Detecting and visualizing anomalies in heterogeneous network events : Modeling events as graph structures and detecting communities and novelties with machine learning

Laetitia Leichtnam

▶ **To cite this version:**

Laetitia Leichtnam. Detecting and visualizing anomalies in heterogeneous network events : Modeling events as graph structures and detecting communities and novelties with machine learning. Cryptography and Security [cs.CR]. CentraleSupélec, 2020. English. NNT : 2020CSUP0011 . tel-03368501

HAL Id: tel-03368501

https://theses.hal.science/tel-03368501v1

Submitted on 6 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

DOCTORAT BRETAGNE LOIRE MATHSTIC

CentraleSupélec

THÈSE DE DOCTORAT DE

CENTRALE SUPÉLEC

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

Par

Laetitia LEICHTNAM

Detecting and visualizing anomalies in heterogeneous network events

Modeling events as graph structures and detecting communities and novelties with machine learning

Thèse présentée et soutenue à Rennes, le 11 décembre 2020
Unité de recherche : CentraleSupélec, IRISA
Thèse N° : 2020CSUP0011

Rapporteurs avant soutenance :

Hervé Debar	Professeur, Télécom SudParis
Davide Balzarotti	Professeur, Eurecom

Composition du Jury :

Président :	Christine Morin	Directrice de Recherche, Inria, Rennes
Rapporteurs :	Hervé Debar	Professeur, Télécom SudParis
	Davide Balzarotti	Professeur, Eurecom
Examinateurs :	François Lesueur	Maître de conférence, INSA de Lyon
	Anaël Beaugnon	Docteure, ANSSI
Dir. de thèse :	Eric Totel	Professeur, IMT Atlantique, Rennes
Co-dir. de thèse :	Ludovic Mé	ARP, Inria, Rennes
	Nicolas Prigent	Docteur, LSTI

Abstract

According to the National Institute of Standards and Technologies (NIST), to ensure the security of an information system it is required to identify threats, protect preventively the system, detect security incidents, respond to attacks and recover from them. Operationally speaking, Security Operational Centers (SOCs) are teams dedicated to the detection, response, and recovery. Their security analysts rely on intrusion detection and analysis tools.

In this thesis, we propose to help security analysts in their tasks by proposing a new approach to detect and display network anomalies. The goal of this thesis is twofold: detecting any security breach in real-time and, in addition, allowing a post-mortem analysis of the techniques used by the attackers.

A first difficulty lies in building a model to represent the various kinds of information the analysts have to handle. In particular, it is useful to represent security data in a way that ensures that the information is both machine-readable, for automatic treatment, and human-readable, for analysis by a human expert. In response to these objectives, we propose a data representation model based on a graph structure. To handle the very heterogeneous data types we have to consider, we rely on knowledge graphs, that allow semantic linking of diverse information.

Once the model in hand, we propose two automatic treatments. The first one focuses on the relations between the pieces of information represented by the links of the knowledge graph model. Using *community detection*, we select sub-graphs representing events that are strongly related to an alert or an IoC and thus relevant for forensic analysis. This brings information to the analyst to explain the alert or the IoC. The second automatic treatment we propose consists in applying *novelty detection* to the graph, in order to realize an anomaly-based intrusion detection system. While traditional approaches in anomaly detection need a large volume of normal and anomalous data to build a good learning model, novelty detection techniques need little or no anomalous data. The difficulty here is to feed the novelty detection algorithm with a graph structure. We indeed rely on a machine learning algorithm named autoencoder, an unsupervised learning technique that does not take a graph but a vector as input. We thus propose a transformation of the graph into a vector, encoding both information contained in the nodes and information related to the structure of the graph (links between nodes). Evaluations on CICIDS 2017 and 2018 datasets show that graph structures representation of security data handled by an autoencoder gives results that are better than common anomaly detection methods, even those based on supervised learning. Notice that our results are good both relatively to the detection rate (no or almost no false negatives) and for the false alert rate (very low amount of false positives).

Even being able to minimize the number of false positives, reducing the cost of alerts interpretation by analysts is also needed. The goal is here to provide the analyst with a representation of security-relevant data that reduces the time and efforts required to analyze alerts. In response, we

propose an immersive visualization of the graph representation in 3D. The visualization highlights the relations between security elements and malicious events or IOCs. It gives a good starting point to the analysts to explore the data and reconstruct a global attack scenario.

To sum up, the general objective of this thesis being to evaluate the interest of graph structures in the field of security data analysis, we propose an end-to-end approach consisting in a unified view of the network data in the form of graphs, a community discovery system, an unsupervised anomaly detection system and a visualization of the data in the form of graphs.

Resumé

Pour assurer la sécurité d'un système d'information, il est nécessaire, selon le National Institute of Standards and Technologies (NIST), d'identifier les menaces, de protéger le système d'information, de détecter les incidents de sécurité, de répondre aux attaques et de revenir à un état sain du système [18]. Les Security Operational Centers (SOC) sont constitués d'analystes de sécurité qui œuvrent à la détection et à la réponse à incident en s'appuyant notamment sur des outils de détection et d'analyse d'intrusions.

Pour détecter les attaques, les analystes ont à leur disposition deux types d'outils : des systèmes utilisant une approche par signature et des systèmes utilisant une approche par détection d'anomalies. Les premiers s'appuie sur des signatures d'attaques alors que les seconds reposent sur l'identification d'activités déviant significativement d'un comportement considéré comme normal. Cette dernière catégorie permet notamment de détecter des attaques de types *zero-day*, c-à-d des attaques jusqu'à alors inconnues. Cependant, les approches par anomalies doivent encore faire face à de nombreux défis [136] pour répondre aux contraintes opérationnelles des SOCs. Ces défis sont la diversité du trafic réseau observé, le manque de disponibilité de données relatives à des attaques pour l'apprentissage, le coût important des erreurs de détection, la difficulté à interpréter les résultats et la difficulté à évaluer les mécanismes de détection employés.

Dans cette thèse, nous proposons d'aider les analystes de sécurité en proposant une nouvelle approche pour détecter, visualiser et interpréter les anomalies dans les échanges réseau d'un système d'information. L'objectif de cette thèse est double : détecter les incidents de sécurité et permettre une analyse post-mortem (forensique) des techniques utilisées par les attaquants.

Les graphes sont rarement pour la détection des intrusions de bout en bout, allant de la représentation des données d'entrée du système de détection à la représentation visuelle des résultats. Dans cette thèse, un graphe est utilisé comme donnée d'entrée pour représenter de manière unifiée les données de sécurité issues de source hétérogènes. A la fin du traitement, un sous-graphe, résultant du processus de détection, est identifié. Il apporte à l'analyste, d'une part, des évènements correspondant à des anomalies, d'autre part, des informations liées au contexte dans lequel ces anomalies ont été produites.

Nos problématiques de recherche s'articulent ainsi autour des trois points suivants:

La construction d'un modèle de données adapté à la sécurité. Les informations de sécurité qui intéressent les analystes d'un SOC peuvent être de différentes nature : il peut s'agir de rapports de sécurité, de journaux d'évènements (logs) ou d'échanges réseau.

Dans cette thèse, nous nous concentrons sur les données liées aux connexions réseau mais nous traitons toutes les informations extraites de la couche IP à la couche Application (requête HTTP, mail, etc.) et intégrons des données issues de sources externes telles que les indicateurs de compromission (IoC, pour Indice of Compromise). En prenant en compte ces différentes données,

nous visons à améliorer la prise en compte du contexte dans l'analyse. Pour traiter ces données très hétérogènes, nous nous sommes appuyés sur des *graphes de connaissances*, qui permettent de lier sémantiquement des informations diverses. Un modèle construit sur un graphe de connaissances peut contenir différents types d'informations précieuses tout en permettant un traitement automatique des données.

La détection d'intrusions. De nos jours, les algorithmes d'apprentissage automatique sont souvent utilisés pour la détection des anomalies [47]. Les techniques actuelles de détection des anomalies reposent souvent sur un apprentissage supervisé, qui nécessite des ensembles de données labellisées. Chaque événement doit être analysé en profondeur pour déterminer s'il fait partie ou non d'une attaque. La labellisation des données est donc très coûteuse et, dans les faits, les experts en sécurité ne disposent pas souvent de données labellisées concernant leur propre système.

L'utilisation de techniques non supervisées, qui ne nécessitent pas de données labelisées, semble être une bonne alternative. Souvent, ces techniques ne sont pas aussi performantes que les techniques supervisées en terme de précision de détection.

Dans cette thèse, nous cherchons à rivaliser avec les techniques supervisées en terme de précision de détection. Nous explorons deux techniques non supervisées. La première se concentre sur les relations entre les éléments d'information représentés par les liens du modèle de graphe de connaissances. À l'aide d'une technique appelée *détection de communautés*, nous sélectionnons des sous-graphes représentant des événements fortement liés à une alerte ou à un IoC et donc pertinents pour une analyse forensique. La deuxième technique non supervisée que nous explorons est la *détection de nouveautés* appliquée à notre représentation de données sous forme de graphe.

La présentation des résultats à destination des analystes de sécurité. Même si l'on parvient à minimiser le nombre de faux positifs, il faut également réduire le coût de l'interprétation des alertes par les analystes. L'objectif est de fournir à l'analyste une présentation des données pertinentes pour la sécurité qui réduise le temps et les efforts nécessaires à l'analyse des alertes. Le défi est de traiter l'important volume de données tout en facilitant l'accès aux informations portées par le graphe.

Nous proposons dans cette thèse une visualisation immersive de la représentation du graphe. Cette visualisation met en évidence les relations entre les éléments de sécurité et les événements malveillants ou les IoCs. Elle donne un bon point de départ aux analystes pour explorer les données et reconstruire un scénario d'attaque en suivant les liens entre les différents objets.

L'objectif général de cette thèse est ainsi d'évaluer l'intérêt des structures de graphe dans le domaine de l'analyse des données de sécurité. Nous proposons une approche de bout en bout, appelée sec2graph, consistant en une vue unifiée des données du réseau sous forme de graphes, un système de découverte de communautés, un système de détection d'anomalies non supervisé et une visualisation des données de sécurité sous forme de graphes.

Nous donnons dans la suite de ce résumé les éléments saillants des différentes étapes de cette

approche de bout en bout.

Le modèle de représentation des données de sécurité

La supervision de sécurité produit un grand nombre d'événements réseau. La diversité des protocoles de communication génère de nombreux fichiers journaux aux formats très variés. De plus, ces fichiers journaux ne sont pas souvent explicitement liés les uns aux autres. Il est donc difficile d'obtenir une vue d'ensemble des activités sur le réseau.

Afin d'exploiter ce grand volume d'informations hétérogènes, les analystes de sécurité partent généralement d'un IoC, c'est-à-dire d'un observable qui suggère qu'une compromission a peut-être déjà eu lieu. Une adresse IP particulière ou un nom de fichier particulier trouvé dans un événement réseau sont des exemples d'IoC. Les analystes recherchent parmi tous les fichiers journaux disponibles toute information relative à cet IoC qui pourrait aider à analyser l'incident de sécurité qui a conduit à cet indicateur.

Pour aider les analystes dans cette analyse, il est nécessaire d'avoir une représentation des données capable de mettre en évidence les relations entre les événements. En particulier, il est utile de représenter les données de sécurité de manière à ce que l'information soit à la fois lisible par la machine, pour un traitement automatique, et lisible par un humain, pour une analyse par un expert. Nous proposons une nouvelle représentation graphique des événements réseau [85] basée sur les objets de sécurité (SO, pour Security Object). Les SO sont les nœuds du graphe généré et correspondent à un sous-ensemble d'attributs provenant des différents événements du réseau. Chaque attribut est lié à un type d'information qui est important du point de vue de la sécurité. La valeur d'un attribut est dérivée de la valeur d'un champ trouvé dans un événement de sécurité dans les fichiers journaux analysés. Un lien entre deux SO indique que ces SO ont été trouvés ensemble dans au moins un événement. Le graphe des objets de sécurité donne ainsi une vision unifiée et riche de ce qui s'est passée sur le réseau. Cette représentation est inspirée du modèle STIX (Structured Thread Information eXpression) [109].

A travers une étude de cas, nous avons montré l'utilité de la structure de graphe pour l'analyste de sécurité. En effet, la structure du graphe met en évidence des sous-graphes fortement connectés qui permettent à l'analyste de se concentrer sur une partie spécifique du graphe.

La découverte de communautés

Dans notre modèle, un graphe peut contenir des millions de nœuds et d'arêtes. Cependant, les cas d'utilisation étudiées révèlent que des sous-graphes peuvent être isolés pour aider l'analyste de sécurité à trouver des éléments d'information pertinents. Les activités normales du réseau sont par exemple plus susceptibles d'être représentées par des communautés fortes et interconnectées d'objets de sécurité. En revanche, les attaques consistent généralement en quelques événements et seront généralement représentées par des SO décentralisés dans le graphe complet. Les objets de sécurité générés à partir de valeurs d'attributs contenant des preuves de la même attaque

sont, par construction, fortement liés. Par conséquent, l'identification d'une attaque dans notre représentation par graphe consiste à identifier des sous-graphes denses entourant un objet de sécurité du type Indicateur et isolés des grands hubs (supposés représenter des activités normales) dans le graphe. Ces types de sous-graphes sont appelés communautés en théorie des graphes.

Nous avons proposé une stratégie pour adapté les approches de détection des communautés à notre modèle de représentation des graphes [85]. Nous avons implémenté un prototype qui découvre les communautés par différentes méthodes et évalué chacune d'entre elles dans la tâche de sélection des sous-graphes pertinents pour l'analyse des sous-graphes.

Nous avons comparé les résultats de plusieurs algorithmes de détection de communautés sur le jeu de données CICIDS 2017 pour différents types d'attaques. Les expériences ont montré que l'approche par détection de communauté obtient de bons résultats dans la sélection d'objets pertinents. Elles ont également montré que la génération de graphes s'adapte à de grands jeux de données comprenant des millions d'événements. La méthode proposée offre un moyen d'analyser une attaque à partir d'un IoC dans une démarche d'analyse forensique.

La détection de nouveautés

Comme la notion de communauté dans les graphes de SO semble distinguer clairement les attaques du trafic normal, nous proposons d'utiliser notre modèle de graphe pour structurer les données d'entrée d'un système de détection d'intrusion. Notre hypothèse est que les graphes de SO fournissent une description riche de ce qui s'est passé sur le réseau, et que cette description riche peut être exploitée efficacement par des mécanismes d'apprentissage automatique.

La détection de nouveauté est typiquement utilisée lorsque la quantité de données anormales disponibles est insuffisante, ce qui est notre cas car les données relatives aux attaques sont heureusement rares. Alors que les approches traditionnelles de la détection d'anomalies nécessitent un volume suffisant de données normales et anormales pour construire un bon modèle d'apprentissage, les techniques de détection de nouveautés ont besoin de peu ou pas de données *anormales* pour être efficaces. Les algorithmes classiques d'apprentissage automatique ne prennent pas un graphe en entrée, mais un vecteur. Le graphe doit donc d'abord être transformé en vecteur. Il faut à la fois coder les informations contenues dans les nœuds et les informations liées à la structure du graphe (liens entre les nœuds) d'une manière commune et unifiée.

Nous proposons un processus pour encoder efficacement les graphes d'objets de sécurité afin qu'un autoencodeur puisse apprendre à partir de données dites *normales* et ensuite détecter les activités anormales[87]. Nous proposons également différentes stratégies pour calculer un *score d'anomalie*, c'est-à-dire un score qui définit à quel point les données réelles diffèrent des données apprises lors de la phase d'apprentissage. Cette approche peut être appliquée à n'importe quel jeu de données sans labellisation préalable des données.

En utilisant les jeux de données CICIDS2017 et CICIDS2018, nous avons montré que la représentation des données de sécurité sous forme de graphe traitées par un autoencodeur donne de meilleurs résultats que les méthodes courantes de détection d'anomalies de la littérature (approches supervisées et non supervisées), y compris les celles utilisant l'apprentissage profond.

La visualisation et l'exploration de graphe

Pour réduire le coût de l'interprétation des alertes par les analystes, nous cherchons à fournir à celui-ci une représentation des données pertinentes. A cette fin, nous proposons une visualisation immersive en 3D. Cette visualisation met en évidence les relations entre les éléments de sécurité et les événements malveillants ou IoC. Elle donne un bon point de départ aux analystes pour explorer les données et reconstruire un scénario d'attaque global.

Pour fournir à l'analyste une présentation des données pertinentes qui réduira le temps nécessaire au traitement des alertes, nous nous sommes concentrés sur les techniques de visualisation de graphes. Les défis sont de faire face à la quantité de données à représenter, de communiquer au mieux les propriétés des données et de faciliter la phase d'interprétation.

Nous avons d'abord proposé une visualisation immersive de la représentation du graphe [88]. Cette visualisation met en évidence les relations entre les objets de sécurité et les événements malveillants et/ou les IoCs. Elle fournit un point de départ aux analystes pour explorer les données et reconstruire un scénario d'attaque en suivant les liens entre les nœuds. Nous avons créé un prototype basé sur la réalité virtuelle. Ce prototype est volontairement simple et offre des contrôles intuitifs pour que l'utilisateur puisse s'adapter rapidement.

Nous avons également proposé une représentation sous forme de *dendrogramme* des attributs des objets de sécurité [86]. Les dendrogrammes sont des représentations hiérarchiques qui aident l'analyste de sécurité à définir les critères d'agrégation des objets de sécurité et à se concentrer uniquement sur les parties intéressantes du graphe.

Conclusion

Pour répondre aux besoins des analyse travaillant dans les SOC, nous avons exploré dans cette thèse l'utilisation des graphes pour la représentation et l'analyse des données de sécurité.

Nous avons d'abord proposé un nouveau modèle d'événements réseau basé sur un graphe, le graphe étant constitué d'objets de sécurité. Ce modèle, basé sur STIX, est destiné à permettre aux analystes de relier facilement les informations pertinentes et facilite ainsi le processus d'analyse. Afin d'améliorer l'analyse forensique, nous avons ensuite proposé un processus basé sur la détection de communautés pour découvrir les objets liés à une attaque identifiée par un indicateur de compromission donné. Concernant la détection d'intrusion, nous avons proposé une technique non supervisée basée sur un autoencodeur pour détecter efficacement les anomalies en utilisant quatre stratégies différentes. À cette fin, nous avons fourni une méthode pour coder à la fois la structure et le contenu du graphe d'objets de sécurité. Enfin, nous avons présenté une approche immersive de visualisation de données permettant d'explorer le graphe.

Toutes ces contributions forment une pipeline de traitement des données de sécurité modélisées sous forme de graphe et permettent à la fois d'effectuer une analyse forensique des données ainsi que de la détection d'intrusion non supervisée.

Acknowledgement

First of all, I would like to express my gratitude to my thesis supervisors Eric Totel, Ludovic Mé and Nicolas Prigent for their support throughout this thesis. I was able to benefit from their advice and their wise criticism. They were always available, patient and I enjoyed debating with them throughout these four years.
I would like to thank Christine Morin, Hervé Debar, Davide Balzarotti, François Lesueur and Anaël Beaugnon for having taken an interest in my work and for having accepted to be members of the jury, and especially Anaël for her detailed proofreading of this manuscript.
During these four years I was part of the CIDRE team at CentraleSupélec. I want to thank them for giving me an academic perspective on research. I would like to thank the members of CIDRE for the scientific and technical discussions that I had with them over the years during the seminars. I also want to thank former CIDRE team members Damien Crémilleux and Christopher Humphries for our discussions and wish them the best with Malizen.
The realization of this work would have been even more difficult without the support of my colleagues and more particularly of Christelle and Pierre-Adrien who showed a great availability and support. It would have been even more difficult to realize this work without their help. I would also like to thank my colleagues who made me smile day after day during the writing of this thesis.
I would also like to thank my proofreaders Antoine and Pierre for their careful reading and their pertinent remarks.
Finally, I would like to express my deepest gratitude to my parents for their constant support and encouragement during these years.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

Information systems are omnipresent in all sectors of activity. These systems are the target of increasingly numerous and complex attacks. There is therefore an ever growing need for better and more reliable security.

The National Institute of Standards and Technologies (NIST) defined in the Cyber Security Framework (CSF) [18] five so called "core functions" that are required to ensure the security of an information system: identify threats against the system, protect the system against these threats, detect security violation, respond to these attacks and recover from them.

In this thesis, we focus on the detection and response aspects. We indeed seek to detect any security violation in real time and, in addition, to allow post-mortem analysis of the attackers' modus operandi (forensic analysis), which is according to the NIST part of the work to realize to respond to attacks. From an operational point of view, the teams dedicated to detection and forensic analysis belong to Security Operational Centers (SOCs). In this chapter, we first present SOCs, insisting on the challenges they face. We then position our research relatively to the functionalities that are classically offered by a SOC. Finally, we list our contributions.

1.1 Security Operation Centers

In this section, we present the functionalities needed in a SOC, the tools that enable implementing these functionalities, and the challenges these tools face. We limit here our analysis to the type of detection and the types of tools explored in this thesis, i.e. anomaly detection by machine learning.

1.1.1 SOC functionnalities

Onwubiko [111] advocates that SOC operations must focus on proactive, protective and continuous security oversight. This includes the ability to collect information about threats, to detect cyber attacks and, in turn, to implement appropriate response and recovery plans to contain, counter and mitigate cyber incidents. According to this reference, a SOC should thus offer these functionalities:

1. Gain general information about threats and about the system under protection (OS, applications, etc.). The sources of these pieces of information can be both internal and external. For example, internal devices may generate logs, events and alerts, while external sources may provide inputs about possible threats. An example of an external source is a list of Indicators of Compromise (IoC). An IOC is an observation on a network or operating system that indicates a computer intrusion. Examples of IOCs are: a virus signature, malicious file hashes, particular IP addresses, URLs or domain names of botnet command and control servers. Once indicators have been identified in an incident response process, they can be used for attack detection.
2. Centralize and analyze the information relative to threats and relative to the system to detect potential suspicious behaviors. This analysis may need some pre-processing such as normalization of logs, events, network information, IoC and other metrics.
3. Detect attacks. This detection is performed by Intrusion Detection Systems (IDS).
4. Display and analyze security-related data. This should be performed in real-time (at worst near real-time) with the help of visualization tools like dashboards. Dashboards are built to structure information visually so that analysts can quickly spot an incident.
5. Respond to detected attacks. This involves remediation plans and the execution of appropriate procedures defined in an incident response manual, e.g, forensic analysis to precisely determine how the attack was performed.
6. Recover from the detected attacks. This consists in restoring the system to a safe state. This ensures service continuity and disaster recovery capability.

