fei, « An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window », *in*: *IFAC Proceedings Volumes* 46 (2013), pp. 12–17.

[36] Min Du et al., « Deeplog: Anomaly detection and diagnosis from system logs through deep learning », *in*: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2017, pp. 1285–1298.

[37] Elastic, *Elastic Common Schema*, `https://github.com/elastic/ecs`, 25-03-2021.

[38] Meng Joo Er et al., « Attention pooling-based convolutional neural network for sentence modelling », *in*: *Information Sciences* 373 (2016), pp. 388–403, ISSN: 0020-0255, DOI: `https://doi.org/10.1016/j.ins.2016.08.084`, URL: `https://www.sciencedirect.com/science/article/pii/S0020025516306673`.

[39] Martin Ester et al., « A density-based algorithm for discovering clusters in large spatial databases with noise. », *in*: *Kdd*, vol. 96, 1996, pp. 226–231.

[40] Jean-Philippe Fauvelle, Alexandre Dey, and Sylvain Navers, « Protection of an information system by artificial intelligence: a three-phase approach based on behaviour analysis to detect a hostile scenario », *in*: *arXiv preprint arXiv:1812.00622* (2018).

[41] Pengbin Feng et al., « UBER: Combating Sandbox Evasion via User Behavior Emulators », en, *in*: *Information and Communications Security*, ed. by Jianying Zhou et al., vol. 11999, Cham: Springer International Publishing, 2020, pp. 34–50, ISBN: 9783030415785 9783030415792, DOI: 10.1007/978-3-030-41579-2_3, (visited on 10/10/2021).

[42] Frédéric Guihéry, Alban Siffer, and Joseph Paillard, « BEEZH: une plateforme de détonation réaliste pour l'analyse des modes opératoires d'attaquants », *in*: (2020).

[43] Xueyuan Han et al., « Unicorn: Runtime provenance-based detector for advanced persistent threats », *in*: *arXiv preprint arXiv:2001.01525* (2020).

[44] Wajih Ul Hassan, Adam Bates, and Daniel Marino, « Tactical provenance analysis for endpoint detection and response systems », *in*: *2020 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2020, pp. 1172–1189.

[45] Simon Hawkins et al., « Outlier detection using replicator neural networks », *in*: *International Conference on Data Warehousing and Knowledge Discovery*, Springer, 2002, pp. 170–180.

[46] Shilin He et al., « Experience report: System log analysis for anomaly detection », *in*: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, 2016, pp. 207–218.

[47] Zengyou He et al., « A frequent pattern discovery method for outlier detection », *in*: *International Conference on Web-Age Information Management*, Springer, 2004, pp. 726–732.

[48] Sepp Hochreiter, « The vanishing gradient problem during learning recurrent neural nets and problem solutions », *in*: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.*02* (1998), pp. 107–116.

[49] Sepp Hochreiter and Jürgen Schmidhuber, « Long short-term memory », *in*: *Neural computation* 9.*8* (1997), pp. 1735–1780.

[50] Adam Horvath, *MurMurHash3, an ultra fast hash algorithm for C#/.NET*, 2012.

[51] Md Nahid Hossain et al., « {SLEUTH}: Real-time attack scenario reconstruction from {COTS} audit data », *in*: *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 487–504.

[52] Olakunle Ibitoye et al., « The Threat of Adversarial Attacks on Machine Learning in Network Security–A Survey », *in*: *arXiv preprint arXiv:1911.02621* (2019).

[53] Matthew Jagielski et al., « Manipulating machine learning: Poisoning attacks and countermeasures for regression learning », *in*: *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 19–35.

[54] Iebeling Kaastra and Milton Boyd, « Designing a neural network for forecasting financial and economic time series », *in*: *Neurocomputing* 10 (1996), pp. 215–236.

[55] Yaakov Katz and W Jpost, « Stuxnet virus set back Iran's nuclear program by 2 years », *in*: *Jerusalem Post* 15 (2010).