Our work focuses on the centralize, detect, display, and respond functionalities. These functionalities indeed correspond to the analyst's need to obtain the most relevant information in order to better assess and respond to the situation. Getting all relevant information is usually called "situation awareness". The most common definition of situation awareness is given by Endsley in [45]: "*Situation awareness is the perception of the elements in the environment within a volume*

of time and space, the comprehension of their meaning, and the projection of their status in the near future". This definition is not specific to cyber security but Barford [14] has given specific requirements for cyber security: *"be aware of the current situation, be aware of the impact of the attack, be aware of how situations evolve, be aware of actors (adversaries) behaviors, be aware of why and how the current situation is caused, be aware of the quality of the collected situation awareness information and the decisions derived from this information and assess plausible futures of the current situation"*. In this thesis, we explore ways to achieve this situation awareness.

1.1.2 Tools used in a SOC and relative challenges

People working in a SOC (i.e., security analysts) use procedures and technologies to detect security policy violations in real time and realize forensic analysis of security incidents. In this purpose, analysts use Intrusion Detection Systems (IDS) and System Information and Event Management (SIEM). IDSes are tools dedicated to the identification of possible security incidents. They log information about these incidents and report them through alerts. SIEMs centralize alerts and logs to facilitate aggregation and correlation from multiple information system components.

Meeting Badford's requirements with these tools is not easy in an operational context, especially when all detection operations must be carried out in real time. This is especially true for the requirement "be aware of the current situation". Indeed, intrusion detection systems are currently not totally reliable: in addition to the problem of missed attacks, the analysis of potential false alarms can be tedious and time-consuming for the analysts. To detect attack, operators have two kinds of intrusion detection system at their disposal: misuse detection systems and anomaly detection systems. The first ones depend on the manual or automatic generation of detection rules whereas the second depends on the identification of activities that differ significantly from behaviors defined as normal. Misuse detection systems detect only known attacks and are thus inefficient to detect zero-day attacks. To handle these disadvantages, extensive research has been performed on anomaly detection that may detect previously unseen attacks. Nowadays, anomaly detection extensively uses machine learning. We follow this path in this thesis, focusing on this form of detection and using machine learning techniques.

Anomaly detection nevertheless presents several drawbacks. In [136], the authors stated that anomaly detection is rarely employed in "real world" operations because the tools borrowed from the machine learning community are not well-adapted to intrusion detection. They inventory the main operational constraints that this type of detection methods must meet to ensure adequacy. The main challenges identified by the authors are the following:

Diversity of network traffic Network traffic is often very diverse, which leads to problems in detecting anomalies in operational environments. The most fundamental characteristics of the network, such as the number of connections, their duration, or the type of protocols, can show immense variability, which can make them unpredictable. Moreover, traffic diversity can be seen in each layer of the OSI Model from the data link layer to the application layer. The model must be able to take into account the diversity of these characteristics.

Low amount of attack-related data Anomaly detection can be seen as a classification

problem with two classes, normal and abnormal. The learning phase of machine learning algorithms needs a significant amount of data for each class. However, attacks often represent only a very small portion of network traffic and can be relatively silent, making them very difficult to learn. Moreover, one of the aims of anomaly detection is to find novel attacks, that cannot by definition be learned. To perform well, those algorithm must therefore have a perfect model of normality.

High cost of errors In intrusion detection, the cost of false positives is very high. False positives require spending a lot of time examining reported incidents only to determine if they really reflect malicious activity. Axelsson introduced in [10] the concept of base-rate fallacy for intrusion detection. He stated that even a very small rate of false positives can quickly make an intrusion detection system unusable. False negatives, on the other hand, can cause serious damage to an organization: even a single missed attack can compromise an entire information system.

Interpretation of the results by the analyst There is a need to interpret the results from the analyst's point of view. The analyst is indeed the key to go from *anomaly detection* to *intrusion detection*. He or she must be able to identify the decisive criteria for the raising of an anomaly, for instance when an event violates the security policy.

Evaluation difficulties Evaluation are performed with public datasets as sources. These datasets are often not representative of real data. As a result, methods tested on such datasets may work very well with forged datasets but may perform poorly on a real one.

In this thesis, we propose machine learning algorithms for security anomaly detection while responding to the challenges mentioned above.

In the next section, we present our research problem and the hypothesis we made to answer it.

1.2 Research problem and hypothesis

To respond to the challenges previously identified and respond to the requirements of situation awareness, we explore in this work the use of graphs for security data representation, analysis and visualisation. Our hypothesis is that graph representation can help SOC analysts in each of these tasks.

Relatively to these three tasks, our research questions are the following.

1.2.1 How to build a data model suited to security?

Before entering into the structuring of data and the selection of relevant information, it is necessary to study the different data sources and their contribution to the detection of attacks.

Security information of interest to analysts can be split in four main families: *static external data*, *dynamic external data*, *static internal data*, and *dynamic internal data*.

Static external data refers to all cyber security-related information from an external source. It includes IoC like the IP address of a suspected command and control server, a suspicious domain name, or a URL that references malicious content. It also includes threat intelligence reports that describe the Tactics, Techniques, and Procedures (TTPs) of attackers: actors, types of systems and information being targeted, and other threat-related information that provides greater situation awareness to an organization. Security alerts are also valuable external sources of information. They take the form of bulletins and vulnerability notes about current vulnerabilities, exploits, and other security issues. *Dynamic external data* refers to data coming from supervised systems that are not part of the information system to protect, such as honeypots. These systems can provide useful information on attack techniques. *Static internal data* contains all internal information that relates to the information system to be protected. This includes machines, applications, deployed configurations or detection procedures for instance. *Dynamic internal data* corresponds to all time-stamped events that have occurred on the information system. In this category, we find system and application event logs, intrusion detection system alerts, etc. Intrusion detection systems often focus on specific information in data such as the duration of a network connection or the number of packets for network events or domain name queries in DNS logs. However, the correlation of all these events is more useful to detect intrusion.

The difficulty lies in building a model that can represent these various kinds of information. It is also necessary to represent security data in a way that ensures that the information is both human-readable and machine-readable. By *readable*, we mean that a human or a process can analyze, interpret, integrate new knowledge and deduce how to react to the information received.

In this thesis, we focus on data related to network connections but we handle all pieces of information extracted from the IP layer to the Application layer (http request, mail, etc.) and integrate external data information such as IoCs. To process this very heterogeneous data, we relied on *knowledge graphs*, that allow semantic linking of diverse information. A model built on a knowledge graph can contain different types of valuable information while allowing automatic processing of the data.

1.2.2 How to detect intrusion?

Nowadays, machine learning algorithms are often used for anomaly detection [47]. Current anomaly detection techniques often build on supervised learning, which needs labeled datasets during the learning phase. However, security experts often do not have labeled datasets of their own events and labeling data is very expensive [2]. Each event must be thoroughly analyzed to determine if it is part of an attack.

The use of unsupervised techniques, that do not require labeled data, seems to be a good alternative.

In this thesis, we explore two unsupervised techniques to detect intrusion, analyze data related to intrusions and deal with the challenges described in Section 1.1.2. The first one focuses on

the relations between the pieces of information represented by the links of the knowledge graph model. Using a technique called *community detection*, we select sub-graphs representing events that are strongly related to an alert or an IoC and thus relevant for forensic analysis. The second unsupervised technique we explore consists in applying *novelty detection* to anomaly detection. Novelty detection is typically used when the quantity of available abnormal data is insufficient. This is our case as attack-related data is rare. While traditional approaches to anomaly detection need a sufficient volume of normal and anomalous data to build a good learning model, novelty detection techniques need little or no anomalous data to be efficient. A challenge is to give a graph structure as an input to the algorithm. Classical machine learning algorithms indeed do not take a graph, but a vector, as an input. The graph must therefore be transformed into a vector. This is challenging as one must both encode information contained in the nodes and information related to the structure of the graph (links between nodes) in a common unified manner.

1.2.3 How to present results to an analyst?

Even being able to minimize the number of false positives, reducing the cost of alerts interpretation by analysts is also needed. The goal is to provide the analyst with a presentation of security relevant data that reduces the time and efforts required to analyse alerts. The challenge is to deal with the huge amount of data to be represented while making it easy to access the information carried by the nodes and links.

We propose in this thesis an immersive visualization of the graph representation. The visualization highlights the relations between security elements and malicious events or IOCs. It gives a good starting point to the analysts to explore the data and reconstruct an attack scenario by following the links between the nodes.

To summarize, the key hypothesis of this thesis is that graphs are a powerful structure for cyber security able to dramatically enhance cyber situation awareness. We also advocate that graphs can help improving anomaly detection and interpreting alerts with the help of visualization.

1.3 Contributions and thesis organization

While graphs are often used in sociological studies, they are rarely used for intrusion detection, from the input of the detection system to its output. In our work, a graph is used as an input to represent in a unified way the heterogeneous data to be analyzed. A sub-graph is identified as an output, which brings to the analyst, on the one hand, alerts corresponding to anomalies and, on the other hand, information related to the context in which these anomalies were produced.

The general objective of this thesis is to evaluate the interest of such graph structures in the field of security data analysis. We propose an end-to-end approach, called sec2graph, consisting of a unified view of the network data in the form of graphs, a community discovery system, an unsupervised anomaly detection system and a visualization of the data in the form of graphs. Figure 1.1 shows the whole workflow from the collection of network event logs to the detection of

anomalies and the discovery of interesting substructures with visualization. Each process of the workflow corresponds to one chapter, as described in the following.

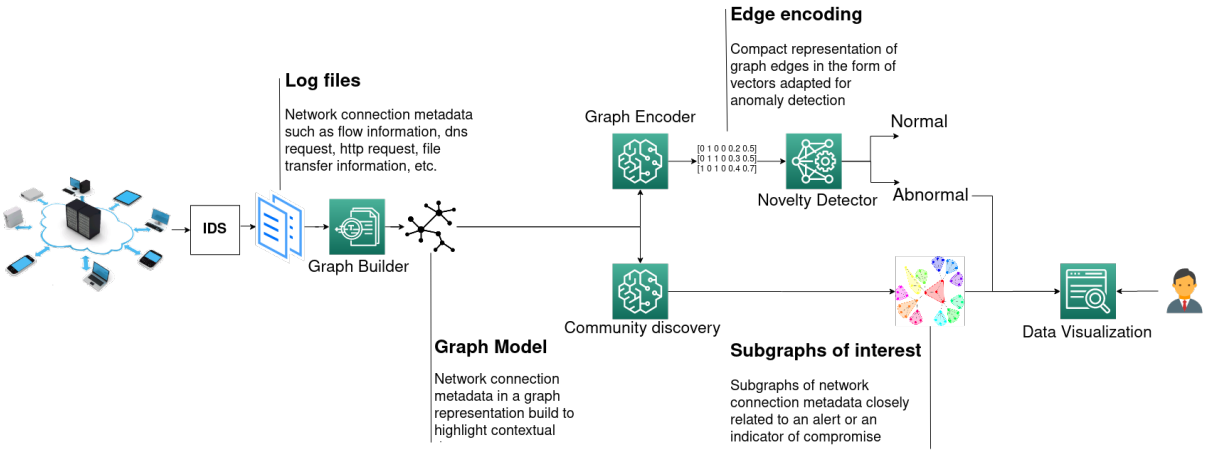

Figure 1.1 – Overview of the sec2graph workflow

Chapter 1 In this chapter, the state of the art, we present data models and ontologies used in the security field. We also review graph and machine learning techniques usages for computer security. Finally, we provide a state of the art on graph visualization techniques for SOC analysts.

Chapter 2 In Chapter 2, we propose a data representation model based on a graph structure. This is our first contribution. It corresponds to the *Graph Builder* process on Figure 1.1. We explain the different design choices we made, especially having in mind that our model should be usable by machines but also by humans. To illustrate our approach, we show how known attacks can be represented using our model.

Chapter 3 This chapter aims at demonstrating the usefulness of graph representation for community discovery. This is our second contribution, which corresponds to the *Community discovery* process in the Figure 1.1. We apply well-known graph algorithms to discover strongly related pieces of data and select among them the one that are linked to alerts and indicators of compromise. The content of this chapter has been published at the WTMC workshop [85].

Chapter 4 Chapter 4 explains how we process graph-structured data with an autoencoder to detect anomalies. This is our third contribution, which corresponds to the *Graph Encoder* and *Novelty Detector* processes illustrated in the Figure 1.1. We also compare in this chapter the results we obtained to those obtained by classical anomaly detection approaches. This comparison is based on experiments we conducted on two different datasets (CICIDS 2017 and 2018). The content of this chapter has been published at the DIMVA conference [87].

Chapter 5 Chapter 5 deals with the visualization of security data in the form of graphs. This is our fourth and last contribution, which corresponds to the *Data Visualization* process in

the Figure 1.1. We propose an immersive approach allowing the analyst to process the data in an "exploratory way": the analyst can move within the graph both in space and time. The content of this chapter has been partially published at the Vizsec symposium [88].

Chapter 6 This last chapter concludes this thesis with a summary of our outcomes, a discussion about the fulfilment of our objectives, and various perspectives for the future.

Chapter 2

State of the art

Contents

Introduction

This thesis mainly addresses three research areas: graph analysis, anomaly detection and visualization. Therefore, we retained these three major areas in this state of the art.

The first part deals with the way heterogeneous security-related pieces of data are collected and linked through different representation models. Then, we review approaches that use graphs for intrusion detection and attack analysis. As it is classical in intrusion detection, data is exploited either to find particular already-known patterns that are characteristic of a given attack (misuse detection) or to identify unusual behavior that indicates the presence of a known or unknown attack (anomaly detection). In this thesis, we focus on this second approach. Finally, we consider the visualization-related proposals that were made to detect and correlate events that are symptomatic of an attack.

2.1 Representing and handling security data with graphs

In any security data-oriented process, the first step consists in selecting and reorganizing data to obtain the most relevant pieces of information. The goal of this step is to represent this data coming from heterogeneous sources of information in a common way. In this section, we present the main existing standards describing which information is useful for security analysis and how it can be represented. Then, we present examples of security ontologies to model security events. Finally, we conclude on the limitations of the models we identified in the literature, and we present what motivated our contributions.

2.1.1 Structured models of cyber security information

Security analysts face a huge volume of data to process. Furthermore, some pieces of information are difficult to use as they are not well structured, making their analysis difficult. Based on this observation, many initiatives emerged that are aimed to standardizing the way security data is represented, regardless of the data sources. To promote information sharing and improve attack detection capabilities, some initiatives from international groups have become standards.

2.1.1.1 The MITRE standards

Considerable efforts have been devoted to categorize security information and standardize data formats and exchange protocols, most notably through the Making Security Measurable (MSM) [100] initiative led by MITRE (MIT Research Establishment). The main goal of MSM [1] consists in "*improving the measurability of security through registries of baseline security data, providing standardized languages as means for accurately communicating the information, defining proper usage, and helping establish community approaches for standardized processes.*" To this end, the MITRE proposed several standards listed in table 2.1.

The CVE [99], CCE [98], CWE [101], CPE [32], OVAL [13], ATT&CK [139] and CAPEC [15] frameworks make it possible to standardize the structure of a given type of security information, for example vulnerabilities for CVE. The goal of these standards is to provide the various stakeholders with a common way to describe pieces of data, for example vulnerabilities. More precisely, the CVE standard includes a description of the vulnerability, a score for criticity, the release and revision date and one or more references to external documentation. This allows the security community to keep up to date about all known vulnerabilies in one common inventory. Similarly, the CCE [98] standard refers to configuration requirements. For example, the CCE entry CCE-5402-3 refers to the following security recommandation: "The SSH login banner should be set appropriately." The CWE standard, proposes a classification of software weaknesses commonly found in implementations. For example, CWE-267 refers to implementations that allows a privilege

1. https://makingsecuritymeasurable.mitre.org/

Standard	Description
Common Vulnerabilities and Exposures (CVE) [99].	Standard identifiers for publicly known vulnerabilities.
Common Configuration Enumeration (CCE) [98].	Standard identifiers for configuration issues.
Common Weakness Enumeration (CWE) [101].	Standard identifiers for software weakness types in architecture, design, or implementation that lead to vulnerabilities.
Common Platform Enumeration (CPE) [32].	Standard identifiers for platforms, operating systems, and application packages.
Open Vulnerability and Assessment Language (OVAL [13].	A language for determining vulnerability and configuration issues on computer systems.
Adversarial Tactics, Techniques and Common Knowledge (ATT&CK) [139].	A knowledge base of adversary tactics and techniques based on real-world observations.
Common Attack Pattern Enumeration and Configuration (CAPEC) [15].	A catalog of common attack patterns and a comprehensive schema for describing related attacks and sharing information about them.
Malware Attribute Enumeration and Characterization (MAEC) [73].	A language to describe malware in terms of attack patterns, detritus, and actions.
Cyber Observables eXpression (CybOX) [17].	A standardized language for encoding information about events observed in an operational domain.
Structured Threat Information eXpression (STIX) [126].	A standardized language for encoding information about attack campaigns.

Table 2.1 – Description of common security standards and inventories

entity to perform unsafe actions. The CPE standard [32] inventories common platform, operating systems and software versions. OVAL [13] describes information about the configurations set up in a machine and it security state. CAPEC [15] and ATT&CK [139] are two approaches that organize knowledge about adversarial behavior. ATT&CK describes common tactics, techniques and procedures used by the attackers organized in several categories such as the reconnaissance phase which consist in collecting information about an information system. CAPEC is focused on application security and describes the common attributes and techniques employed by adversaries to exploit known weaknesses in cyber-enabled capabilities (e.g., SQL Injection, XSS, etc.)

All these common standards handle pieces of data that are often strongly connected. For example, the CPE and CVE standards include information about products, the first one to describe the products in themselves and the second one to describe their known vulnerabilities. However, it is difficult to link pieces of information of both these standards, since, for instance, the names of the products referenced in the CVE descriptions are embedded in a text field. In addition, the CPE name of a product is not necessarily used.

Entity-association models. The CybOX [17] and MAEC [73] models have tackled this problem by proposing entity-association models that make it possible to represent information of varied natures (entities) and to represent links between these pieces of information as associations. In other words, they propose graph-based structures to link information of different types (vulnerability, software, attack campaign, etc.).

CybOx [17] is a standardized language for representing information about cyber-observables. It is intended to be flexible enough to allow to describe both cases of observables that have been measured in an operational context as well as more abstract models for potential observables that could be targets. It offers a total of 81 basic objects such as the *Linux Package* object or the *GUI Window* object. However, the diversity of the proposed objects and the complexity of its schema

(deep hierarchical tree) made its application difficult.

MAEC [73] is a structured language for representing information about malware based upon attributes such as relations between malware samples and their behaviors. MAEC aims at:

- enabling correlation, integration, and automation;
- improving human-to-human, human-to-tool, tool-to-tool, and tool-to-human communication about malware;
- allowing for the faster development of countermeasures by enabling the ability to leverage responses to previously observed malware instances;
- reducing potential duplication of malware analysis efforts by researchers.

While these two standards were regularly updated, they both have evolved and merged in a new graph-based model called STIX.

STIX is a language format used to exchange CTI. The version 2.0 of STIX is based on the MAEC and CAPEC standards and integrates the CybOX standard in a restructured form (18 new objects containing more information against 81 objects and without the complexity of the deep hierarchical tree architecture). STIXv2.0 represents information related to attack targets, attackers (adversaries), their modus operandi (TTPs), indicators of compromise (IoC) that could reveal an attack, IoC observed in the logs, exploitable vulnerabilities and remediation procedures. STIXv2.0 also allows to represent the relations between all these pieces of information. STIXv2.0 is thus a connected graph of nodes (the objects describing security-related pieces of information) and edges (the relations). It allows us to describe events as a collection of heterogeneous objects in a standardized way, which was not possible with the previous standards [2].

In details, STIXv2.0 is made of *STIX Domain Objects* and *STIX Cyber-observable Objects* defining the nodes, and STIX relations (including both external *STIX Relationship Objects* and embedded relations) defining the edges. The STIX v2.0 model is shown in Figure 2.1.

STIX Domain Objects (SDO), shown on the right side of the figure, are objects representing key CTI concepts such as an attacker's identity or modus operandi. Attack Pattern, Vulnerability or Indicator are example of SDO. The Indicator object is particularly interesting because it contains a pattern that can be used to detect suspicious or malicious activity or can refer to the signature rule of an IDS. It is thus the key object to link expert knowledge on observed data about an attack.

STIX Cyber-observable Objects (SCO), shown on the left side of the figure, are objects that represent observed facts about a network or host that may be related to a CTI concept to form a more complete understanding of the observed threat. The information about the actual compromise indices observed in the logs is particularly relevant to our work.

STIX Relationship Objects (SRO) are the set of all relations between SDO or between an SDO and an SCO. They are depicted in grey in Figure 2.1. Note that the Sighting object, shown in purple in the middle of the figure between the SDO and SRO, is also a SRO in STIX model. This particular object denotes that something in CTI (e.g., an indicator, malware, tool, threat actor, etc.) was identified.

When this thesis began, STIX 2.0 did not contain any SRO links between cyber-observables

2. notice that the CybOX model was a first attempt to do this

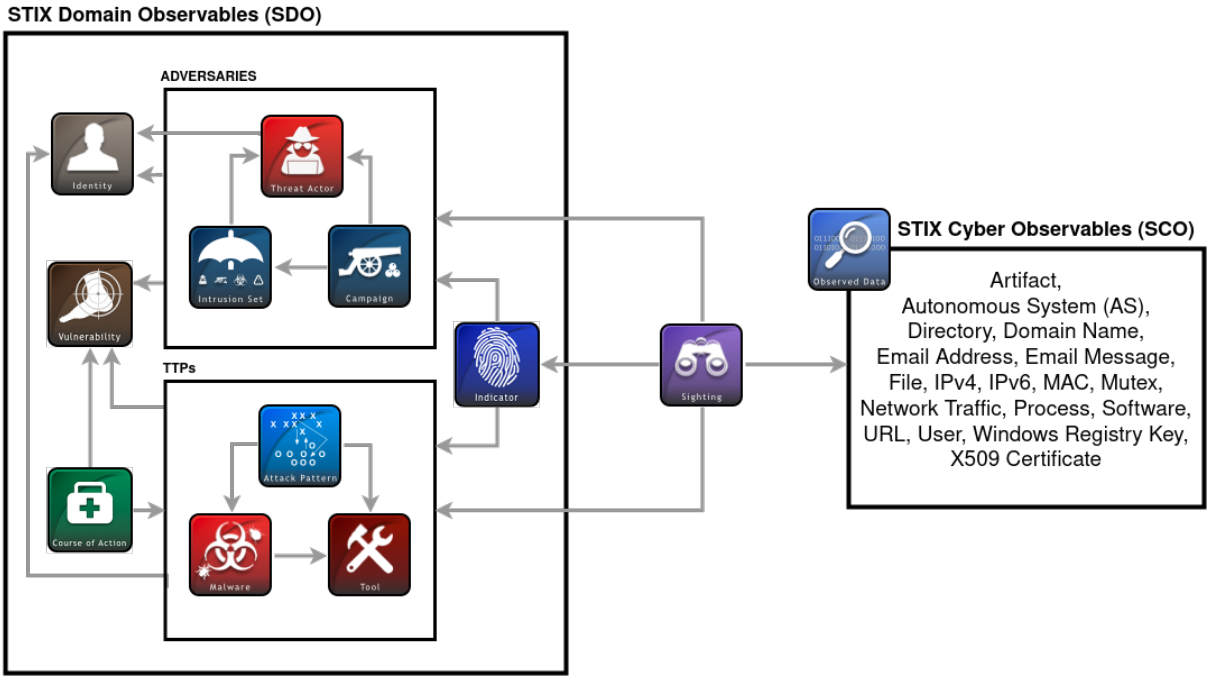

Figure 2.1 – STIXv2.0 Model

(SCO). My proposal was to add these links. While refining STIX for the 2.1 specification, the committee reached consensus that the STIX 2.0 graph model (with links only between SDO and SRO, and between SDO) was insufficient to support critical CTI use cases. Consequently, in STIX 2.1, they introduce seven types of relations between these types of cyber-observable Objects: domain-name, ipv4-addr, ipv6-addr, mac-addr and autonomous-system. This modification published on March 20, 2020 is closer to the model proposed in this thesis but the number of relations between cyber-observables is still very restricted. In this thesis, we propose a larger number of links between cyber-observables.

STIX is the first significant unified language able to represent both external threat intelligence and internal observables. However, it is not designed to represent all possible objects that can be found in network events but only those that are already identified as related to an attack. Moreover, only a few relations between SCOs are described. As will be presented later, our approach is similar to STIX and CybOX as it merges different types of security-related pieces of data in a unique model with a graph-based structure. However, our proposal differs from STIX and CybOx by allowing to represent all events in a forensic analysis perspective.

2.1.1.2 Cyber threat intelligence models and platforms

While the MITRE Consortium provides several standards on which we based our data model and notably the CybOX and the STIXv2 model, we were also interested in platforms to share IoC to identify objects and relations of interest. We discuss here the most used and referenced

tools and models, that are OpenIOC [55], IDMEF [37]/IODEF [36], OTX Alienvault [112] and MISP [153]. In addition, we describe the OpenCTI [6] proposition.

OpenIOC [55] is focused on describing technical characteristics of a given threat through an extensible XML schema. It offers a comprehensive vocabulary for describing low level attributes. To that end, it is made of 1211 different terms like the *DNS Host* that can be used to find malicious hostnames that an infected host attempted to resolve or the *Service Name* often used to look for known malicious services. These IoC terms can then be used as an input to configure various IT security monitoring and detection tools like anti-virus, IDS, firewalls, etc. This standard is actually similar to the first version of CybOX which used an XML format to describe security events and malware (what Barnum [16] calls "cyber-observable"). This format shares with ours the idea of describing data through cyber-observables.

IODEF and IDMEF. The IETF normalized two RFCs related to the description of security incidents: the IODEF (Incident Object Description and Exchange) format [36] defined in RFC5070 and updated in RFC7970 and the IDMEF (Intrusion Detection Message Exchange Format) format [37] defined in RFC4765 and RFC4766.

IDMEF [37] defines information that should be issued by intrusion detection systems in their alerts. It is an object-oriented model that consists in 33 classes containing 108 fields. IDMEF proposes to standardize the way alerts are written by enriching the alerts with information allowing the analyst to understand the context of the attack. This model can be used to describe the source of an attack, the target or the exploited vulnerabilities. It is notably used by the Prelude SIEM [120]. In France, IDMEF and IODEF are the models recommended in the state administration to ensure interoperability of the different services [143]. In our case, we propose a model to normalize the way network events are described independently of their malicious nature. Indeed, we consider that a network event can be linked to an attack without having triggered an alert.

IODEF [36] aims at facilitating the exchange of information between Computer Security Incident Response Teams (CSIRTs). IODEF extends IDMEF to the description of incidents as higher level elements, while being able to include or reference an IDMEF alert as initial information about a security incident.

OTX (AlienVault Open Threat Exchange) [112] is an online platform for sharing information about cyber-threats, malware or fraud campaigns. OTX offers corresponding Indicator for these threats, malware and fraud campaigns. OTX supports the following indicators: Classless Inter-Domain Routing (CIDR) rules, CVE, five different formats of FileHash, FilePath, IP addresses, Mutex name, NIDS rules, URI, URL, YARA rules, Domain, Email, and Hostname. Notice that, these indicators can be found in the STIX Cyber Observables objects. In addition, there are detection rules (YARA rules, NIDS rules) found in the STIX Indicator object as well as vulnerabilities (CVE) found in a dedicated Vulnerability object of the STIX model. These similarities between these two models show that these types of objects are of particular importance in the detection of attacks.

MISP (Malware Information Sharing Platform) [153] is an online platform for collecting and sharing IoCs. The main motivation of MISP is to offer a comprehensive set of information, and to

allow the user to decide by him-/herself the level of granularity of information that he/she wants to share and import. It is for instance possible to collect information that are only related to a given APT, or to selectively share information with some specific partners. The MISP core format is a simple format used to exchange events and attributes. It consists in 165 attributes, each one corresponding to one or more of the 16 categories proposed by MISP. Among these attributes, 40 attributes belong to the category of particular interest to us, the Network Activity category. For example, port, ip-dst, ip-src, domain and email-body belong to the Network Activity category. MISP objects are added to the MISP attributes to allow advanced attribute combinations. For example, the MISP network connection object is composed of eleven attributes: community-id, dst-port, first-packet-seen, hostname-dst, hostname-src, ip-dst, ip-src, layer3-protocol, layer4-protocol, layer7-protocol, and src-port. A series of relations are also defined along with the objects which can be used to create relations between objects.

To build our model, we selected some types of Indicator/Object proposed by OTX and MISP. We limited ourselves to those types that are widely used by analysts and, of course, to those types that are adapted to the data we could collect, namely network traffic. For example, we find in our model objects such as IP address, Port, Filename or Email. As these objects where not sufficient to represented the relations between them, we created more complex objects, composed of other objects to express the observed data. For example, the NetworkConnection object is a composition of source and destination addresses among others. This composition is expressed by a relation in our model.

Released more recently by the French ANSSI, OpenCTI [6] is an open source platform allowing organizations to manage their cyber-threat intelligence knowledge and observables. The goal of OpenCTI is to model the full understanding of a threat or campaign without limitation. To that end, it proposed mechanism to structure, store, organize and visualize technical and non-technical information about cyber-threats. The pieces of data are structured using a knowledge schema based on the STIXv2 standards. The model is composed of hierarchical entities and relations. In their roadmap (to the best of our knowledge this model has not been implemented), three objectives are directly linked to the notion of graph model: the exploitation of the hypergraph potential, the implementation of a graph exploration engine, and the optimal use of graph traversing and search algorithms. The OpenCTI roadmap shows the interest for SOCs to exploit the potential of graphs in security event analysis and threat detection. Our work is consistent with it.

2.1.1.3 Security ontologies

An ontology is a *consensual, formalized textual specification of conceptualizations* [58]. It provides common and shared knowledge about a domain and can be communicated to people and application software. Previous standards and models do not always meet the needs of security analysts, so many models and ontologies have been developed to better address these issues. In the following, we discuss four different approaches.

Lu *et al.* [96] propose an ontology that contains the information modelled by STIX, but also additional information useful to the analyst. While STIX allows situational awareness by providing information related to IoC and observables, it does not provide information on activities not linked to an IoC, even if this activity could be informative. For example, some small pieces of information captured on networks can be valuable in the presence of other elements. This is why the authors integrate information from network sensors (for example Snort alerts) in their ontology and emphasize the recognition of attacks. This approach allows mixing internal data with vulnerability data but has not been tested on a large scale. The authors indeed only present two queries over the ontology involving Snort alerts and vulnerabilities. In addition, this approach is only tested using a subset of the Skaion dataset (the IDS Snort alerts for the CGI buffer overflow attack). By contrast, we tested our approach by taking into account all the metadata of network sessions on a large volume of data. Moreover, our approach is based on an automatic anomaly detection process while [96] is based on manual query. We believe that automation allows to take into account many possible relations between objects, when the manual approach is necessarily limited to the relations that come to the mind of the analyst.

Some approaches merged several standards and ontologies to get the most of each model. Following this path, CoCoa [111] is an ontology designed for SOC analysts. It proposes a model based on a knowledge-graph that links internal and external sources of information. It can handle five types of threat intelligence and information sources: events and logs, network information, structured digital feed, semi- and un-structured feed, and threat intelligence. The authors argue that the model allows for a proactive monitoring and the detection of threats that would not have ordinarily been detected through only events and logs. Their ontology can also be used to identify the techniques, tactics and procedures of an attack through the analysis of the knowledge graph.

Similarly, the Unified Cybersecurity Ontology (UCO) [142] and the graph SEPSES [69] are intended to integrate information from external sources to improve situation awareness. Both these ontologies integrate heterogeneous data from diverse sources (CAPEC, CVE), but the UCO integrates a whole knowledge schema from cybersecurity systems models (e.g. CybOX and STIX) and cybersecurity standards (e.g. CAPEC, CVE). Both these pieces of work describe use cases to demonstrate the possibilities of graph reasoning. As in our approach, the authors of CoCoa, UCO and SEPSES aimed at highlighting the links between internal data, such as logs, and data from external sources. Related publications show that these ontologies can actually be queried for security analyses, thus being compliant with the NIST cybersecurity framework [18]. Our approach is rigorously similar, as it proposes a data model and uses it to automate security analysis. However, the authors of these ontologies do not provide feedback on experiments with a large volume of data. They only describe manually crafted queries on the knowledge graph, without any form of automation. They also do not explain how they extract interesting data in internal events. By contrast, our approach is fully automated and has been tested on the CICIDS dataset. In addition, our objective is intrusion detection, by contrast with theirs that consists in analyzing an attack after it was discovered. Nevertheless, we advocate that it would be possible for us to add information to our model so as to not only detect, but allow a deeper analysis of an already detected attack.

Ekelhart's approach. The three previous approaches focus on creating links between pieces of information from different external data sources or between events and external knowledge databases (e.g. vulnerabilities). In-depth analysis of log information to link events from various sources (e.g. firewall, syslog, web server log, database log) and establish causal chains has remained largely a tedious manual process until now. Eklehart *et al.* [43] focuses on an automatic process to create links between events. The approach they propose consists in linking together events that share similar information (e.g. two events, regardless of their nature, that refer to the same user, or the same file). The authors have implemented their solution and tested it on real data, using SPARQL queries to build links. Our approach is similar but focuses on network events instead of system events. Beyond this difference in the nature of the events we consider, our implementation choices (gremlin queries and a graph-oriented database) make our tool more scalable.

The issue of representing events and more broadly security data is not specific to SOCs. Forensic analysts have adapted some of the models presented above, adding specific elements that are often related to the legal context which is of course crucial in forensics. Considerations carried out in the forensic field have contributed to improve the models for the SOCs and vice versa. For example, Casey *et al.* [29] propose the DFAX (Digital Forensic Analysis eXpression) model, based on CybOX in which information relative to the actions of the attacker and those of the analyst (the later to control data integrity) is added.

In [30], an evolution of the DFAX model called CASE is presented. This new model is more flexible and can be used in many contexts. The extensibility of the model is important as new protocols and new applications can quickly outdate a model. The model is by construction extensible as it is possible to add new object types according to the needs and to the data used as inputs.

These references show that the graph structure is useful in post-mortem security analysis. DFAX [29] takes advantage of the comparison between information graphs constructed at different times. In addition, the ease of evolving the graph model by adding objects and links is highlighted by [30]. We want to show in this thesis that, similarly to forensics, graphs can be useful in the context of intrusion detection.

In this section, we showed that a number of ontologies have emerged to represent security data and facilitate manual or automatic security data processing. The aim of these representations is to improve situational awareness. The heterogeneity of the data sources, however, makes this task difficult. One of the tracks often favoured is the use of knowledge graphs that allows events to be placed in a context by linking them to other events or by linking them to knowledge from external sources. The STIX model is often used as the basis of these pieces of work.

Our approach is also based on this model, but, as we will show later, we will especially develop the links between internal events.

2.1.2 Automated analysis of graphs for security supervision

In this section, we first present work related to graphs encoding, i.e., how a human-readable graph can be transformed into a computer-readable one to enable a given processing or making it more efficient. In a second time, we present pieces of work relative to classical treatments that can be applied to graphs. If these treatments come from various domains, we remind that this thesis focuses on the domain of security supervision.

2.1.2.1 Definitions

In this section we give important definitions and notations that will be used to discuss the pieces of work presented in the next sections.

Definition 1 (Graph)**.** A graph $G = (V, E)$ is a collection V of n vertices $v_1, v_2, ..., v_n$ together with a set E of edges.

A graph can be directed, in which case there is a source and destination for each edge, or undirected. We distinguish *homogeneous graphs* which are graphs where all nodes in G are of the same type and all edges are of the same type, and *heterogeneous graphs* that have multiple types of edges and/or multiple types of nodes.

Definition 2 (First-order Proximity)**.** The first-order proximity captures the direct neighbor relations between vertices. It is the local pairwise proximity between two connected vertices. For each pair of vertices (v_i, v_j), if $(v_i, v_j) \in E$, the first-order proximity between v_i and v_j is 1; otherwise, the first-order proximity between v_i and v_j is 0.

First-order Proximity is often represented in an intuitive way by an adjacency matrix. The adjacency matrix A of G is an $n * n$ matrix where $A_{i,j} = 1$ if there is an edge between v_i and v_j, and $A_{i,j} = 0$ otherwise.

Definition 3 (n-order Proximity)**.** The n-order proximity captures the n-step relations between each pair of vertices. For each pair of vertices (v_i, v_j), the n-order proximity ($n \geq 2$) is the number of n-length paths between v_i and v_j.

In the following, we first present the challenges that are inherent to graph representation. Then, we present some techniques that have been proposed to that end.

2.1.2.2 Encoding the structure of a graph

How a graph is encoded depends on how it will be processed afterwards. Classical approaches such as data blocks linked by references or adjacency matrices are often used as a basis and are adapted to allow an easier or a more efficient processing. In particular, when the graph is too large, it is not necessarily desirable or even possible to encode all the information in the graph; it is then necessary to choose judiciously the information to be kept. In this section, we present some general requirements for graph encoding, and then give some examples.

Requirements for the encoding. In [164], the authors identify four challenges inherent to graph representation: structure preserving, content preserving, data sparsity and scalability. In addition to these challenges, Hamilton *et al.* [61] raise, among others, the problem of improving interpretability.

In relation with this thesis, we were especially interested in the three following issues: structure and content preserving, scalability and interoperability. Since we build our own graph, we left aside the data sparsity problem which refers to a problem of non-completeness of the graph (node relations or attributes).

Dealing with *structure and content preserving*, graph encoding must ensure that the outputs for two identical graphs are identical, and that, with respect to which feature is considered important, graphs that are similar/close to one another lead to similar/close outputs of the encoding. Generally speaking, any characteristic of the original graph (global structure, local structure, attributes of a node or edge) that is important in the sense of the field of application (and therefore of future processing), must be found in the result of the encoding.

For sec2graph, we have chosen to focus on the close neighborhood (local structures), our interest being to learn the normal links between events that are closed to each other. In addition, the values of the attributes of the nodes and edges involved in these local structures are taken into account in the encoding we propose (see Chapter 5).

Dealing with *scalability*, real-world graphs may consist in millions or billions of vertices. The potentially very large size of the graphs to encode is therefore a challenge for the graph encoding task. The result of the encoding of the nodes and edges must, nevertheless, be such that the processing to be applied afterwards can be applied in a time and/or memory efficient manner. However, the encoding algorithm itself should not be too time- and memory-consuming. In our case, the encoding should be carried out in near real time to allow the SOC to react as quickly as possible in case of an attack. As will be shown later, we ensure the efficiency of the encoding by considering only local structures of the input graph and do not seek to preserve the overall structure of the graph (see Chapter 5).

Dealing with *interpretability* [61], researchers must ensure that their encoding methods are truly representing relevant graph information, and not just exploiting statistical tendencies. We advocate that it is especially important for the domain of cybersecurity, where results of the treatments performed on the output of the encoding to detect an attack must be explainable for a security analyst.

For each piece of work that was performed during this thesis, we evaluated our proposal with respect to these criteria. More details will be found in the relevant chapters.

Encoding techniques. Graphs are particularly difficult to represent in an efficient way because their representation can be costly in terms of memory but also costly in terms of processing time during their analysis. The adjacency matrix is a well-known example of graph representa-

tion. This representation is a matrix of size n^2, with n the number of nodes in the graph. In our case, manipulated graphs can quickly have millions of nodes which becomes computationally expensive. Moreover, this representation only allows to represent the local structure of a graph (direct neighborhood of nodes). The attributes of nodes and edges are not taken into account and the computation of the global neighborhood of nodes (k-order proximity with $k \geq 2$) requires matrices operations.

To solve this problem, many approaches propose to encode the graph in a compact form while adapting the encoding of the graph to the processing that will be applied to it. This is often done using a learning algorithm [61]. The objective is to learn which characteristics of the graph are the most interesting for the processing applied to the graph and thus to keep only a limited number of characteristics. The general idea is based on a learning algorithm based on an encoder and a decoder. This principle described in Hamilton's survey [61] is illustrated in Figure 2.2.

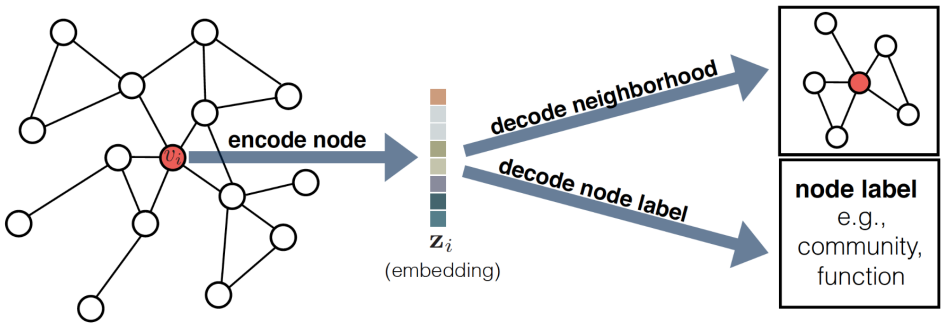

Figure 2.2 – Overview of the encoder-decoder approach as illustrated in Hamilton's survey .

First, all the information on a node of the graph that are identified as useful are concatenated. For instance, this can be its attributes, information about its neighbors, or structural information such as node centrality or position in the graph. The encoder computes from this vector v_i a vector z_i of reduced dimension. Next, a decoder rebuilds user-specified information from the small z_i vector. This may be information about the local neighborhood of the v_i node (e.g., the identity of its neighbors) or a classification label associated with the v_i node (e.g., a community label). By jointly optimizing the encoder and decoder in successive iterations, the system learns which characteristics of the graph are important for graph analysis while providing encoding of the graph in a small vector space (the z_i vector).

There are three main kinds of techniques for encoding a graph: matrix factorization-based techniques, random-walk-based techniques, and deep learning techniques. We present each of them in the following. Then, we detail which techniques fulfill the requirements mentioned in the previous section, as well as their advantages and disadvantages.

Factorization-based algorithms represent the connections between nodes in the form of a matrix and factorize this matrix to obtain an encoding. Among others, the matrices used to represent the connections include node adjacency matrix, Laplacian matrix and node transition probability

matrix. GraRep [28] is a technique for learning vertex representations of weighted graphs. This technique learns low dimensional vectors to represent vertices appearing in a graph and integrates global structural information of the graph. It uses the transition probability matrix. HOPE (High-Order Proximity preserved Embedding) [113] embeds a graph into a vector space where the structure and the inherent properties of the graph are preserved. HOPE can preserve n-order proximity of large scale graphs. These techniques make it possible to preserve the structure of the graph in its representation but not its content. Moreover, matrix calculation is often expensive, making these techniques poorly scalable.

Random walks have been used to approximate many properties in the graph including node similarity. A random walk is defined as a random path that connects two nodes in the graph. By traversing the neighborhood of a node in a random way, the algorithm builds an encoding of this node according to the attributes of the encountered nodes. As they perform a random sampling of the input graph, random walk-based algorithms are especially useful when the graph is too large to be taken as a whole. Encoding techniques using random walks on graphs to obtain node representations have been proposed in DeepWalk [116] and node2vec [57].