[56] Yoon Kim et al., *Structured Attention Networks*, 2017, DOI: 10.48550/ARXIV.1702.00887, URL: https://arxiv.org/abs/1702.00887.

[57] James Kirkpatrick et al., « Overcoming catastrophic forgetting in neural networks », *in*: *Proceedings of the national academy of sciences* 114 (2017), pp. 3521–3526.

[58] Marius Kloft and Pavel Laskov, « Online anomaly detection under adversarial impact », *in*: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 405–412.

[59] Oleg Kolesnikov and Wenke Lee, *Advanced polymorphic worms: Evading ids by blending in with normal traffic*, tech. rep., Georgia Institute of Technology, 2005.

[60] Moshe Kravchik and Asaf Shabtai, « Can't Boil This Frog: Robustness of Online-Trained Autoencoder-Based Anomaly Detectors to Adversarial Poisoning Attacks », *in*: *arXiv preprint arXiv:2002.02741* (2020).

[61] Hans-Peter Kriegel, Matthias S hubert, and Arthur Zimek, « Angle-based outlier detection in high-dimensional data », *in*: *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, ACM Press, 2008, p. 444, ISBN: 978-1-60558-193-4, DOI: 10.1145/1401890.1401946.

[62] Christopher Kruegel et al., « Automating mimicry attacks using static binary analysis », *in*: *USENIX Security Symposium*, vol. 14, 2005, pp. 11–11.

[63] Taku Kudo, « Subword regularization: Improving neural network translation models with multiple subword candidates », *in*: *arXiv preprint arXiv:1804.10959* (2018).

[64] Taku Kudo and John Richardson, *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*, 2018, arXiv: 1808.06226 [cs.CL].

[65] Leslie Lamport, « Time, Clocks and the Ordering of Events in a Distributed System », *in*: *Communications of the ACM 21, 7 (July 1978), 558-565. Reprinted in several collections, including Distributed Computing: Concepts and Implementations, McEntire et al., ed. IEEE Press, 1984.* (July 1978), 2000 PODC Influential Paper Award (later renamed the Edsger W. Dijkstra Prize in Distributed Computing). Also awarded an ACM SIGOPS Hall of Fame Award in 2007., pp. 558–565, URL: https://www.microsoft.com/en-us/research/publication/time-clocks-ordering-events-distributed-system/.

[66] D. Lanoe, M. Hurfin, and E. Totel, « A Scalable and Efficient Correlation Engine to Detect Multi-Step Attacks in Distributed Systems », *in*: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, Oct. 2018, pp. 31–40, DOI: 10.1109/SRDS.2018.00014.

[67] Léo Lavaur et al., « The Evolution of Federated Learning-based Intrusion Detection and Mitigation: a Survey », *in*: *IEEE Transactions on Network and Service Management* (2022).

[68] Laetitia Leichtnam et al., « Forensic Analysis of Network Attacks: Restructuring Security Events as Graphs and Identifying Strongly Connected Subgraphs », *in*: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2020, pp. 565–573.

[69] Laetitia Leichtnam et al., « Heterogeneous Logs Graph Visualization and Clustering for Attack Traces Discovery », *in*: *2018 IEEE Symposium on Visualization for Cyber Security (VizSec)*, 2018.

[70] Laetitia Leichtnam et al., « Sec2graph: Network attack detection based on novelty detection on graph structured data », *in*: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2020, pp. 238–258.

[71] Wen-Nung Lie, « Automatic target segmentation by locally adaptive image thresholding », *in*: *IEEE Transactions on Image Processing* 4.7 (1995), pp. 1036–1041.

[72] Tsung-Yi Lin et al., *Focal Loss for Dense Object Detection*, 2018, arXiv: 1708.02002 [cs.CV].

[73] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou, « Isolation forest », *in*: *2008 Eighth IEEE International Conference on Data Mining*, IEEE, 2008, pp. 413–422.