DeepWalk [116] generates multiple random walks to retrieve the neighborhood attributes and integrates them into the encoding. This technique preserves n-order proximity between nodes by maximizing the probability of observing the last k nodes and the next k nodes in the random walk centered at one node.

Similarly, node2vec [3] [57] preserves n-order proximity between nodes by maximizing the probability of occurrence of subsequent nodes in fixed length random walks. It differs from DeepWalk by using biased-random walks that include both breadth-first search (walk in the nearest neighborhood of the node) and depth-first search (walk going as far as possible from the origin node) strategies. Choosing the right balance between these two strategies allows node2vec to preserve the structure of the community as well as the similarity between nodes.

Deep neural networks have also been used to encode graphs. Generally speaking, the idea is to build an encoder and a decoder so that the output preserves certain characteristics of the input or so that the output is equal to the input. Auto-encoders are then generally used for the later.

Deep autoencoders are especially interesting for dimension reduction as they are able to model non-linear structures in the data.

In Structural Deep Network Embedding (SDNE), Wang *et al.* [154] proposed to use deep autoencoders to preserve the first- and second-order network proximity. The approach consists in linking two neural-based learning processes. The first one is unsupervised and aims to make its output equal to its input. It preserves the local structures of the graph that is encoded (i.e., the encoding allows for reconstructing the neighborhood of a given node). The second one is supervised and ensures that the encoding of two nodes with similar neighborhoods are themselves similar.

Deep Neural Networks for Learning Graph Representations (DNGR) [27] combines random walks with deep autoencoder. The model consists in three components: random walks, positive

3. Other very close propositions also exist: Struct2vec [124], [160]

point-wise mutual information (PPMI) calculation and stacked denoising autoencoders. Random walks are used on the input graph to generate a probabilistic co-occurence matrix, analogous to similarity matrix in HOPE. This matrix is transformed into a PPMI matrix (which measures the likelihood of co-occurrence of two elements) which is used in input of a stacked denoising autoencoder to obtain the encoding. A stacked denoising autoencoder improves the robustness of the encoding process in presence of noise in the graph. It also allows to capture the underlying structure required for tasks such as link prediction and node classification. These two methods take as input the global neighborhood of each node. This practice can be computationally expensive and thus sub-optimal for large graphs, limiting the scalabilty of the approach.

Graph Convolutional Networks (GCNs) [72] tackle the scalability problem by defining a convolution operator on graph. This supervised algorithm iteratively aggregates the encodings of a sampling of neighbors for a node (a function mixes the current encoding with the previous one to obtain the new encoding). Aggregating encodings of only local neighborhoods makes the approach scalable while multiple iterations allow the learned encoding of a node to characterize global neighborhood.

GraphSAGE [61] is also a general inductive framework that leverages node feature information (e.g., text attributes) to efficiently generate node encodings for previously unseen data. It is actually close to GCN, forming individual encodings for each node and generating encodings by sampling and aggregating the features of a node's local neighborhood.

DeepGL [128] is a general inductive graph representation learning framework for learning deep node and edge features that generalize across-networks. DeepGL starts by deriving a set of basic graph characteristics called graphlet features and automatically learns a multi-layer representation of the graph where each successive layer uses the output of the previous layer to learn the characteristics of a higher order. DeepGL naturally supports attributed graphs. In addition, DeepGL is scalable for large networks through efficient parallel implementation.

Table 2.2 summarizes the advantages and disadvantages of each types of techniques describe above relatively to the three first requirements, i.e. structure-preserving, content-preserving and scalability. The interpretability cannot be evaluated here as it mainly depends on the task that follows the encoding.

Types	**References**	**Advantages**	**Disadvantages**
Factorization-based	GraRep [28] HOPE [113]	- capture local and global structure	- high time and memory cost, not scalable - do not capture the content
Random Walk	DeepWalk [116] node2vec [57] struct2vec [124]	- relatively efficient - capture local structure	- do not capture global structure - do not capture the content
Deep Learning	GCN [72] DNGR [27] SDNE [154] DeepGL [128] GraphSAGE [61]	- capture non-linearity in features - capture local and global structure - capture the content	- high time cost

Table 2.2 – Comparison of graph encoding techniques

To summarize, factorization-based techniques only capture the structure of the graph. Their scalability is a major bottleneck because carrying out factorization on a matrix with millions of

rows and columns is memory intensive and computationally expensive. Random walk approaches are relatively efficient but again they do not capture the content and they only capture local structure. Deep learning methods can both preserve the structure and the content. They also have the ability to capture non-linearity, but their computational time cost is usually high.

Of all the methods seen previously, deep learning methods correspond most to our needs because they retain both the structure and the content of the graph. The efficiency problems of these methods can be reduced by using sampling strategies. In addition, as we will see in the section on anomaly detection, the deep learning methods can be directly used in the context of anomaly detection, reducing the total cost of pre-processing and processing. In our case, we have chosen for Sec2graph to use an edge-based encoding that preserves first and second-order proximity. The content is also preserved and our method is scalable up to billions of nodes by using sampling strategies.

2.1.2.3 Graphs analysis

The most common objectives for graph analysis are community or cluster identification, node classification, link prediction and root cause analysis. In the remainder of this section, we present pieces of work related to each of these objectives. Note that the work presented in this section does not necessarily focus on security but is relevant to a number of application domains, ranging from social science to biology.

Community and cluster detection. A community is a set of nodes that are linked together by many edges. Accordingly, two communities are distinct if only few edges link the vertices of the two sets of nodes. A cluster is a set of nodes in which members are similar, i.e, they both share the same attributes values and have similar neighborhood structures. Accordingly, nodes of two distinct clusters have few similarities, both in term of attributes values and in term of neighborhood.

As many practical issues can be solved thanks to communities or clusters identification, especially in the context of social networks, revealing the underlying community structure of complex graphs has become a crucial and interdisciplinary topic. In [49], the authors review community detection techniques in graphs. They state that the main purpose of community algorithms is to try to infer properties and relations between vertices, which are not available from direct observation measurements.

Examples of graph community detection for cybersecurity are presented in [40] and [115]. Ding *et al.* [40] define a network intrusion as an attempt to enter in a community of hosts to which one host does not belong. This suggests that in a network, intrusion attempts may be detected by looking for communications that does not respect community boundaries. Based on this principle, the authors build an intrusion detection mechanisms that looks for flows that do not respect those communities. Their results suggest that community-based methods are useful for network intrusion detection system.

Hercule [115] represents inter-log similarities within a graph of log events. In this representation, a node represents a log event and an edge represents a predefined similarity relation between two logs events. Community techniques are then applied on the graph to identify the set of events related to a given attack. The identification of information related to a known attack occurrence is performed thanks to an indicator of compromise. Experiments on APT attacks show that the system performs well in this task (accuracy of 88% on average), and once the events related to an attack have been identified, it allows a forensic analysis on it.

Examples of graph clustering are presented in [51] and [140]. BotTrack [51] creates a dependency graph between hosts to identify malicious network connections generated by peer-to-peer botnets. A clustering technique is then used to identify the network traffic containing a given order placed by a master to a collection of zombies. Clustering is also used in a second step to identify the attacks launched by the zombies. BotTrack uses for this a clustering method called DBscan in conjunction with the Pagerank algorithm to detect bots. This method is able to detect stealthy botnets using peer-to-peer communication infrastructures and not exhibiting large volumes of traffic.

Studiawan *et al.* [140] propose a method to detect an anomaly in the access control logs of an operating system based on a clustering method. The logs are first pre-processed and then clustered using an enhanced version of the MajorClust algorithm. A score that takes into account factors such as the total number of cluster members, the frequency of events in the log file, and the waiting time between specific activities is then presented to the security analyst.

Node classification. In most cases, node classification is realized through a semi-supervised training, where labels are only available for a small proportion of nodes. The goal is to label the complete graph based only on a small training data set. Common applications of semi-supervised node classification include classifying proteins according to their biological function and classifying documents or individuals into different categories/communities.

Hamilton *et al.* [60] introduced the task of classifying nodes in an inductive manner, i.e. where the goal is to classify nodes that were not seen during training, for example by classifying new documents in evolving information graphs.

Many security and privacy issues can also be modeled as a graph classification problem, where the nodes of the graph are simultaneously classified by collective classification [145, 138, 91]. State-of-the-art classification methods for this type of graph-based security and privacy analysis follow the following paradigm: assign weights to the edges of the graph, iteratively propagate the reputation scores of the nodes in the weighted graph, and use the final reputation scores to classify the nodes in the graph.

Aesop [145] and Marmite [138] use this technique to propagate known malicious reputation of some file to unknown ones and thus detect potential malware. Aesop builds an homogeneous graph with nodes representing files and edges representing the presence of two files on the same host. The label propagation is performed by the belief-propagation algorithm. The graph in Marmite is an heterogeneous graph made up of file, fqdn, url and ip nodes that represents how files are downloaded (by which hosts and from which servers). The label propagation is performed by the

Bayesian-Label propagation algorithm.

Li *et al.* [91] performed a study on the topological relations among hosts in malicious Web infrastructures. They discovered that these hosts are often connected to other malicious hosts and do not receive traffic from legitimate sites. The authors developed a graph-based approach that relies on a small set of known malicious hosts as seeds to detect dedicated malicious hosts in a large scale.

While classification is often applied on nodes, two variants exist, namely edge classification and subgraphs classification. For example, Grover *et al.* [57] explain how feature representations of individual nodes can be extended to pairs of nodes (i.e. edges) for edge classification. Shervashidze [133] proposes a graph classification method based on the Weisfeiler-Lehman test of isomorphism on graphs.

Link prediction. The goal of link prediction is to predict missing edges, or edges that are likely to appear in the future. Link prediction is at the core of recommendation systems, including predicting missing friendship links in social networks or affinities between users and movies. More generally, link prediction is closely related to statistical relational learning [78], where a common task is to predict missing relations between entities in a knowledge graph.

Heard *et al.* [62] propose to model the normal traffic behavior of an IP address for anomaly detection. Normal network traffic is made of automated and human behaviors. The difficulty is to separate regular automated tasks permitted by the client from malware sending regular communications to a controller or trying to scan the internal network. The article presents a Fourier analysis technique to predict communications (edges) between two hosts and detect anomalies (unpredicted edges).

Metelli *et al.* [103] propose a Bayesian model and anomaly detection method for characterising graph structure and modelling likely-new edge formation. Arrivals of new edges in the graph represent connections between a pair of client and server not previously observed that in some cases might suggest the presence of intruders or malwares.

Root cause analysis. A graph structure allows to link nodes indirectly (see n-order Proximity definition in Section 2.1.2.1). Thus, depending on the semantics of the links, the exploration of the neighborhood of a node of interest allows to find dependency or cause and effect relations between nodes indirectly related. When a graph represents temporal events, it is also possible to find sequences of strongly linked events. In the security field, this consists in finding the source of an attack (root cause) and each steps related to it (attack graph).

King and Chen [71] propose to reconstruct a chain of events in a dependency graph to perform intrusion analysis. Starting from a single point of detection (for example, a suspicious file), their tool, BackTracker, identifies files and processes that may have affected that point of detection and displays the event chains in a dependency graph. Similarly, Hossein *e*t al. [65] use sequences of system calls to build attack graphs and detect the root cause and the attack steps. Milajerdi *e*t al. [104] use audit logs to reconstruct the history of attacks using traces from common Advanced Persistent Threat attacks. Kobayashi *e*t al. [76] use `syslog` events to infer causality between

security system events. These proposals are however limited since they only consider one type of event format. This contrasts with [159] in which the authors propose to discover causal dependency in heterogeneous events to detect multi-step attacks.

Conclusion on graphs analysis. In this section, we showed that graph analysis techniques have very frequently been applied in the analysis of graphs representing security data. Community detection, node classification, link prediction and graph traversals have successfully been used to detect attacks or malicious hosts/files. In this thesis, we focus on two graph analysis techniques which are community detection and node classification. Community detection will allow us to detect all events related to the same attack. For the classification of nodes, we actually use one of its variants, namely the edge classification to detect anomalies in the edges of our graph. The detection of anomalies is detailed in the next section.

2.2 Anomaly detection

Intrusion detection techniques are classically distinguished between approaches based on the knowledge of attacks (misuse detection) and approaches based on the knowledge of the normal and expected behavior of the system (anomaly detection). Both have advantages and disadvantages [38].

In this thesis, we focus on the detection of anomalies. Anomaly detection consists in identifying previously unknown patterns in data. Indeed, we are particularly trying to identify new forms of attacks (zero days) while coping with the fact that attackers often evade detection through subtle modifications of their tools and *modus operandi*.

While many approaches have been proposed for this purpose, we rely in this thesis on machine learning. As we saw in section 2.1.2.2, encoding of input data is an important step before its processing in this context.

In this section, we first present the existing approaches for data encoding (features engineering). Then, we present approaches for anomaly detection using machine learning (including statistical methods).

2.2.1 Features engineering

The input data of a machine learning algorithm is generally structured through a given number of features (i.e., attributes). Machine learning algorithms require features with specific characteristics to work properly. Hence, there is a need for a number of pre-processing steps applied on data: selection of relevant part of data, normalization, cleaning, outlier identification, etc. This serie of steps in refereed to as “feature engineering”.

Apruzzesse *et al.* [7] show that features engineering is an important task in cyber-security by evaluating the performance of an intrusion detection classifier when trained with different features and different data sets. In fact, the main difficulty in machine learning is that network traffic cannot be used directly as an input. It is necessary to build feature vectors constructed from

it. Thus, feature engineering is an essential task to efficiently process data. According to [102], feature engineering is more complicated when dealing with network monitoring (the domain of this thesis), because of the diversity of data types in raw data packets and derived data fields. In addition, the very specific nature of the data requires a high expertise in security in order to properly choose the features while minimizing any potential loss of information.

Features engineering can be split in two processes: features selection and features processing. In the remaining of this section, we first review features selection in studies relative to security. Then, we review features processing.

2.2.1.1 Features selection

In the following, we describe some commonly used techniques to select and represent security features for network analysis. That is why we only deal here with techniques related to the selection of network data attributes. We identify two tendencies of data pre-processing for anomaly based network intrusion detection system, namely network packet header analysis and payload analysis.

Onut *et al.* [110] identify three kinds of packet header analysis techniques depending on the type of network features being used: basic features derived directly from packet headers, single connection-derived features and multiple connection-derived features.

Basic feature selection is used by NIDS such as PHAD [97] and SPADE [137] that use IP or UDP/TCP headers fields as inputs. PHAD is thus able to detect probe and DoS, while SPADE is able to detect network scans or port scans.

Single connection features are time-based statistical measures, such as session duration or the number of network packets exchanged during a connection. They are useful to detect unusual data flows. Early *et al.* [42] detect when a protocol is not used as intended, or in a way that significantly deviates from the normal behavior using features extracted from flow session. Similarly, Estevez *et al.* [46] wants to detect anomalies in the use of protocols in computer networks with a quantification of the TCP header space and the use of a Markov chain. Quantification is the process of constraining an input from a continuous set of values (such as real numbers) to a discrete set (such as integers). The normal use of the protocol is then modeled using a Markov chain, using sequences of observations as inputs.

Multiple connection-derived features are constructed by monitoring basic features over multiple flows or connections. This technique is used for example in [80]. Lakhina *et al.* use entropy to detect anomalies spanning multiple connections. Entropy captures changes in the distribution of traffic characteristics in a single value, and the observation of entropy time series over multiple characteristics reveals unusual traffic behaviors.

Many attacks consist in inserting exploit code in the payload of network packets. It is therefore interesting to look for anomalies in the payload. PAYL [155] builds a byte-frequency distribution model of network traffic payloads. The result is a feature vector containing the relative frequency count of each of the 256 possible bytes in the payload. To exploit this type of feature, PAYL uses the Mahalanobis distance to compare the supervised traffic to a model previously learned and

raises an anomaly if the distance is too high. This method is applied to detect zero-day worms which can produce an unusual byte-frequency distribution.

In [75], the authors use features crafted by security experts like string length histograms, string entropy and flags indicating the existence of special characters in strings.

Due to the complexity of payload analysis, many techniques focus on small subsets of the payload, like the HTTP request. Kruegel *et al.* [79] constructs six features based on the HTTP queries containing attributes. These six features are based on attribute length, attribute character distribution, structural inference, token finder and attribute presence or absence.

Zeek (formerly Bro) [114] uses packet parsers to rebuild the traffic embedded in the IP/ICMP packets, to allow selection of specific features of this embedded traffic (for example, an HTTP request).

Maxwell *et al.* [102] present several feature engineering techniques for machine learning-based classifiers. The features selected by these various techniques are used as input for the same classifier. The main result of this study is that no automatic feature selection technique can achieve better classification results. One conclusion to be drawn from this is that human expertise is actually required for the features selection task. This is also what [141] shows when presenting a comparison of the features selection methods used in nineteen studies related to security.

In this thesis, we used the ability of the Zeek NIDS decoder to parse and extract information contained both in the header and in the payload. For example, Zeek can extract the TCP flags but also the MIME type, the name or the hash of a transferred file or the URL requested in an HTTP transaction. That way, we do not loose information from the payload and are still able to detect anomalies from the header if the network traffic is encrypted.

2.2.1.2 Features processing

Once a set of basic set of features is built, these features must be prepared to be handled by the machine learning algorithms. Some feature processing techniques consider each feature individually and heavily depends on the type of the feature. Three kinds of data are to be considered: numerical ones, categorical ones and one type that is specific to the domain of network security, IP addresses. Other feature processing methods consider all features in order to select only relevant ones for the data analysis step.

Numerical data. In [82, 26, 89], the authors used a technique called Z-score standardization for numerical values such as the size of a network packet. This method consists in computing the standard deviations. It ensures that all numeric attributes in a dataset have the same scale, thus preventing one attribute from being erroneously more prevalent than another. Another method consists in using clustering algorithms to group numerical values in clusters. Gaussian Mixture Model are a well known method to cluster numerical data [150]. A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite

number of Gaussian distributions with unknown parameters. The method consists in assigning each numerical value to one Gaussian to split the data into different groups. Categorical data pre-processing is then performed. We use this last technique in this thesis as it allows to consider all features as a category.

Categorical data. In [82], the authors use the method of one-hot encoding for categorical values. This method consists in creating one feature for each different value that was seen. One limitation of this method is that it can result in very large features vectors. To solve this problem, the authors of [82] and [26] select only the most frequent values to create the features vector. This practice however heavily relies on the samples that are used. Hernandez *et al.* [64] compare four engineering techniques. The first one, indicator variables method (e.g. one-hot encoding) uses a binary feature for each value of the feature. Such data conversion results in a dataset which dimensionality strongly depends on the numbers of values representing the features and does not depend on the number of classes. The second one, the conditional probability method replaces each observed value with an array of the conditional probabilities of obtaining each class given the fact that the attribute has a particular value. The third method is based on a decision tree system that converts each value of a feature into a real number. These methods are compared with the most commonly used mapping which simply replaces values with subsequent natural numbers (e.g. label encoding). The experiment in the paper shows that the accuracy obtained for the classification of attack types is improved for both methods specially for Probe and DOS classes. However, these methods were only tested against classification task whereas our task is to detect anomalies with unsupervised methods. As will be shown later, we have chosen the one-hot encoding method which is simple and allows to quickly interpret the results by retrieving the original values.

IP adresses. IP addresses are somewhat peculiar data midway between numerical and categorical data. Indeed, two "close" IP addresses can belong to the same subnet. However, comparing two IP addresses not belonging to the same subnet does not make sense. The challenge is to encode the fact that two adresses might belong to the same subnet without prior knowledge. The authors of [33] were interested in this problem and compared three techniques, a 32-bit vector representation, a 4-Bytes representation and an extended-Byte representation (method where cross-octet information is incorporated). The last method was the best in the task of classifying malicious IP addresses. As will be shown later, and since our task is to detect anomalies by learning the normal behavior of the internal devices, we focused during this thesis on the most-used internal IP address and encodes them with the one-hot encoding technique.

Features selection In [135], the authors use four different methods to deal with feature selection and encoding: leave-out single-value attributes, namespace correlation, data correlation (Pearson [52]), and feature selection. The first method, *leave-out single-value*, consists simply in removing all single-valued attributes from the set of selected features. The second method is a *namespace correlation*: some entity attributes may refer to the same piece of information although

their names are different. The method simply consists in removind duplicates. Similarly, highly correlated attributes are removed with the use of the third method, *data correlation*. The detection of these correlations is carried out using Pearson's techniques. Finally, the authors use a combination of random forest, neural network and Bayes classifier to select the relevant features with the fourth method *feature selection*.

There is no consensus about the best way to select and then process and encode the features extracted from security events. Some authors have been able to demonstrate the relevance of some particular attributes to detect specific attacks with a supervised approach and have achieved good results for specific datasets.

However there is little proof that the approach is transferable to other datasets. Moreover, our objective being to propose an unsupervised method, we decided to encode the results according to two criteria: scalability, and the ability to be processed by anomaly detection algorithms. As the information processed is heterogeneous, we have chosen a binary encoding that is suitable for the different types of information encountered.

2.2.2 Anomaly detection using machine learning

The term *anomaly* has several definitions. Generally speaking, Barnett and Lewis [118] define an anomaly as an "observation (or a sub-set of observations) which appears to be inconsistent with the remainder of that set of data". In the security field, the NIST defined anomaly-based detection as "the process of comparing definitions of what activity is considered normal against observed events to identify significant deviation" [130]. Anomaly-based techniques, also known as profile-based or anomaly detection, are based on the creation of a baseline profile representing normal/expected behavior, and on the fact that any observed deviation of current activity from this profile is considered an anomaly. We distinguish here four main families of anomaly detection methods, namely statistics-based, clustering-based, classification-based and deep learning-based methods. Notice that some pieces of work combine techniques from two families to build an hybrid approach. We present below these four families, as well as a few hybrid approaches.