[74] Fucheng Liu et al., « Log2vec: A Heterogeneous Graph Embedding Based Approach for Detecting Cyber Threats within Enterprise », *in*: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1777–1794.

[75] Wei Liu et al., « SSD: Single Shot MultiBox Detector », *in*: *Lecture Notes in Computer Science* (2016), pp. 21–37, ISSN: 1611-3349.

[76] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning, *Effective Approaches to Attention-based Neural Machine Translation*, 2015, DOI: 10.48550/ARXIV.1508.04025, URL: https://arxiv.org/abs/1508.04025.

[77] Cyprien de Masson d'Autume et al., « Episodic Memory in Lifelong Language Learning », *in*: *CoRR* abs/1906.01076 (2019), arXiv: 1906.01076, URL: http://arxiv.org/abs/1906.01076.

[78] Leland McInnes, John Healy, and Steve Astels, « hdbscan: Hierarchical density based clustering. », *in*: *J. Open Source Softw.* 2.*11* (2017), p. 205.

[79] Tomas Mikolov et al., « Efficient Estimation of Word Representations in Vector Space », *in*: *CoRR* abs/1301.3781 (2013).

[80] Sadegh M Milajerdi et al., « Holmes: real-time apt detection through correlation of suspicious information flows », *in*: *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 1137–1152.

[81] Alan Mills and Phil Legg, « Investigating Anti-Evasion Malware Triggers Using Automated Sandbox Reconfiguration Techniques », en, *in*: *Journal of Cybersecurity and Privacy* 1.*1* (Mar. 2021), pp. 19–39, DOI: 10.3390/jcp1010003, URL: https://www.mdpi.com/2624-800X/1/1/3 (visited on 10/07/2021).

[82] Yisroel Mirsky et al., « Kitsune: an ensemble of autoencoders for online network intrusion detection », *in*: *arXiv preprint arXiv:1802.09089* (2018).

[83] MITRE, *ATT&CK data sources*, https://github.com/mitre-attack/attack-datasources, Accessed: 16-03-2021.

[84] Bao N Nguyen et al., « GUITAR: an innovative tool for automated testing of GUI-driven software », *in*: *Automated software engineering* 21.*1* (2014), pp. 65–105.

[85] Guansong Pang, Kai Ming Ting, and David Albrecht, « LeSiNN: Detecting Anomalies by Identifying Least Similar Nearest Neighbours », *in*: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, IEEE, Nov. 2015, pp. 623–630, ISBN: 978-1-4673-8493-3, DOI: 10.1109/ICDMW.2015.62.

[86] Chetan Parampalli, R Sekar, and Rob Johnson, « A practical mimicry attack against powerful system-call monitors », *in*: *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, 2008, pp. 156–167.

[87] Cláudia Pascoal et al., « Robust feature selection and robust PCA for internet traffic anomaly detection », *in*: *2012 Proceedings Ieee Infocom*, IEEE, 2012, pp. 1755–1763.

[88] Thomas Pasquier et al., « Runtime analysis of whole-system provenance », *in*: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1601–1616.

[89] Vern Paxson, « Bro: a system for detecting network intruders in real-time », *in*: *Computer networks* 31 (1999), pp. 2435–2463.

[90] Judea Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*, Elsevier, 2014.

[91] Kexin Pei et al., « Hercule: Attack story reconstruction via community discovery on correlated log graph », *in*: *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, ACM, 2016.

[92] Murad A Rassam, Mohd Maarof, Anazida Zainal, et al., « Big Data Analytics Adoption for Cybersecurity: A Review of Current Solutions, Requirements, Challenges and Trends. », *in*: *Journal of Information Assurance & Security* 12.*4* (2017).

[93] Joseph Redmon and Ali Farhadi, « Yolov3: An incremental improvement », *in*: *arXiv preprint arXiv:1804.02767* (2018).