2.2.2.1 Statistical methods

Statistical methods are widely used for anomaly detection. From a very general point of view, the principle consists in building a model of the behavior of the monitored system from statistics on data generated by this system.

Many different approaches have been used. We present here two kinds of methods. The first one is based on the detection of the exceeding of a determined threshold. The second one is based on data compression. We propose two examples for this last method, the wavelet analysis and the principal component analysis.

Anomalies are related to abrupt changes in the data. Most often, thresholds are defined for certain observed features. If a threshold is exceeded, a change is considered to have occurred. As we have seen in section 2.2.1.1, Wang *et al.* [155] propose a statistics-based method based on

the computing of the Mahalanobis distance over a feature vector containing the relative frequency count of 256 possible bytes. In details, the Mahalonobis distance is used to determine the similarity between a series of known and unknown data. The authors use it during the detection phase to calculate the similarity of new data against a pre-computed profile. The detector compares this measure against a threshold and generates an alert when the computed distance exceeds this threshold.

The main idea of compression methods for anomaly detection is to use a projection space of smaller dimension and then reconstruct the data in the original space. The difference between the input and output data, called the reconstruction error, is then used to identify the anomalies in a dataset. Wavelet analysis consists in modeling non-stationary dataseries. Such data series may contain signals that can vary in amplitude and frequency. Data series are modeled using wavelets, which are powerful basis functions localized in time and frequency. Hamdi and Boudriga [59] propose an anomaly detection method using wavelets. It relies on identifying attack-related anomalies by differentiating between dangerous and non-threatening anomalies. This task was achieved based on the concept of period observation, where wavelet theory was used to decompose one-dimensional signals in order to analyze both their spectral frequencies and time localization. Principal Component Analysis (PCA) is a method to convert a set of observations to a set of linearly uncorrelated variables called principal components. In the converted principal components, the largest possible variance is held by the first component and each subsequent component holds the following highest variance. The PCA is often used as a tool to analyze data and make predictive models. Lakhina *et al.* [81] were the first to explore this method in the security field. They addressed the anomaly diagnosis problem in network wide-traffic by using PCA to separate network connections into normal and anomalous subspaces. The main idea was that PCA results in a reduced subspace of k variables (principal components) which corresponds to normal network traffic behavior, while the remaining subspace consists in anomalies or noise. Then, every new traffic measurement is projected onto these subspaces so that different thresholds can be set to classify these measurements as normal or anomalous. Their work was responsible for the massive attention on PCA-based approaches for anomaly detection received in the last decade.

2.2.2.2 Clustering methods

Clustering analysis aims at identifying groups of similar entities. These groups are also called classes or clusters. Entities belonging to one cluster are similar by construction while entities from two different clusters are dissimilar. Clustering techniques can be used for outlier detection that consists in identifying values which are too far away from any cluster. Two main methods are used in cluster analysis, KNN that is supervised, and K-means that is unsupervised.

The K-Nearest-Neighbor (KNN) method consists in defining a membership class for each observation. An observation is classified according to the majority result of the membership class statistics of its k closest neighbors. Liao *et al.* [93] uses KNN to classify the behavior of a program as normal or intrusive. The program behavior is represented by the frequencies of system calls.

Each program is then classified using the KNN classifier.

2.2.2.3 Classification-based methods

Classification is widely used in the anomaly detection field. The main idea can be summarized in two steps. First, during a training phase, a classifier is built using labeled training data. Then, during a testing phase, this classifier is used to classify any new instance of data.

According to the number of labels in the training dataset, classification-based anomaly detection techniques can be either one-class (actually there are two classes: normal data and abnormal data) or multi-class. In the first case, the abnormal data will be considered as an attack. In the second case any new traffic classified in one of the abnormal classes will be considered as an attack.

Here again, numerous approaches have been proposed in the literature. We present here the most-frequently used ones.

Naïve Bayes classifier. Naïve Bayes classifiers are a simple probabilistic classifiers commonly used for network intrusion detection problems. It is based on the assumption that all of the features of an observation independently contribute to the probability that this observation belongs to a particular class. This classifier computes the probability of a certain instance belonging to a singular class and keep the class with the highest probability.

Klassen *et al.* [74] proposed a Naïve Bayesian approach to detect in real time black holes, selective forwarding and DDoS attacks. The system monitors packets sent from nodes to detect any abnormality. The classifier first assumes that data are normally distributed. Then, the probability of a sample belonging to a class is calculated by a normal distribution probability procedure.

Tao *et al.* [94] also use a Naïve Bayesian approach. They combine it with a time slicing function and exploit the relation between time and network traffic. Their research is based on the hypothesis that network traffic changes at distinct times and that some traffic does not occur at a particular time.

Support Vector Machine. SVM is an efficient tool widely used for multiclass classification. It is a classifier based on finding a separation hyperplane in the feature space between two classes so that the distance between the hyperplane and the nearest data points of each class is maximized. However, the number of dimensions considered affects the performance of SVM-based IDS. Another issue is that SVM treats different features of data equally while many features are redundant or less important. SVM is also computationally expensive. In order to mitigate this problem, dimension reduction techniques are often applied to extract important features. For example, [90] proposed a SVM-based intrusion detection system based on a hierarchical clustering algorithm to preprocess the dataset before SVM training. This reduces the number of features to be considered by the SVM classifier.

Decision tree. A decision tree is a decision support tool representing a set of choices in the graphical form of a tree. The various possible decisions are located at the end of the branches

(the "leaves" of the tree), and are reached based on decisions made at each step. Random forest is a classification method used for classification or regression tasks. It is considered a learning method in which the decision tree is taken as the basic classifier. The random forest comprises a set of classifiers made of trees, each tree being randomly constructed, independent and identically distributed. Each tree in the set results in a vote for the most popular input vector class. Zhang and Zulkernine [165] proposed an anomaly-based approach using a Random Forest classifier. The model is trained to classify flows into the network service classes HTTP, FTP, POP, SMTP, and telnet. The method considers a flow that is wrongly classified as an outlier. That can also be the case if the value of an anomaly function is greater than a threshold.

2.2.2.4 Deep learning

Artificial Neural Networks (ANNs) are machine learning algorithms inspired by the central nervous system. They are made of layers of neurons connected by adaptive weights. An ANN with one hidden layer is able to produce any non-linear continuous founction. A Deep Neural Network (DNN) is an ANN with a number of layers superior to three. The goal of a DNN is to learn more complex functions. In this stacked format, the lower layers learn easy features while the upper layers analyze the output of previous layers and learn missing, difficult, or hidden features. Well-known types of deep neural network are Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN) and AutoEncoders (AE).

A recurrent neural network (RNN) extends the capabilities of a traditional neural network, which can only take fixed-length data inputs, to handle input sequences of variable lengths. The RNN processes inputs one element at a time, using the output of the hidden units as an additional input for the next element. Yin *et al.* [162] attempt to integrate an RNN in an IDS for supervised classification learning (RNN-IDS). They study the performance of the model in binary classification and multiclass classification, and they find that the RNN-IDS model gets a better accuracy compared to methods such as SVM, Random Forest or Naive Bayes. The long-short-term memory (LSTM) units are introduced to enable RNNs to manage problems that require long-term memories. LSTM units contain a structure called a memory cell that accumulates information. Kim *et al.* [70] apply this technique to the KDD Cup 1999 dataset.

A convolutional neural network (CNN) is a neural network meant to process input stored in arrays. Input example is an image, which is a two-dimensional array of pixels. Wang *et al.* [156] use a CNN for malware traffic classification by taking traffic data as images. This method needs no hand-designed features but directly takes raw traffic as an input of the classifier. Raw traffic data in pcap format is processed to be transformed into CNN input data following the four following steps: traffic split, traffic clearing, image generation and IDX conversion.

Finally, autoencoders are a class of unsupervised neural networks in which the neural network takes a vector as an input and tries to match the output to that same vector. By taking the input, changing its dimensionality, and reconstructing the input, one can create a higher or lower dimensionality representation of the data. These types of neural networks learn compressed data encoding in an unsupervised manner. Additionally, they can be trained one layer at a time,

reducing the computational resources required to build an effective model. To our knowledge, only [11] and [105] proposed a fully-unsupervised approach. The authors of [11] use two types of auto-encoders (namely stochastic denoising auto-encoder and deep auto-encoder) to detect anomalies in the NSL-KDD dataset. The authors of [105] propose to remedy the problems of data with little or no security label by proposing an unsupervised and a semi-supervised approach. The idea is to use an auto-encoder in association with a classification algorithm for the semi-supervised approach. The latter is then trained on a restricted portion of labeled data. In the unsupervised approach, the auto-encoder is used alone. The study was carried out on the NSL-KDD and CICIDS2017 datasets. The results are good only in the semi-supervised approach, even if the unsupervised approach seems to isolate some attacks.

2.2.2.5 Combination of several approaches

The examples seen above use only one technique to detect anomalies. However, some research focus on coupling different techniques to improve the performance of their intrusion detection system. The most common approach is to use an unsupervised method to sort the data first, followed by a supervised method to reduce the number of false positives. This approach makes it possible to take advantage of the best of both methods: on one hand, there is no need to label a large set of data, and on the other hand, the number of false positives is reduced.

In [151] the authors present an analyst-in-the-loop security system, where the analyst expertise is put together with state-of-the-art machine learning to build an end-to-end active learning system. The system has four key features: a behavioral analytics platform, an ensemble of outlier detection methods, a mechanism to obtain feedback from security analysts, and a supervised learning module. The outlier detection methods are a matrix decomposition-based outlier detector, an autoencoder and a density-based outlier detection. An outlier score combining the three scores obtained from the unsupervised algorithm is then computed. The supervised learning module is based on a random forest.

DeepLog [41] uses deep learning (LSTM) and classic mining (density clustering) approaches to detect anomalies in system logs. In addition, the authors propose a process to incrementally update the DeepLog model in an online fashion so that it can be adapted to new log patterns over time. Furthermore, DeepLog constructs its model in an online fashion so that it can adapt to new log patterns over time. Finally, users can diagnose the detected anomaly and perform root cause analysis.

Two studies [3, 122] add a supervised layer to the unsupervised output of the autoencoder. The general idea is to use the auto-encoder to identify normal traffic. Traffic that is not considered normal by the auto-encoder is then provided to a supervised classification device, trained on data labeled to identify attacks.

Finally, Lee and Park [84] propose AE-CGAN (autoencoder-conditional GAN), an intrusion detection system based on autoencoders and random forest for situations in which there are significant imbalances between normal and abnormal traffic. Based on the unsupervised learning models autoencoder (AE) and the generative adversarial networks (GAN) model they propose to

oversample rare classes based on the GAN model in order to solve the performance degradation caused by data imbalance after processing the characteristics of the data to a lower level using the autoencoder model.

In this section, we presented the state of the art of machine learning applied to anomaly detection. The latest studies show that algorithms based on deep learning are generally more accurate in detecting anomalies than more traditional machine learning methods as they can learn more complex models. However, they often require more computation and are less interpretable. There is also no real consensus on the use of supervised or unsupervised methods, some using both methods at the same time. In our work, we choose to use a deep autoencoder. This method does not need labelled data but a particular attention must be taken to tune some hyper-parameters. Moreover, as our input data are graph-structured data, the pre-processing steps are very important.

2.3 Graph visualization for security

While detection algorithms play an important role in intrusion detection, human expertise remains a fundamental element in incident response and in the analysis of alerts. Indeed, completely eliminating false positives seems impossible, and even if it was, the analyst would still need to understand the events to propose an appropriate response to the incident. This is why it is important to help analysts in their decision making by proposing relevant visualizations. Numerous proposals have been made to visualize either raw data (pcap files for instance) or pre-computed data (alerts and result from automatic analysis). Also, those visualizations can be used either to simply display the information at hand in an efficient manner (reporting) or to allow the analyst to interact with it to go deeper in the analysis (visual analytics).

As this work focuses on network security events, the scope of this thesis is limited to the visualization of network communications or alerts. We also focus on representations that allow to compare and correlate various events to detect malicious intents.

To that end, visualizing relations in security data is common. A graph structure is well suited to encode logical links between various types of events. As a consequence, and with notable exceptions such as [35, 66, 158], link graphs are often used for visualizing relations among similar entities (e.g. network nodes or users) or various types of entities (e.g., the various features of an alert coming from an IDS or from a pcap file).

In this section, we first present the pieces of work on visualization of homogeneous graphs, i.e. graphs for which all nodes correspond to the same type of entities. We then present the pieces of work representing heterogeneous graphs.

2.3.1 Visualizing homogeneous graphs

A graph with a single type of node and a single type of edge is called homogeneous. In network security, this type of graphs are often used to represent the communication between host machines. In this section, we introduce some homogeneous graph visualization specifically design to highlight attacks on network systems.

Percival [5] allows the security analyst to understand the security status of the network and to monitor security events occurring on the system. The proposed visualization allows to follow the actual progress of the attack according to the remediation measures executed. It also provides an overview of the possible evolutions of the attacks. The network topology view is typical of an homogeneous graph visualization: each node represents a device of the network and each edge is a step of an attack path between two network devices. As such, Percival is mainly a report tool that allows the analysts to assess the risks on his/her network. It evolved to MAD [4]. Compared to Percival, a color code has been added to provide information about the attributes of the nodes and the edges. While the represented graph is still homogeneous, this addition provides visual analytics capabilities (see Figure 2.3).

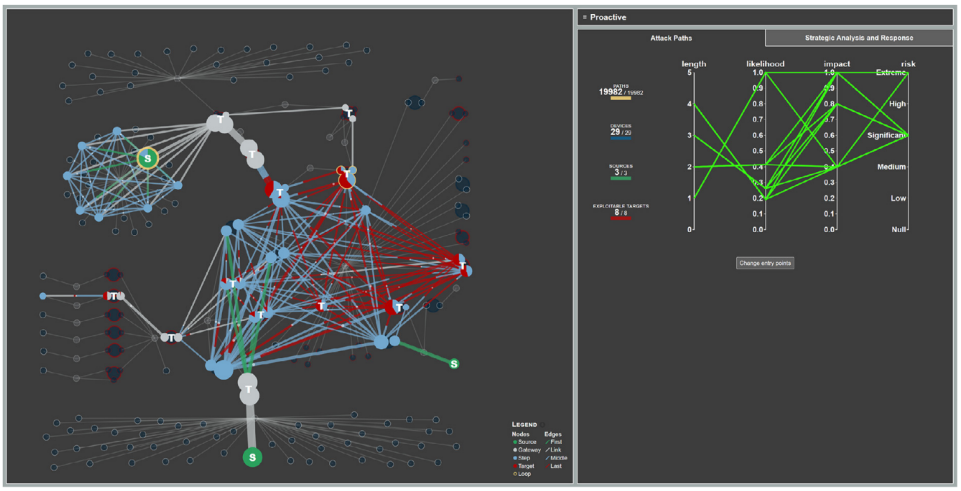

Figure 2.3 – An example of the MAD visual analytics prototype GUI

In [8], Arendt *et al.* describe the user-centric design and development of a decision support visualization for active network defense. The tool, named Ocelot, helps cyberanalysts in assessing threats about computers affected by a malware and quarantined healthy parts of a network. The web-based visualization prototype integrates and visualizes multiple data sources through the use of a hybrid space partitioning tree and node link diagram. A screenshot of the application is shown in Figure 2.4.

VisAlert [95] is designed specifically for understanding the nature, time and location of events. At the center of this visualization is a graph in which nodes are network devices. They can be organized either as a map indicating a geographical location or a a graph indicating the topology of a network. Around this central visualization, a serie of concentric rings corresponds to time windows of events. The rings are also equally divided into parts that show the types of events, such as alerts from intrusion detection systems for example.

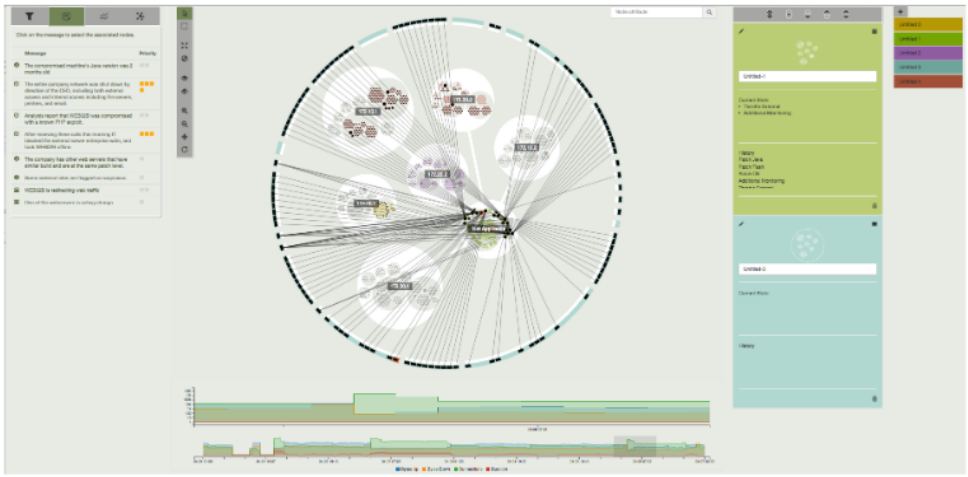

Figure 2.4 – An example of the Ocelot visual analytics prototype GUI

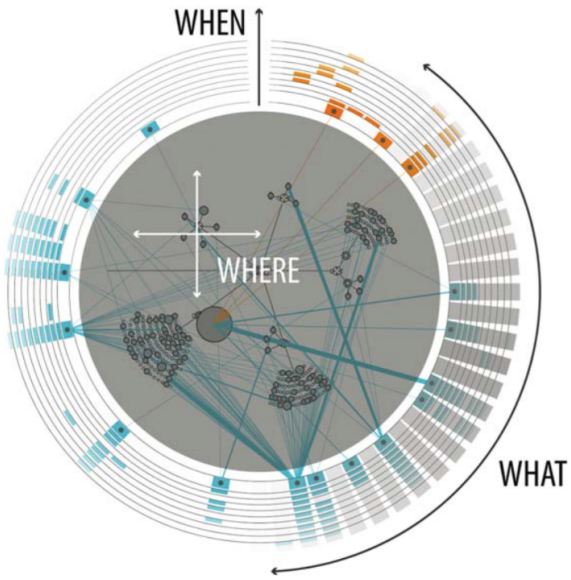

Figure 2.5 – The Vizalert alerts are mapped onto a colored slice of a ring corresponding to its type, from oldest to youngest. To show where the alert occurred, the space inside the ring houses a spatial visualization to which each alert is linked

2.3.2 Visualizing heterogeneous graphs

A graph with two or more types of node and/or two or more types of edge is called heterogeneous. In security, heterogeneous graphs are used to display high dimensional types of data or complex systems. In this section, we introduce heterogeneous graph display designed to help a security analyst to get an overview of the security in complex networks.

In [148], a visual analytics tool provides a simplified representation of the most important

elements of a security data set and their relations. The visualization technique is designed to address two common shortcomings of existing graph visualization techniques: scalability of visualization and comprehensibility of results. The main goal of this visual analytics tool is to provide security analysts with an effective way to reason interactively about various attacks. The tool was tested to analyze spam campaigns, with the ultimate goal of getting a better comprehension of the root causes behind the structure and the organization of those attacks. First, a force-directed algorithms is used to visualize clusters of security events, allowing the security analyst to easily inspect the relation and shared connections among a group of events. A visual cluster is created either when different security events share many (low weight) connections to the same set of nodes representing feature values or when they have large weight edges to a small number of nodes. Then, the tools replace nodes and edges with a more abstract graph visualization and create a network visualization that require less screen real estate. It makes easier to understand in the global context of the network and the role of each node. This visual representation of security events still conveys two pieces of information to the security analyst: the volume of security events associated with a single feature value and the volume of security events that are linked to two or more feature values.

In [25], the authors propose a visualization that allows security experts to analyze and enrich semi-structured intelligence information on cyber threats. They demonstrate the feasibility of their concept through the visualization of data described using the STIX model. They create an interactive visualization that allows security experts to extract knowledge and information from incident documentation. They also enable the exchange of explicit and implicit knowledge between security experts. To visualize the data, the authors use a force-directed graph scheme. They also provide interactive features that allow security experts to adjust the presentation themselves, addressing the problem of node-link graphs consisting in their limited extensibility in terms of large numbers of strongly connected nodes.

Figure 2.6 – KAVAS [25], a Knowledge-Assisted Visual Analytics concept for the Structured Threat Information eXpression

Cygraph [108] is a tool to maintain situational awareness in the presence of cyberattacks and to focus on protecting critical assets. It provides a graph knowledge base about vulnerability, threat indicators, and mission dependencies within a network environment. Cygraph provides various modes of interaction with its visualization interface, including graph queries, expand-

ing/pivoting/filtering queries, visual synthesis, and dynamic graph evolution over time. CyGraph also supports visual grouping of graph nodes, based on property values or by manual selection. It integrates multiple graph layers (network infrastructure, cyber defensive posture and threats, ..) that define subsets of the overall model space, with relationships within and across each graph layer. CyGraph is mainly based on attack graphs and targets mainly traditional network systems.

APT-Hunter [134] is a visualization tool that helps security analysts to explore login data for discovering patterns and detecting malicious logins. It also helps security analysts to integrate their knowledge about the organization and the network to discover malicious login events in an enterprise network. It is based on the observation that a characteristic pattern of an attacker who stole credentials deviates from benign patterns and can be used to detect malicious logins. Using APT-Hunter, security analysts can iteratively discover suspicious and benign login patterns with the help of an interactive node-link visualization tool. The three objectives that drive the design of the platform are: enhancing the recognition of login patterns, enabling the expression and matching of login patterns, and enabling the selection and filtering of login events. The APT-Hunter user interface consists of five panels: Search, Filter, Alert, Details, Visualizer. The Visualizer is the most important panel. It represents the login data using a node-link diagram. In this visualization, nodes represent computers and links show login events from the sources to the destinations. Users can interact with nodes to locate them on the screen or to see more details about them. Some details about computers and logins are encoded using icons and color of the nodes and links. Icons of the nodes show the type and role (e.g., web-server, domain controller) of the computers. The color of the link shows if it is a suspicious login. In addition, geolocation of the nodes is shown by the placement of all the computers in the same location next to each other and grouped by a colored canvas. The visualizer enables the recognition of login patterns and abnormalities. A screenshot of the panel is shown is Figure 2.7.

An important challenge in generating link graphs is defining the layout [152], i.e., the position of the nodes and the connection to be drawn for each edge. One of the criteria for a well laid-out graph is that edges must not overlap, since overlapping edges make it hard to read a graph. Another criteria is the number of nodes that can be visualized simultaneously. Too many nodes will make the graph illegible. Finally, the interpretability of the results is the third important criteria.

The three types of visualization presented above all use a force-based graph layout. To deal with the scalability and readability problem, they use two techniques: they either let the user filter interactively the information to be displayed [25, 134] or they aggregate the displayed nodes [148]. [25] and [134] use different types of icons and colors to identify different type of nodes. Finally, in [148], the size of the aggregation node represents the number of aggregated nodes.

Parallel coordinates are used for plotting multivariate data. They are very useful for comparing many variables and representing the relations between them. In parallel coordinates, each variable is given its own axis and all the axes are set in parallel. VisFlowConnect [163] (see Figure 2.8) uses this technique for NetFlow datasets visualization, aiming security situational awareness. The goal of this tool is to help a security analyst to detect and investigate anomalous traffic between a local network and external domains. Central to the design is a parallel coordinates view that

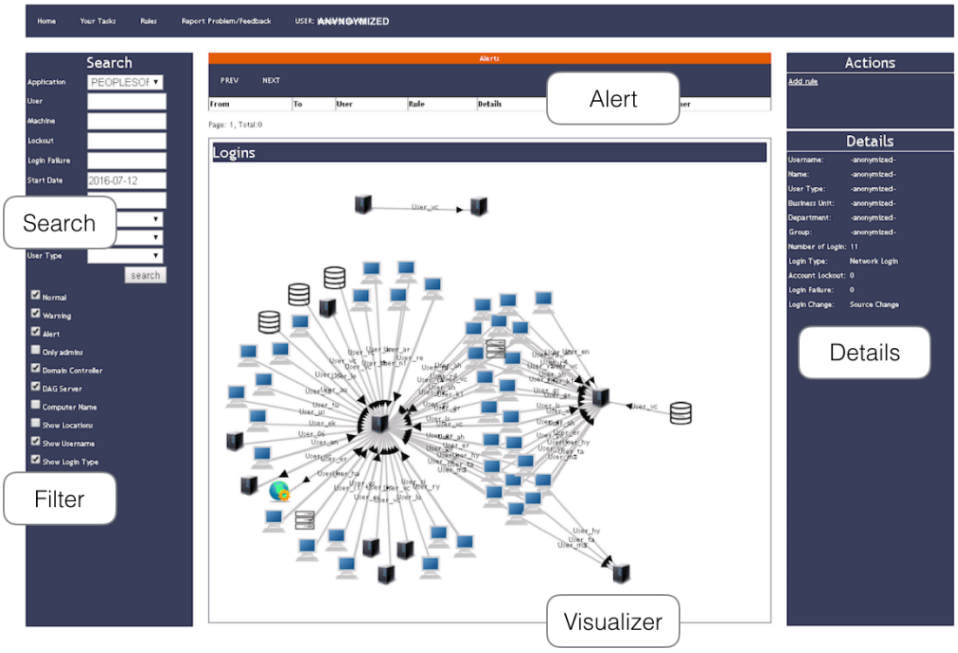

Figure 2.7 – An example of the AptHunter visual analytics prototype GUI

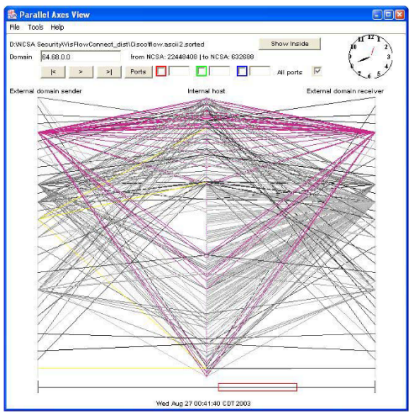

Figure 2.8 – In VisFlowConnect [163], parallel coordinates visualization shows Netflow records per source address, destination port and destination address.

displays NetFlow records as links between two machines or domains while employing a variety of visual cues to assist the user. VisFlowConnect also provides several filtering options that can be employed to hide uninteresting or innocuous traffic.

We have seen in this section that there are numerous way to visually represent security data and that the goal of this visualization is mostly to help the analyst to better understand the situation and to better find anomalies in the data by looking for anomalous patterns. As our model is based on a graph structured model, we focused on graph visualization panel. Previous

pieces of work show us that occlusion and scalability is often a problem with using graph, for which different proposals have been made.

Conclusion

In this state of the art, we first described the work on structuring security data in all its forms. While all authors seem to agree on the need to standardize data into a single model, there is currently no single model that prevails over the others, even if the STIX model, which enables to represent heterogeneous data in the form of object graphs, is particularly influential. Our proposal is to adapt this model to network event analysis. We describe it in Chapter 2.

Automatic analysis often requires a pre-processing step for the data features so that the data can be properly processed. This step, called feature engineering, requires technical expertise to properly choose the parameters to be taken into account for attack detection or to learn methods on the data. In addition, the graph approach requires special attention to allow both node attributes and structure to be retained in our input data.

We then explored the main trends in graph analysis methods and we saw how graphs are used in the security domain. Our contribution is the discovery of sub-graphs of interest representing attacks using community detection algorithms. We present this aspect in Chapter 3.

We then show the classical methods of anomaly detection. Chapter 4 shows our contribution on the detection of anomalies within a graph using an auto-encoder.

Finally, we present the work about allows visualizing network events and information and how they help security experts in their analysis. Then we specifically treated graph visualization techniques and detailed the constraints inherent to graphs to ensure scalability and interpretability of the represented data. Our contribution consists in the interactive and immersive representation of graphs representing network security events. We present it in Chapter 5.

Chapter 3

Security related data representation model

Contents

Introduction

Security monitoring produces a large number of network events. The diversity of communication protocols generates numerous log files with a wide variety of formats. In addition, these log files are often not explicitly linked to each other. It is therefore difficult to obtain an overall view of activities on the network.

In order to exploit this large volume of heterogeneous information, security analysts usually start from an indicator of compromise (IoC), i.e., an observable that suggests that a compromise may have already occurred. Examples of such IoC are a particular IP address or a particular file name found in some network event. Analysts look among all the available log files for any information related to this IoC that could help analyzing the security incident that led to this indicator. This approach is referred to as forensic analysis.

To help analysts in this forensic analysis, there is an important need for a representation of data able to highlight relations between events. In this chapter, we propose a new graph-based representation of network events based on Security Objects (SO). SOs are the nodes of

the generated graph and correspond to a subset of attributes coming from the various network events. Each attribute is related to a type of information that is important from a security point of view. The value of an attribute is derived from the value of a field found in a security event in the analyzed log files. As will be made clearer later in this chapter, a link between two SOs indicate that theses SOs have been found together in at least one event. By construction of the graph, security objects present in several events appears only once in the graph. This allows to create links between these events. The SO graph thus gives a unified and rich vision about what happened on the network, which is much more interesting for the analyst than a collection of heterogeneous and unrelated log files.

This chapter is organized as follows: Section 3.1 describes the security graph model we have designed to handle network information. Then, we present in Section 3.2 the implementation and some results on the scalability of our graph approach. Finally, Section 3.3 leverages a use case where the graph traversal can help the analysts to understand an attack.

The content of this chapter has been published at the IEEE Euro S&P Workshop on Traffic Measurements for Cybersecurity 2020 (WTMC2020) [85].

3.1 Building security object graphs from network events

In this section, we propose a restructuring of event logs coming from network sources to emphasize subsets of attributes that are of interest from a security perspective. To do so, we use a graph structure noted $G = (V, E)$, with V being the set of nodes and E being the set of edges.

In this section, we first define nodes V as *security objects* and present how we build the set of edges E among security objects to capture the semantics of a given event. Then, we explain how we build the global graph representing all security events present in our log files.

3.1.1 From a security event to a security object graph

Each log file can be described as a sequence of n ordered events $\{e_1, e_2, ..., e_n\}$ where e_i is an event resulting from the observation of an action in the network. Each logged event is made of several fields that differ depending on its type. For each type of event e, we select the fields corresponding to key information and create one or several Security Objects according to the field we find. A SO is thus a set of attributes, each attribute corresponding to a particular event field.

By construction, the set of SO attributes is included in the set of events attributes and the following properties are true:

- for a given event, the SO attribute sets are disjoint, this means that we do not add redundancy in the graph with our representation. In relational databases, normalization is performed to avoid redundancies and thus save memory space, facilitate updates and ensure data consistency [68]. With this property, we get the same advantages for our graph model.
- SOs can only retain a subset of the attributes of the event, which means that an SO is not a self-contained unit. We need several SOs to reconstruct the information contained in

one event. This property justifies our implementation choices for the community detection module in Chapter 4 and the anomaly detection module in Chapter 5.

Each event leads to a set of security objects. To preserve information related to a given event, links are created between security objects coming from this event. Let $l \in E$ be a link between two nodes s and d. l is defined by the quadruplet $(s, d, l_{time}, l_{type})$. l_{type} corresponds to the type of the link and l_{time} refers to the timestamp of the event. For example, if a network connection is logged at timestamp t_0, a *NetworkConnection* object is created and linked to an *IPAddress*: this link is of type *has_src_address* and has its attribute timestamp set to t_0.

The semantic of these links is derived from the CybOX model [16]. The various SO categories, their respective attributes and their links are represented in Figure 3.1. For clarity, colors and symbols are used in the figure to identify the various categories of SOs.

The types of nodes that compose our graph is of course a crucial element in relation to our objective, that is to provide the analyst with a better vision of security insthey may highlight. Following this idea, *Network* objects (♣ in grey) such as IP addresses or port numbers can highlight possible port scans or malware spreading from host to host. *Network Services* objects (♥ in blue) such as DNS represent common targets as they can paralyze a whole network. Well known attacks on these systems are DHCP Spoofing or DNS Poisoning and Spoofing. *File Transfer* objects (■ in yellow) such as file checksum or mail objects are valuable to detect an attack campaign based on malware spreading. *Application Services* objects (● in purple) allow to capture specific characteristics of popular application such as *http_referrer* that can be symptomatic of a CSRF attack [23]. *Security Services* objects (♠ in orange) such as invalid certificate can be indicators for a Man-In-the-Middle attack or a brute-force attack. Finally, *Alerts* objects (♦ in red) represent potential attacks detected by an IDS or protocol anomalies detected in the monitored network. We include in this category the `Indicator` SO that corresponds to an Indicator of Compromise i.e., an artifact observed on a network that indicates an intrusion with high confidence, as well as the `Weird` SOs that corresponds to an alert issued by an anomaly detector.

Figure 3.1 – Complete Model Representation

To show how we derive our SOs and their links from an event type and its attributes, we consider an event from the `conn.log` file with all its fields. All the fields, their types, their descriptions and the associated SO types are given in Table 3.1.

Fields	**Type**	**Description**	**security object**
ts	time	timestamp of first packet	∅
uid	string	unique identifier of connection	NetworkConnection
id.orig	addr	source IP address	IPAddress
id.orig_p	port	source port number	NetworkConnection
id.resp_h	addr	destination IP address	IPAddress
id.resp_p	port	destination port number	Port
proto	enum	transport layer protocol of connection	NetworkConnection
service	string	application protocol ID sent over connection	∅
duration	interval	how long connection lasted	NetworkConnection
orig_bytes	count	number of payload bytes originator sent	NetworkConnection
resp_bytes	count	number of paylod responder sent	NetworkConnection
conn_state	string	connection state	NetworkConnection
local_orig	bool	value=true if connection originated locally	deleted
local_resp	bool	value=true if connection responded locally	deleted
missed_bytes	count	number of bytes missing (loss packets)	NetworkConnection
history	string	connection state history	NetworkConnection
orig_pkts	count	number of packets originator sent	NetworkConnection
orig_ip_bytes	count	number of originator IP bytes	NetworkConnection
resp_pkts	count	number of packets responder sent	NetworkConnection
resp_ip_bytes	count	number of responder IP bytes	NetworkConnection
tunnel_parents	set[string]	if tunneled, connection uid of encapsulating parents	∅

Table 3.1 – The field types description and security objects for network connections

Network connections are at the basis of our model. In a `conn.log` file, the *uid* is the unique identifier of a network connection Therefore, to represent a conn.log network connection, we first build a NetworkConnection SO using the `conn.log` event *uid* value as its own *uid* attribute.

The *id.orig_h*, *id.orig_p*, *id.resp_h* and *id.resp_p* fields are the addresses and port numbers. IP addresses are arguably very important pieces of information that are often seen in Indicator of Compromise sharing platforms. The destination port number is also useful to link two events sharing the same service and can therefore be relevant to identify specific types of attacks. However, the source port is often set dynamically and is not often representative of any link between two elements. As a consequence, in our graph model, we collect the *id.orig_h* and *id.resp_h* addresses as two new *IPAddress* objects. We also collect the destination port number in a new *Port* object. However, as the source port is not as relevant as the others, we let it as an attribute in the *NetworkConnection* object.

As the fields *proto*, *duration*, *orig_bytes*, *resp_bytes*, *conn_state*, *misssed_bytes*, *history*, *orig_pkts*, *orig_ip_bytes*, *resp_pkts* and *resp_ip_bytes* define the characteristics of the connection, we collect them as attributes in the *NetworkConnection* object.

local_orig and *local_resp* are fields that indicate respectively if the source or the destination adresses are part of the local subnet, this information being configured in the Zeek configuration. As we did not define the local subnets, these fields are always empty and can be deleted from our model. If local subnets were to be defined in the configuration, it will be interesting to add these fields to the *IPAddress* object as they are directly related to this SO. This aspect has been left for future work.

service and *tunnel* are particular fields. If they are set, they refer to another object. For example, if the value of the field *service* is set, it means that Zeek was able to decode the service and has created another event in an other log file. Similarly, if the the field *tunnel* is set, it refers to an other network connection that encapsulated the network connection we consider. We will see how we handle relations with other events in the next section.

Finally, timestamp *ts* is added as an attribute of each link to keep the chronology of the events.

Once all the SOs have been extracted from an event, we build a graph representing it based on the model on Figure 3.1. Figure 3.2 shows the graph representing a network connection. It is made of four SOs: a source `IP Address` SO, a destination `IP Address` SO, a `Destination Port` SO and the `NetworkConnection` SO itself. This last SO captures attributes corresponding to the fields we identified as less important to create relations between events. The relations linking all these SOs together are *has_src_address*, *has_dst_address* and *has_dst_port*.

conn.log	ts uid id.orig_h id.orig_p id.resp_h id.resp_p proto service duration orig_bytes resp_bytes conn_state local_orig local_resp missed_bytes history orig_pkts orig_ip_bytes resp_pkts resp_ip_bytes tunnel_parents

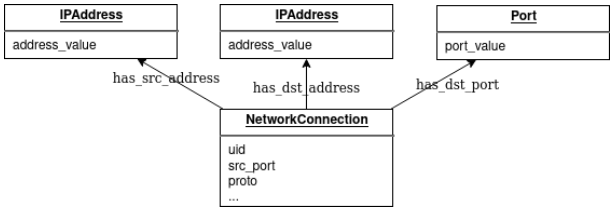

Figure 3.2 – Building a graph from one network connection as shown in the conn.log file

This example is based on a network event coming from Zeek, but the model can be extended to other type of events by following the same building process.

In this section, we explained how to build an SO graph from a single event. In the next section, we describe how to iterate this process to build a graph representing multiple events coming from multiple sources.

3.1.2 From heterogeneous log events to a graph of security objects

After having explained in the previous section how we create a graph for a single event, we present in this section how we build a single graph from multiple events coming from multiple sources.

Building a single graph from multiple events is performed as follows: for each event, and according to its type, we extract the SOs and the links between them as described in the previous section. In other words, we first build a subgraph representing this event. We then take each SO of the subgraph. If this SO already exists in the global graph (for instance, the same *IPAddress* was already identified in a previous event), we replace the SO in the new subgraph by the SO

that already exists in the global graph. Therefore, if an event contains an SO that was already found in a previous event, the subgraph that represent it will be linked to the global graph trough this SO. In detail, we have selected nine types of objects to link events together. These are IP addresses, domain names, destination ports, file names, URIs, MAC addresses, email addresses as well as connection and file transfer identifiers (often assigned by network analyzers). We insist on the fact that it is the type of each object that is taken into account. Thus, an IP address object present in a connection log as the source address and in a DNS resolution as the requested IP address will make the link between the two events regardless of its meaning in the log. The only exception is the port, which must be a destination port. Indeed, a correlation rule between two events with a source port and a destination port is of little interest from a security point of view. The choice of these objects comes partly from the study of the types of IoCs frequently used and partly from the experience of security analysts.

Our approach also allows to collect events coming from different log types in the same graph. As an example, let's consider three log events extracted from the Zeek [114] analysis of the CICIDS2017 dataset [131]. The three log events represent the same FTP connection analyzed by different modules of the Intrusion Detection System. The first event e_1 comes from the `conn.log` file. It is a report on the TCP network connection from the IP address `192.168.10.15` to the IP address `192.168.10.50` on port `21`. The second event e_2 comes from the `ftp.log` file. It provides details about the FTP reply. The third event e_3 corresponds to file transfer details. A graph for each of these three events is represented on the left on Figure 3.3. We represent the global graph composed of six SOs and obtained from the three previously described sub-graphs on the right on Figure 3.3: the first event is surrounded by a solid blue line ($e1$), the second is surrounded by a dotted red line ($e2$) and the third is surrounded by a small yellow dotted line ($e3$). e_1 and e_2 share a reference to the same `NetworkConnection` SO (same `uid` value) and e_2 and e_3 share the same `FileTransfer` SO (same `fuid` value).

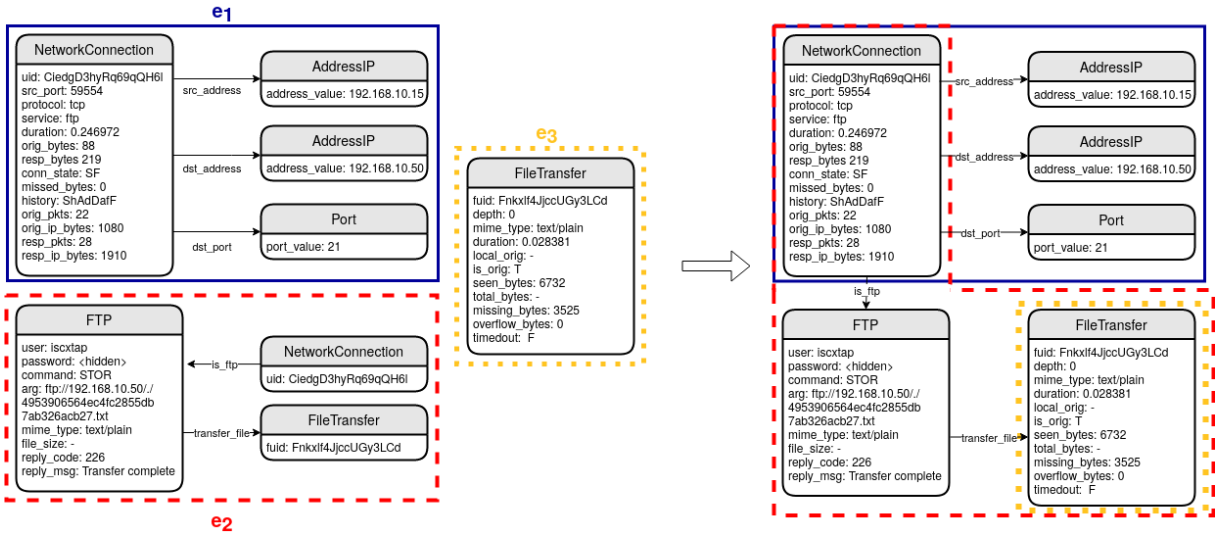

Figure 3.3 – Combining multiple events in a single graph

In order to build the global graph encompassing the various log types, we simply apply the same process to the different events coming from them. We first repeat the process with another type of log file, namely `ftp.log` that groups specific information about FTP connections. The fields of this type of event are written in the table at the top of Figure 3.4. In this type of log, we extract the `uid` and `fuid` fields, respectively the network connection identifier and the identifier of a file transfer. We build two new objects, the `FTP` object and the `FileTransfer` object that we attach on the ones already built as shown on Figure 3.4.

ftp.log	ts uid id.orig_h id.orig_p id.resp_h id.resp_p user password command arg mime_type file_size reply_code reply_msg data_channel.passive data_channel.orig_h data_channel.resp_h data_channel.resp_p fuid

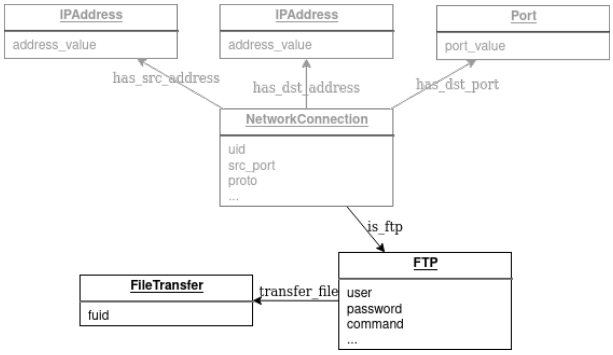

Figure 3.4 – Complementing the graph built from a network connection with information relative to a FTP connection, as shown in the `ftp.log` file

Similarly, we extract the `fuid` and `conn_uids` fields from the `files.log` file that groups every piece of information about file transfer and add the relevant attributes to the `FileTransfer` SO. We also add a new `is_file_transfer` edge between the `NetworkConnection` SO and the `FileTransfer` SO.