[94] Joseph Redmon et al., *You Only Look Once: Unified, Real-Time Object Detection*, 2016, arXiv: `1506.02640 [cs.CV]`.

[95] Shaoqing Ren et al., *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, 2016, arXiv: `1506.01497 [cs.CV]`.

[96] David Rolnick et al., « Experience Replay for Continual Learning », *in*: *CoRR* abs/1811.11682 (2018), arXiv: `1811.11682`, URL: `http://arxiv.org/abs/1811.11682`.

[97] Ishai Rosenberg et al., « Adversarial machine learning attacks and defense methods in the cyber security domain », *in*: *ACM Computing Surveys (CSUR)* 54.*5* (2021), pp. 1–36.

[98] Frank Rosenblatt, *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*, tech. rep., Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[99] Urko Rueda et al., « TESTAR: from academic prototype towards an industry-ready tool for automated testing at the user interface level », *in*: *Actas de las XX Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2015)* (2015), pp. 236–245.

[100] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, *Learning internal representations by error propagation*, tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[101] R. Sadoddin and A. A. Ghorbani, « An Incremental Frequent Structure Mining Framework for Real-time Alert Correlation », *in*: *Computers & Security* 28 (2009), pp. 153–173.

[102] Iqbal H Sarker et al., « Cybersecurity data science: an overview from machine learning perspective », *in*: *Journal of Big data* 7.*1* (2020), pp. 1–29.

[103] Bernhard Schölkopf et al., « Support vector method for novelty detection », *in*: *Advances in neural information processing systems*, 2000, pp. 582–588.

[104] Rico Sennrich, Barry Haddow, and Alexandra Birch, *Neural Machine Translation of Rare Words with Subword Units*, 2016, arXiv: `1508.07909 [cs.CL]`.

[105] Yun Shen and Gianluca Stringhini, « Attack2vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks », *in*: *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 905–921.

[106] Zafar Singhera, Ellis Horowitz, and Abad Shah, « A graphical user interface (gui) testing methodology », *in*: *International Journal of Information Technology and Web Engineering (IJITWE)* 3.*2* (2008), pp. 1–18.

[107] Pablo Sprechmann et al., *Memory-based Parameter Adaptation*, 2018, arXiv: `1802.10542 [stat.ML]`.

[108] Blake E Strom et al., « Mitre att&ck: Design and philosophy », *in*: *Technical report* (2018).

[109] tesseract-ocr, *tesseract*, `https://github.com/tesseract-ocr/tesseract`, 2019.

[110] Lisa Torrey and Jude Shavlik, « Transfer learning », *in*: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI global, 2010, pp. 242–264.

[111] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni, « Survey of machine learning techniques for malware analysis », *in*: *Computers & Security* 81 (2019), pp. 123–147.

[112] Fredrik Valeur, *Real-time intrusion detection alert correlation*, Citeseer, 2006.

[113] Sai Omkar Vashisht and Abhishek Singh, *Turing Test in Reverse: New Sandbox-Evasion Techniques Seek Human Interaction*, en, 2014, URL: `https://www.fireeye.com/blog/threat-research/2014/06/turing-test-in-reverse-new-sandbox-evasion-techniques-seek-human-interaction.html` (visited on 10/10/2021).

[114] Ashish Vaswani et al., « Attention is all you need », *in*: *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[115] Kalyan Veeramachaneni et al., « AI2: training a big data machine to defend », *in*: *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, IEEE, 2016, pp. 49–54.

[116] Rakesh Verma, « Security analytics: Adapting data science for security challenges », *in*: *Proceedings of the fourth ACM international workshop on security and privacy analytics*, 2018, pp. 40–41.

[117] J. Viinikka et al., « Processing intrusion detection alert aggregates with time series modeling », *in*: *Information Fusion* 10 (2009), pp. 312–324.