By combining the different log files, the graph enables to deduce relationships between events coming from different log types and thus, as will be shown in Section 3.3, to learn more complex patterns.

3.1.3 Comparison with CybOX and STIX Models

The CybOX and STIX models provide a solid basis for the representation of security events in graphical form. However, these models are mainly intended to describe compromise indicators for knowledge sharing with the cyber community. In this section, we indicate how our model differs from these standards.

3.1.3.1 Comparison with CybOX model

As described in the Chapter 2, CybOx [17] is a standardized language for representing information about cyber observables. It offers a total of 81 basic objects including 26 objects linked

files.log	ts fuid tx_hosts rx_hosts conn_uids source depth analyzers mime_type filename duration local_orig is_orig seen_bytes total_bytes missing_bytes overflow_bytes timedout parent_fuid md5 sha1 sha256 extracted extracted_cutoffextracted _size

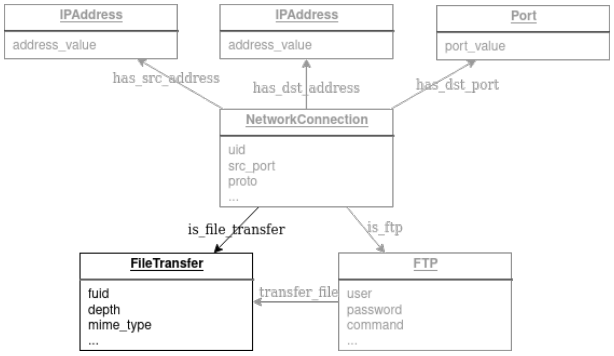

Figure 3.5 – Complementing the graph built from a network connection and the ftp connection with information relative to the file access, as shown in the `files.log` file

to network connections as, for example, *Address*, *DNS Query*, *HTTP Session* or *X509 Certificate*. However, the complexity of this schema (deep hierarchical tree) makes its application difficult.

For example, a simple network connection in CybOX is described by a total of seven linked instance of four different objects: an instance of the*NetworkConnectionObject* itself, two instances of the *SocketAddressObject*, two instances of the *PortObject* and two instances of the *AddressObject*. If the network connection is an HTTP session or a DNS request, an additional object, the *Layer7Connection* object has to be added. The UML diagram of CybOX *NetworkConnectionObject* is depicted in Figure 3.6. It is extracted from the OASIS specifications of the *NetworkConnectionObject* [21].

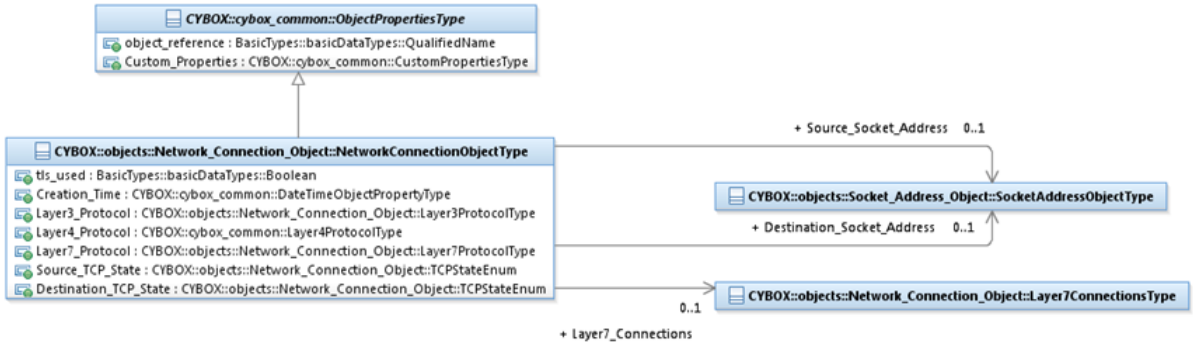

Figure 3.6 – UML diagram of the *NetworkConnectionObject* CybOX class

The *SocketAddressObject* is itself the composition of two objects, the *PortObject* and the *AddressObject*. The UML of the *SocketAddressObject* is represented in Figure 3.7. It is extracted from the OASIS specifications of the *SocketAddressObject* [22].

Figure 3.7 – UML diagram of the *SocketAddressObjectType* CybOX class

The subgraph induced for a simple network connection has a depth of three (i.e., three hops in maximum to reach a node from another node), whereas our representation of a network connection only has four SOs and a depth of two. Moreover, it requires up to 13 objects to describe an HTTP session with CybOX, and the induced graph has a depth of six. A complete example representing an HTTP Network Connection instance in CybOX language is given in Annex 6.2.3.

While CybOX comes in a variety of objects, it can extensively describe only two types of application protocols. It only proposes to describe DNS queries and HTTP sessions. In our model, we propose to represent not only HTTP and DNS, but also SSH, FTP, Kerberos, SMTP, SNMP, DCE/RPC and DHCP sessions.

Network communications offer lots of protocol likely to be used by attackers. The choice of describing more protocols was thus made in the Sec2Graph model to be able to distinguish as precisely as possible attacks from normal events. In addition, in order to be as silent as possible, the attackers adapt themselves to the attacked network and try to blend in with the current traffic to avoid detection. Taking into account detailed information about the protocols used can help to distinguish these attacks from normal traffic. For example, knowing the exact version of OpenSSH can distinguish a regular user from an attacker. Moreover, in our case, we want to perform graph analysis like graph traversals. The deep hierarchy complicates the graph traversal, sometimes requiring to cross a large number of nodes before reaching the desired one. A deep hierarchy also induces a higher number of nodes. The Sec2Graph model was tuned to provide a model able to represent several millions of events.

The complexity of CybOX was one of the reason why STIX integrated it in its own model with some simplifications. We present STIX in the next section.

3.1.3.2 Comparison with STIX model

STIXv2 integrates part of the objects of the CybOX standard in an enriched form. In STIXv2, there are less objects corresponding to observables than in CybOX but these objects contain more information since some CybOX objects have been merged into a single STIX object.

In total, STIXv2 integrates ten cyber-observables objects related to network connections including *Domain*, *Email Address*, *Email Message*, *File*, *IPv4 Address*, *IPv6 Address*, *MAC Address*, *Network Traffic*, *User* and *X509 Certificate*.

Moreover, the structure of the STIX relational graph is less complex than in CybOX. STIXv2.0 does not contain any links between objects representing cyber-observables and even if the com-

mittee introduced in STIXv2.1 [1] seven relations between cyber observable Objects, the hierarchy between the object is not as deep as in CybOX.

For example, a network connection in STIXv2.1 is represented in json in Figure 3.8. STIXv2.1 represents a network connection with three instances of objects, an instance of the *network-traffic* object and two instances of the *ipv4-addr* object. Unlike CybOX and Sec2graph, it does not represent port as specific objects but as attributes of the *network-traffic* object. In our model, we choose to represent destination port in a separate object because this information is useful to detect port scan attacks or non-standard port usage.


```
{
  "type": "ipv4-addr",
  "spec_version": "2.1",
  "id": "ipv4-addr--4d22aae0-2bf9-5427-8819-e4f6abf20a53",
  "value": "198.51.100.2"
}
{
  "type": "ipv4-addr",
  "spec_version": "2.1",
  "id": "ipv4-addr--ff26c055-6336-5bc5-b98d-13d6226742dd",
  "value": "198.51.100.3"
}
{
  "type": "network-traffic",
  "spec_version": "2.1",
  "id": "network-traffic--2568d22a-8998-58eb-99ec-3c8ca74f527d",
  "src_ref": "ipv4-addr--4d22aae0-2bf9-5427-8819-e4f6abf20a53",
  "dst_ref": "ipv4-addr--ff26c055-6336-5bc5-b98d-13d6226742dd",
  "protocols": [
    "tcp"
  ],
  "src_port": 2487,
  "dst_port": 1723,
}
```


Figure 3.8 – A network connection in STIX v2.1 model

STIXv2.1 is closer to the model proposed in this thesis than STIXv2.0, but the number of relations between cyber-observables is still restricted and STIXv2.1 cannot extensively describe application protocols. STIXv2.1 does not allow a detailed description of the protocols that are used and of all the attributes related to them. Indeed, STIX focuses on the sharing of IoC, not on the description of every details of network connections. It aims to provide to security analysts an overall context of an attack with specific IoC. In this thesis, we want a detailed view of the network communications in a system and each attribute has its importance. We assume that an attack can take place without a IoC being present nor an alert being raised. Indeed, even if they are important elements, IoCs can be outdated as attackers change their IP address or domain name to evade detection for example. Similarly, attackers can bypass a misuse-based IDS by modifying their attack protocol or by using an unknown technique. Our model therefore takes into account

1. This modification was published on March 20, 2020

the IoC and alerts from IDSs, but also describe the connections in sufficient detail for an analyst to be able to spot anomalies that reveal attacks.

3.2 Model implementation

Our implementation fulfills two requirements: first, it is scalable and second, it makes information analysis easier for security analysts. In this section, we first describe the technical choices we made to meet the scalability requirement, test the implementation resulting from these choices on a large dataset and present a use case illustrating information searching and linking by the security analyst.

3.2.1 Implementation and configuration setup

In this section, we describe our implementation choices regarding data acquisition, extraction of data of interest, link building and data storage. These implementation choices have in common that they make possible to use our tool in a real-world environment where it must be able to manage millions of events in a reasonable time.

3.2.1.1 Data acquisition

We chose to use three heterogeneous types of data, two internal types of data producing real-time data on network activities and alerting on potential attacks and an external static type of data providing information on indicators of compromise. We have chosen these types of data because they are frequently used in SOCs and allow us to validate the fact that our model adapts to the data used.

The first type of internal data is the Zeek Intrusion Detection System (formerly Bro)[114]. Zeek is an NIDS that inspects in depth all traffic on a link or in network capture files for signs of suspicious activity. It provides a set of log files that records a network's activity in high-level terms. These logs include a record of every connection seen on the wire, as well as application-layer transcripts such as HTTP sessions with the requested URIs. Zeek writes this information into tab-separated log files.

We use this system because it is able to generate heterogeneous and detailed information in a structured format from a network capture. This information can then be used for forensic analysis and/or to obtain a status report on a network.

All Zeek log files used in this thesis are listed in Table 3.2. Zeek provides also other types of files such as `radius.log`. However, as we do not use datasets including radius connection, we do not integrate them into our model. Once again, if needed, those types of information can easily be added by extending the model with new classes of security objects and new links.

For all the datasets used in this thesis, we provided Zeek with a capture file and used the generated log files as sources of our processing pipeline.

File	Description
conn.log	IP, TCP, UDP, ICMP connection detail
dns.log	DNS query/response details
files.log	File analysis results
ftp.log	FTP request/reply details
http.log	HTTP request/reply details
ssh.log	SSH handshakes
ssl.log	SSL handshakes
weird.log	Anomalies and protocol violations
x509.log	x509 Certificate Analyzer Output
syslog.log	Syslog messages
dhcp.log	DHCP lease activity
kerberos.log	Kerberos authentication
smtp.log	SMTP transactions
snmp.log	SNMP communications
tunnel.log	Details of encapsulating tunnels
rdp.log	Remote Desktop Protocol (RDP)

Table 3.2 – Description of Zeek log files

The second type of internal data is Suricata [2] alerts. Suricata is another NIDS which provides signature-based alerts. It also proposes, as Zeek, metadata about the protocol headers in a json file. At the begin of this thesis, Zeek protocol parsers and loggers capabilities were more complete than the ones of Suricata. However, at time of writing, Suricata released its version 5.0.0 (October, 15^{th} 2019) that added RDP, SNMP, FTP and SIP parsers and loggers to its capabilities, making it comparable to Zeek. We generated Suricata alerts based on the open-source set of rules from Emerging Threats [3]. Suricata can generate many false positives but they are a good source of information for the initial step in a forensic analysis. We integrated them as Indicator objects in our model and linked them to the network connections that triggered the alert.

We finally used the OTX database as an external static source of IoC. Each IoC is an instance of the class Indicator. We kept 6 types of IoC: IP addresses, URI, Domain, Email and FileHash. These IoCs can indeed directly be linked to the SOs of our model IPAddress, URI, Domain, Mail, and File.

These three types of data provide three common sources of data seen in a SOC: network connection data, IDS alerts and threat intelligence data. This ensures that the proposed model is applicable to data used in a real-world environment. Other data sources could have been used, such as system logs but, in the context of this thesis, we have restricted the type of data handled to network data.

3.2.1.2 Data extraction and entity linking

To parse and extract fields from the chosen data sources, we uses the Gremlin language. Gremlin is a graph traversal language [127] used to retrieve data from a graph and modify them efficiently.

2. https://suricata-ids.org/
3. https://rules.emergingthreats.net/

In our implementation, for each file and for each record of that file, we extract relevant attributes and create SOs according to the structure defined in the Sec2graph model presented in Figure 3.1.

For the SOs *IPAddress*, *DestinationPort*, *File*, *URI*, *Mail*, *Domain* and *Mac*, we defined a specific method *getOrCreate* and defined the attributes of these objects as primary key. This allowed us to efficiently find if an instance of an object already exists. If so, a new link to the requested instance was created, else we created that object and then the associated link. The graph can be built dynamically without waiting that all objects are created to build the links. This is very important in SOC to be able to handle real-time data processing. However, as a counterpart, this step is time-consuming. The use of indexes is therefore necessary.

3.2.1.3 Data storage

Networks handle millions or billions of connections everyday. The database we choose to represent all these connections must thus allow to process a high volume of data without latency.

For our graph database, we used Janusgraph [132] with Elasticsearch [44] as an indexer and Cassandra [31] as a backend. JanusGraph is a scalable graph database optimized for storing and querying graphs. Elasticsearch indexing capabilities are used to speed up query processing. The method *getOrCreate* is indeed time consuming if there is no index as it requires to search in all instances of an object if a specific instance was already created. Cassandra was chosen because it ensures the isolation property. Transactions are often executed concurrently to handle the volume of ingested data. The isolation property (in case of transactional databases) ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. This ensure that two identical instances of one object are not created concurrently.

3.2.2 Scalability

To evaluate the scalability of our proposal, we chose to use the CICIDS2017 dataset [131] that is made of five pcap and csv files encompassing more than two million network sessions. This dataset was generated at the Canadian Cybersecurity Institute at the University of New Brunswick and contains five days (Monday to Friday) of mixed traffic, benign and attacks such as DoS DDoS, BruteForce, XSS, SQL injection, infiltration, port scan, and botnet activities.

Normal traffic was generated using the CIC-B-Profile [131] system that can reproduce the behavior of 25 users using various protocols (FTP, SSH, HTTP, HTTPS and SMTP). Attacks were executed using classic tools such as Metasploit and Nmap.

According to [54], the CICIDS2017 dataset is the most recent one that models a complete network configuration with components such as firewalls, routers, modems, and a variety of operating systems such as Windows, Ubuntu Linux or Macintosh and that has been used in several studies. The protocols in the capture (e.g., HTTP, HTTPS, FTP, SSH) are representative of the protocols that are used in a real network and a variety of common attacks are covered.

The generation of the complete graph took about 4 hours to generate a graph representing more than 6.2 millions events using the graph-oriented database Janusgraph and a script in *gremlin* language. The details of the distribution of nodes per classes is given in Figure 3.9. To decompose each log event in nodes and vertices, each log event is parsed based on several attributes composing the system objects. Each system object is compared to existing system objects in the database. If the system object already exists, we only add relationships to the existing system object.If not, we create the node and the edges according to the model presented in Figure 3.1. Verifying that there was no object duplication was the most computation-expensive operation.

Objects	**Count**
NetworkConnection	2 119 243
IPAddress	30 105
Port	4 254
Dcerpc	917
Dns	1 957 596
Dpd	10
Domain	36 500
FileTransfer	622 666
Filename	182

Objects	**Count**
Ftp	5 293
Kerberos	3 016
Http	521 172
Ssh	8 254
Ssl	332 768
URI	155 931
Weird	153 262
X509	295 512

Figure 3.9 – Number of objects created by categories

We argue that the computation time for graphs of millions of nodes is reasonable. Moreover, a large part of the algorithm can be computed in parallel and optimization can be made on the storage back-end and on the indexer. Figure 3.10 shows the time required to generate a graph according to the number of events. To draw this curve, we measured the time elapsed from the beginning of the generation to see if the generation time linear or proportional to the number of nodes/edges in the graph. The trend curve associated to the measures indicates that the time complexity is polynomial with the following equation: $f(x) = 4.8e^{-10}x^2 + 0.0027x + 387$. Therefore, the algorithm runs in time $O(n^2)$.

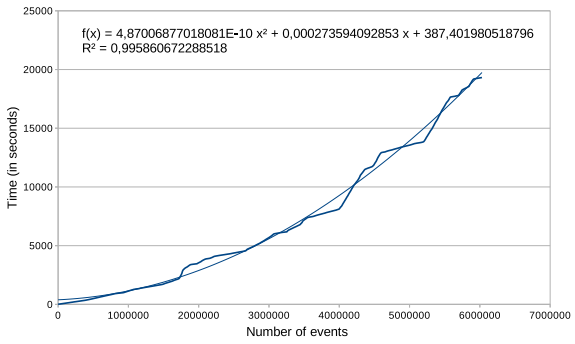

Figure 3.10 – Time to perform graph generation according to the number of events

The generation of graphs with millions of nodes representing 56 hours of network traffic only took 4 hours. Our graph generation algorithm is therefore scalable, its execution time being much shorter than the production time of the corresponding events.

3.3 Use case and security analysis examples

In this case study, we use a dataset made of logs from Zeek that were generated in the Stratosphere Labs as part of the Malware Capture Facility Project [4] in CVUT University, Prague, Czech Republic. The objective of this project is to store long-lived real botnet traffic and to generate labeled netflow files.

The dataset CTU-Malware-Capture-Botnets-254-1 describes a probable attack of type WannaCry Ransomware. WannaCry is a self-replicating ransomware malware. In May 2017, it was used in a massive global cyber attack, affecting more than 300,000 computers in over 150 countries.The infected host is *192.168.1.123*. We also discover two malicious addresses that are associated to the Wannacry campaign according to the IoC sharing platform OTX, *128.31.0.39* and *193.23.244.244*.

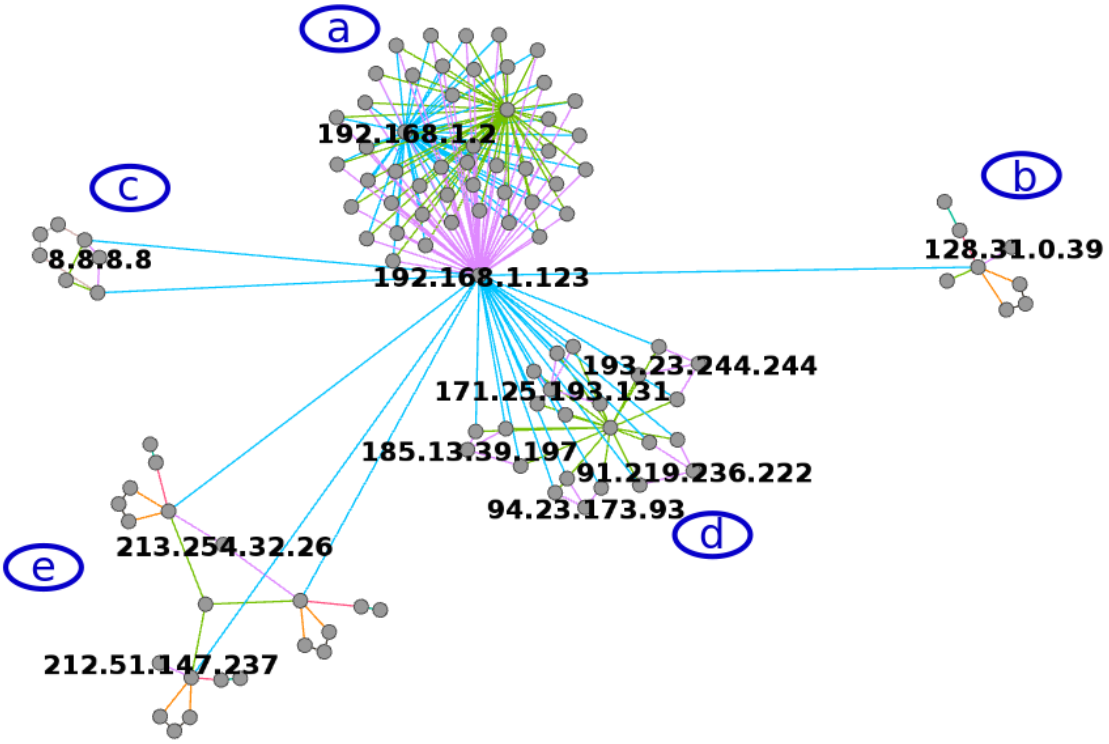

Figure 3.11 – Graph representation of the CTU-Malware-Capture-Botnets-254-1 on Gephi [19].

The graph in Figure 3.11 represents the graph built from the Zeek files in the dataset. The three main types of edges are drawn in pink *has_dst_address*, in green *has_dst_port* and in blue *has_src_address*. The potential victim is at the center of the graph (192.168.1.123). The graph structure allows to split the network connections on several groups by forming subgraphs densely connected. Five subgraphs can be identified:

4. S. Garcia. Malware Capture Facility Project, 2017.

- The first subgraph (a) corresponds to ICMP connections (port number equals to 1) to our target IP 192.168.1.123 and from the gateway address of the network 192.168.1.2.
- The second subgraph (b) corresponds to connections to a suspicious IP address related to the Wannacry attack (128.31.0.39).
- The third subgraph (c) corresponds to connections to the anycast address of Google Public DNS.
- The fourth subgraph (d) corresponds to multiple rejected connections on port number 443 originating from our potential victim.
- The fifth subgraph (e) corresponds to SSL connections through port 9001.

Security analysts are particularly interested by the two groups where IoC were found , i.e. subgraphs (b) and (d).