[118] David Wagner and Paolo Soto, « Mimicry attacks on host-based intrusion detection systems », *in*: *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, ACM, 2002, pp. 255–264.

[119] Evan D Wolff, KM Growley, MG Gruden, et al., « Navigating the solarwinds supply chain attack », *in*: *The Procurement Lawyer* 56.*2* (2021).

[120] Weng-Keen Wong et al., « Bayesian network anomaly pattern detection for disease outbreaks », *in*: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 808–815.

[121] Charles Xosanavongsa, « Heterogeneous Event Causal Dependency Definition for the Detection and Explanation of Multi-Step Attacks », PhD thesis, CentraleSupélec, 2020.

[122] Charles Xosanavongsa, Eric Totel, and Olivier Bettan, « Discovering Correlations: A Formal Definition of Causal Dependency Among Heterogeneous Events », *in*: *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, 2019, pp. 340–355, DOI: 10.1109/EuroSP.2019.00033.

[123] Tom Yeh, Tsung-Hsiang Chang, and Robert C Miller, « Sikuli: using GUI screenshots for search and automation », *in*: *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, 2009, pp. 183–192.

[124] Chong Zhou and Randy C Paffenroth, « Anomaly detection with robust deep autoencoders », *in*: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2017, pp. 665–674.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Science de la donnée en appui des opérations de cybersécurité

Mot clés : Supervision de sécurité, Détection d'attaques nouvelles, IA expliquable

Résumé : Pour se prémunir des organisations cyber-criminelles et des APTs, les opérateurs de systèmes d'information doivent définir et mettre en oeuvre des politiques de sécurité strictes. Cependant, il est impossible de définir et maintenir des politiques qui bloquent toutes les attaques sans impacter les fonctionnalités du système. Par conséquent, pour réagir au plus vite aux attaques, la supervision des systèmes et la réponse aux incidents de sécurité sont souvent confiées aux Security Operation Centres (SOC) et aux Computer Emergency Response Teams (CERT). Lors de la supervision de systèmes d'information, des comportements légitimes entraînent régulièrement des fausses alertes. Ceci entraîne une fatigue liée aux alertes, les analystes étant surchargés d'informations au point où ils ne sont plus attentifs aux variations causées par des comportements adverses. Cette thèse décrit des méthodes s'adressant aux analystes ne possédant pas de connaissances en science de la donnée. Elles leur permettent de créer et adapter des analytiques de sécurité qui reposent sur des méthodes d'apprentissage machine afin d'automatiser une plus grande part de leurs procédures d'investigation. Cette thèse se concentre sur la détection d'anomalies pour relever des comportements inhabituels, ainsi que sur le regroupement d'alertes par similarité. En particulier, l'application de ces méthodes à la détection d'attaques inconnues est explorée. Nous décrivons également une méthode permettant de générer de l'activité utilisateur dans le but de créer des jeux de données d'évaluation réalistes.

Title: Data science in support of cybersecurity operations

Keywords: Security monitoring, Novel attack detection, Explainable AI

Abstract: To defend against sophisticated cyber-criminal organizations and APTs, IT system operators should define and enforce strict security policies. However, defining and maintining perfect security policies that block all attacks and do not impact usability of the system is impossible. Therefore, security monitoring and incident response are often entrusted to Security Operation Centres (SOC) and Computer Emergency Response Teams (CERT). When monitoring an IT system, legitimate behaviours repeatedly triggers false alarms. This causes alert fatigue, as analysts are overloaded to the point they can miss subtle variations that are the consequence of adversary behaviours. This thesis describes methods to allow security analysts, with no expertise in data science, to create and adapt security analytics that leverage machine learning models to automate more of their investigation procedures. This thesis focuses on anomaly detection to highlight unusual behaviours and clustering to regroup similar alerts and avoid repeating the same alerts again and again. We specifically explore the application of these methods to novel attack detection. To assess our approach, we also describe a method to automatically generate user activity to create realistic datasets.