Instead of looking at all events containing a reference to the IP address *128.31.0.39*, the graph structure allows to simply request the neighborhood of the nodes to discover security objects directly related to the malicious IP address directly (only one connection) and indirectly (2 and 3-hops away). The query in gremlin and the associated results on the 2-hops and 3-hops neighborhood of the IP address are given respectively in Figure 3.12 and Figure 3.13. The results indicate that the connection is an SSL connection and that a certificate has been transfered. An analyst can directly identify the issuer of the certificate and decide if further action is needed.

```
gremlin> g.V().has('address_value','128.31.0.39').in().out().valueMap().unique()
==>[address_value:[192.168.1.123]]
==>[address_value:[128.31.0.39]]
==>[port_value:[9101]]
==>[filetransfer_overflow_bytes:[0],filetransfer_timedout:[F],filetransfer_depth:[0],
filetransfer_is_orig:[F],filetransfer_seen_bytes:[581],filetransfer_duration:[0.000000],
filetransfer_mime_type:[application/pkix-cert],fuid:[FYL9oq3frxxLLwnLZ3],
filetransfer_analyzers:[SHA1,MD5,X509],filetransfer_total_bytes:[-],
filetransfer_local_orig:[F],filetransfer_missing_bytes:[0]]
==>[ssl_curve:[secp256r1],ssl_client_subject:[-],ssl_last_alert:[-],
ssl_client_issuer:[-],ssl_version:[TLSv12],
ssl_cipher:[TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256],
ssl_established:[T],ssl_subject:[CN=www.f4mn3leykgi5kkmd.net],ssl_session_id:[F],
ssl_issuer:[CN=www.snlbigdcqpx5.com]]
```

Figure 3.12 – A Gremlin request to find 2-hop neighborhood of an IP Address object

The analysis of the subgraph (d) shows that the subgraphs are solely composed of rejected connections on port 443 to five different IP address coming from the supposed victim. Each time, they were 3 attempts for each destination address. This behavior is not dangerous by itself and the port used is often used legitimately but depending on the policy of the information system, it can be seen as an anomaly and requires to isolate the machine.

This use case shows that the analysis with a graph representation of security objects can make the analysts work easier by exhibiting links that otherwise should have been made visible. It also shows the importance of grouping nodes. In the next chapter, we explore further splitting the graph with the automatic detection of dense subgraphs called communities.

```
gremlin> g.V().has('address_value','128.31.0.39').in().out().out().valueMap().unique()
==>[certificate_version:[3],certificate_subject:[CN=www.f4mn3leykgi5kkmd.net],
certificate_sig_alg:[sha256WithRSAEncryption],basic_constraints_path_len:[-],
certificate_key_type:[rsa],certificate_key_length:[2048],
certificate_not_valid_before:[1488931200.000000],certificate_key_alg:[rsaEncryption],
certificate_serial:[7397D93AA2CF2686],certificate_exponent:[65537],
basic_constraints_ca:[-],certificate_curve:[-],
certificate_not_valid_after:[1497311999.000000],
certificate_issuer:[CN=www.snlbigdcqpx5.com]]
==>[domain_value:[www.ofgvaqpqbfqdzenmlroe.com]]
```

Figure 3.13 – A Gremlin request to find 3-hop neighborhood of an IP Address object

Conclusion

In this chapter, we presented a new data model based on the STIX model that allows to represent network connections as a graph of security objects. The objective of this graph model is to allow analysts to link useful pieces of information and thus make the analysis process easier.

We described the process to build the graph from network events and external sources of information such as IoC databases. We compared our model with CybOX and STIX model. We then provided details on the implementation and showed that our approach is scalable and can be used within a SOC. We also showed the usefulness of the graph structure for the security analyst through a case study.

We finally have shown that the graph structure highlights strongly connected sub-graphs that allow the analyst to focus on a specific part of the graph.

In the following, we will show how to use the graph structure to automatically display the most relevant elements to the analyst.

Chapter 4

Community discovery

Contents

Introduction

In our model, a graph can contain millions of nodes and edges. However, the use case presented in Chapter 3 reveals that subgraphs could be isolated to help the security analyst to find relevant pieces of information. Normal network activities are for instance more likely to be represented by strong, interconnected communities of security objects (SOs). By contrast, attacks typically consist of a few events and will generally be represented by decentralized SOs in the graph. SOs generated from attribute values containing evidence of the same attack are, by construction, strongly linked. Accordingly, identifying an attack in our graphical representation consists in identifying dense sub-graphs surrounding an SO being of the Indicator type and isolated from large hubs (assumed

to be normal activities) in the graph. These types of subgraphs are called communities in graph theory.

In this chapter, we explore the use of community detection methods and the challenges induced by these methods. We propose a strategy to tackle these challenges and adapt the community detection approach to our graph representation model.

This chapter presents the following contributions:

- We present the problem of community detection and the challenges this problem raises.
- We present how community detection algorithms could solve our problem of identifying relevant sub-graphs and how we have adapted the algorithms to our graph model.
- We compare the results of several community detection algorithms on the CICIDS 2017 dataset for different types of attacks. We evaluate our results based on the relevance of selected subgraphs and show that the community detection approach achieves good results in the task of selecting relevant objects.
- We discuss the relevance of the results according to the algorithm and the type of attacks and highlight the limits and potentials for improvement of our approach.

This chapter is organized as follows. Section 4.1 presents the problem of community detection and its challenges and explains how community detection techniques can help in isolating relevant subgraphs for security analysis. Section 4.2 presents experimentations with several community detection algorithms on the CICIDS 2017 dataset. Finally, Section 4.3 discusses the results obtained and exposes the limitations of the approach.

The experimental results presented in this chapter have been published in 2020 at the IEEE Euro S&P Workshop on Traffic Measurements for Cybersecurity (WTMC2020) [85].

4.1 Discovering communities in graphs for highlighting attack-related sub-graphs

In this section, we first present the problem of community detection and introduce modularity measure. We also describe some challenges related to community detection in graph analysis. Finally, we present some common techniques used in community detection with their advantages and drawbacks.

4.1.1 Definition of the community detection problem

In the social analysis domain, research has been carried out to identify people strongly connected to similar people but relatively isolated from others. These groups of people are called communities and techniques to find them are named community detection algorithms [49].

In relation to our work, the identification of communities can provide insight into how the graph is organized, allowing to focus on regions of the graph that are isolated and may reveal an anomaly.

An indicator or alert is used as a signal to identify communities of interest. All indicators do not have the same confidence index. Alerts, on their side, can be false positives. Nevertheless, the presence of more than one of these indicators as well as the presence of one or several alerts within a community may indicate an attack. The selection of elements close to an indicator or alert within the graph can make the analysis by security experts easier and help eliminate false positives.

For example, it is useful to identify the role of nodes in the communities to which they belong. Central nodes can be distinguished from those located at the border of communities, that themselves can serve as bridges between communities. In the context of an attack, central nodes can be clues of reconnaissance phase (port scan), and bridge nodes can show the means by which an attack can be spread to a whole information system. In a graph containing millions of nodes, identifying the role of nodes without splitting the graph in communities is very difficult.

Community detection algorithms are often based on the modularity maximization method first presented by Newman and Girvan in [107]. Its objective is to evaluate the quality of a graph partition. Modularity Q is defined as follows:

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j)$$

where $A_{i,j}$ represents the weight of the edge between o_i and o_j, $k_i = \sum_j A_{ij}$ is the sum of the weights of the edges attached to vertex o_i, c_i is the community to which vertex o_i is assigned, the δ-function $\delta(u, v)$ is 1 if $u = v$ and 0 otherwise and $m = \frac{1}{2} \sum_{ij} A_{ij}$. The maximization of the modularity measure allows separating the nodes into communities. Graphs with high modularity have dense connections between the nodes within the same community, but sparse connections between nodes in different communities.

4.1.2 Common challenges of community detection

While community detection is useful for understanding the structure of a graph, it introduces a number of challenges presented below.

Overlapping problem. Community detection algorithms often split a graph into disjoints communities but sometimes two communities can overlap. The overlapping problem refers to nodes that can belong to multiple communities. In this case, the node will be randomly assigned to one or the other community.

In our case, it means also that while an analyst wants to retrieve events linked to an indicator, the partitioning does not guarantee that all the objects representing an event are in the same community.

To overcome this difficulty, we add an additional step at the end of the community detection algorithm: if events are partially represented due to missing objects in a community, we add these missing objects to the community. We obtain a division of our graphs into smaller sub-graphs and

keep only the community containing the bigger number of alerts or IoCs.

Scalability problem. Finding communities in a graph is an NP-difficult problem. Rigorously accurate community detection algorithms can only be applied to small graphs, the computation time being too large for graphs with millions of nodes.

It is therefore common to use approximation algorithms, i.e. methods that do not provide an exact solution to the problem, but only an approximate solution, with the advantage of lower complexity. Approximation algorithms are often non-deterministic, as they provide different solutions to the same problem, for different initial conditions (like the order of nodes/edges) and/or different parameters of the algorithm (like the size of communities). The objective of these algorithms is to provide a solution that differs only slightly from the optimal solution while being computationally fast.

As our graph contains millions of nodes and edges, the only possible choice is to use approximation algorithms, and therefore non-deterministic algorithms. As the quality of the partition is not our main goal, if the approximation provide relevant elements for forensic analysis in a reasonable time, we consider that the algorithm achieves its goal.

Validation problem. As Fortunato points out [50], there is no universal protocol to validate the results of community detection algorithms and compare their performance.

The quality of partitioning is often measured using the modularity measure. However, depending on the structure of the graphs, the number of edges, and the centrality of the nodes, this measure is not necessarily well suited. For example, the size of communities can also be an important factor to take into account depending on the field of study.

The goal of our approach is not to maximize the modularity measure but to extract subgraphs useful for forensic analysis. To validate the communities found by the different algorithms, we thus choose our own criteria, that expresses the number of useful objects selected in a given community in the case of attack analysis. We detail how we evaluate this in Section 4.2.

4.1.3 Common methods used for community detection

We detail three classical methods of community detection in graphs, i.e., the Fast Greedy algorithm, the Louvain algorithm, and the Label Propagation algorithm.

Fast Greedy algorithm. The *Fast Greedy* community detection algorithm by Clauset *et al.* [34] is an agglomerative hierarchical clustering method that optimizes the modularity measure. It merges individual nodes into communities in a way that greedily maximizes the modularity score of the graph by considering a simplified formula of the modularity equation. This algorithm runs almost in linear time on sparse graphs.

The simplified formula of the modularity equation proposed by Clauset *et al. et al.* is the following:

$$Q = \sum_i (e_{ij} - e_i^2)$$

where e_{ij} is the fraction of edges in the network that connects the communities c_i and c_j and $e_i = \sum_j e_{ij}$.

The workflow is as follows:

1. Start with $k = n$ communities (every node is its own community, n being the number of nodes in the graph).
2. Calculate ΔQ the change in modularity for every pair of communities.
3. Merge the two communities having the greatest increase (or the smallest decrease) in Q.
4. Repeat steps 2 and 3 until all nodes are contained in one single community.

The final node partition is selected by calculating Q at every split, and selecting the final number of communities to match the maximum value of Q. The modularity of the graph is computed at each merge, and the level with the highest Q is selected.

Louvain algorithm. The *Louvain algorithm* [24] is a widely used greedy optimization method for modularity maximization that runs in time $O(n \log n)$ and is therefore more suitable for large graphs. In addition, it does not require to specify in advance the number of communities to be found. The Louvain algorithm works as follows: first, it searches for small communities by optimizing modularity locally. Then, it groups the nodes belonging to the same community and builds a new graph whose nodes are the communities. These steps are repeated iteratively until a maximum modularity is achieved.

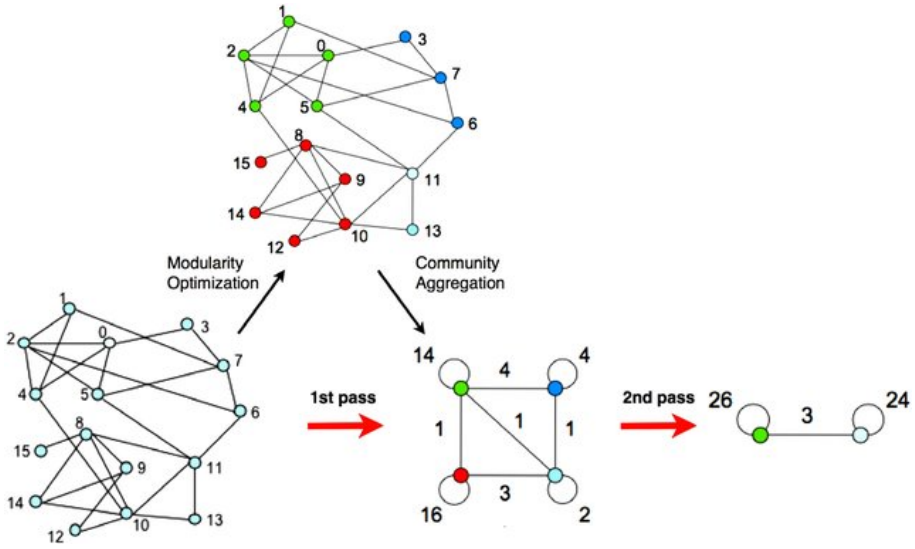

Figure 4.1 – An overview of the Louvain algorithm (extract from [24])

The algorithm can be described as follows:

Phase 1

1. Begin with n communities (every node is its own community, n being the number of nodes in the graph).

2. Consider each node n_i sequentially, calculate ΔQ, the change in modularity made by moving n_i to the community of each of its neighbors n_j.
   - Make the move associated with the maximum gain if the gain is positive.
   - If there is no possible improvement in modularity, n_i remains in its community.
3. Repeat step 2 until no further improvement can be made.

Phase 2

1. Begin by creating a new network where the nodes are the communities output from the first phase:
   - Weight the edges between these new *nodes* with the sum of the edges between the members of the corresponding communities; edges within communities are now self-loops.
   - Reapply Phase 1 to this network.
2. Repeat Phase 2 until no more changes increase modularity.

This process is explained in Figure 4.1 extracted from [24]. Phase 1 corresponds to the Modularity Optimization operation and Phase 2 corresponds to the Community Aggregation operation.

In phase 1 at step 2, nodes are considered sequentially. This may affect the final result because the output depends on the order of the nodes being considered. The authors argue that this does not significantly influence the modularity obtained but does influence computation time. Moreover, Blondel *et al.* [24] compare their method with the Fast Greedy algorithm and show that they surpassed the Fast Greedy algorithm which has a tendency to produce too large communities while being slower.

Label Propagation algorithm. Another commonly used technique for community detection is the *Label Propagation* technique[123]. It computes communities by propagating labels throughout the graph. The intuition behind this algorithm is that a single label can quickly become dominant in a group of densely connected nodes, but it will be difficult for it to reach a weakly connected region. Labels will remain within a group of densely connected nodes, and nodes that end up with the same label at the end of the algorithm can be considered part of the same community. This algorithm does not require prior information about the communities. Initially, each vertex is assigned a different label. After that, each vertex chooses the dominant label in its neighborhood in each iteration. Ties are broken randomly and the order in which the vertices are updated is randomized before every iteration. In more details, the Label Propagation algorithm works as follows:

1. Every node begins with a unique label.
2. Modify in a random sequential order the label of each node to be that of the majority of its neighbors.
3. Repeat step 2 until each node has the label of the majority of its neighbors.

In the end, the unique labels define the communities in the network. As for the Louvain algorithm, and because of the random selection in step 2, this algorithm does not have a unique

solution. In [106], the authors compare the Label Propagation algorithm and the Louvain algorithm. They found that if the Label Propagation algorithm is slightly faster than the Louvain algorithm, the Louvain algorithm has better results in finding communities.

We should mention that we studied three other community detection algorithms (*Infomap* [129], *Spinglass* [146] and *Walktrap* [119]). The first one generates too many small communities and the two others are too slow.

As the results of partitioning can depend on the nature of the graph, we choose to evaluate the relevance of community detection in the forensic analysis task with the three methods we described in this section: the Fast Greedy algorithm, the Louvain algorithm, and the Label Propagation method.

4.2 Implementation and experimental results

To implement the community detection, we used the *python-louvain* library [1] for the Louvain algorithm and the *igraph* library [2] for the Fast Greedy algorithm and for the Label Propagation algorithm. We also used the *networkx* library [3] for the graph processing. We kept the default parameter for each algorithm. All our experiments were carried out with a Linux computer with 8 GB of RAM.

Results were evaluated through the criteria of **attack detection relevance**: our approach must allow reducing the number of objects to analyse without removing relevant information for the analyst such that he or she can obtain all the objects related to an Indicator.

In the following sections, we first introduce the dataset we used and evaluate the relevance of our approach with three community detection algorithms, i.e., the Louvain algorithm, the Label Propagation algorithm and Fast Greedy algorithm. We then discuss the strength and limits of our approach.

4.2.1 Choice of the dataset

We choose to use the CICIDS 2017 dataset [131] presented in Section 3.2 to perform the community detection since the protocols in the capture (e.g., HTTP, HTTPS, FTP, SSH) are representative of protocols used in a real network and a variety of common attacks are covered, allowing us to evaluate our method on a realistic dataset. Recall that this dataset contains five days (Monday to Friday) of mixed traffic, benign and attacks such as DoS, DDoS, BruteForce, XSS, SQL injection, infiltration, port scan, and botnet activities. The data set is also labeled, allowing us to quantify the effectiveness of our method. As in the previous chapter, we used the Zeek IDS tool to generate log files from the capture files. Details about the dataset and the number of generated events are presented in Table 4.1.

1. https://github.com/taynaud/python-louvain
2. https://github.com/igraph/python-igraph
3. https://networkx.github.io/

In addition, we generated alerts with the Suricata IDS using the EmergingThreats rules package. Indicator objects referring to these IDS alerts act as nodes of interest and allow us to select communities of interest.

Date	**Attacks**	**Nb of packets**	**Nb of alerts /IoC**	**Nb of Zeek events**
Monday	∅	11.709.971	79	1.162.527
Tuesday	BruteForce: FTP Patator, SSH Patator	11.551.954	2.511	995.213
Wednesday	DoS/DDoS: Slowloris, Slowhttptest, Hulk and GoldenEye, Heartbleed Attack	13.788.878	77	1.474.868
Thursday	Web Attacks: Web Brute Force, XSS and SQL Injection. Infiltration attacks: exploit metasploit, Cool disk	9.322.025	25.973	1.019.783
Friday	DDoS LOIT, Botnet ARES, PortScans	9.997.874	365	1.374.021

Table 4.1 – Description of the CICIDS2017 dataset and number of security events generated per day.

4.2.2 Evaluation criteria

The evaluation criteria is the relevance of attack detection, and thus the relevance of the objects selected by the community detection algorithms. To our best knowledge, there is no previous evaluation of data reduction and attack analysis with a graph-based model on the CICIDS2017 dataset.

We define False Positives (FP), False Negatives (FN), True Negatives (TN), and True Positives (TP) as follows: FP are edges wrongly selected, FN are edges that do not appear in the selected graph but that are part of an attack, TN are edges not selected in the graph and that are not being part of an attack and finally, TP are edges correctly selected. Note that these definitions of true/false positive/negative do not correspond to those used in intrusion detection. Indeed, our goal here is communities detection and our definitions make it possible to evaluate the results of the three tested algorithms.

To compute these values, we need to know which links in our graph correspond to events generated by an attack and which links correspond to normal traffic. In the CICIDS2017 dataset, an event is labeled with the type of attack. We use the labels as follows: if an event is part of an attack, we add an "attack" attribute equals to '1' to all links in the subgraph representing that event. Otherwise, we add an "attack" attribute equals to '0'. The feature *attack* is then used to compare the set of edges selected by the community detection algorithm and the set of edges having the *attack* attribute set to '1'.

To evaluate the efficiency of the model and the community detection methods, three common measures are used: *Precision*, *Recall* and *F1-score*. These measures are based on the FN, FP, TN and TP scores.

- **Precision** corresponds to the ratio of correctly retained edges divided by the set of retained edges. It tends to 1 if only malicious edge are added to the selected graph.

$$Precision = \frac{TP}{TP + FP}$$

- **Recall** corresponds to the percentage of correctly retained edges divided by the set of truly malicious edges. It tends to 1 if no malicious edge is forgotten.

$$Recall = \frac{TP}{TP + FN}$$

- Finally, the **F1-score** takes into account both precision and recall. While accuracy measures the proportion of all correctly labeled edges over all edges, we choose to use the F1-score metric that is more suitable when there is an imbalanced class distribution (which is often the case in the security field) and when the reduction of false negative and false positive is more important.

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

4.2.3 Experimental results on attack detection relevance

To evaluate whether our proposal is able to correctly retain relevant objects and relations, we built seven graphs, each one corresponding to half a day of traffic involving attacks. The attacks retained for the evaluation are: FTP brute force, SSH brute force, Heartbleed, Web attack, Infiltration attack, ARES Botnet and Portscan.

We then performed community discoveries on each graph and retained communities containing IDS alerts. As mentionned earlier, we used the implementation of Fast Greedy [34], Louvain [24] and Label Propagation [123] algorithms to evaluate the ability of each algorithm to select relevant sub-graphs containing events related to an attack.

Algorithm	**Recall**	**Precision**	**F1-score**
Fast Greedy	0,683	0,382	0,490
Louvain	0,943	0,633	0,757
Label Propagation	**0,969**	**0,778**	**0,863**

Table 4.2 – Synthesis of Recall, Precision and F1-score results per community discovery algorithm

The results of Precision, Recall, and F1-score computation presented in Table 4.2 shows that the Label Propagation method is the best method for Precision, Recall, and F1 before the Louvain algorithm and the Fast Greedy method from Clauset et al. The results for Recall are particularly good (0.969) indicating that the graph model associated with the Label Propagation method

makes the majority of events related to an attack to be selected within the same community. This is an interesting result because it shows that computing communities allows to identify an important part of the information related to an attack.

Moreover, the Precision results for the same algorithm (0.778) show that it allows to isolate well the events related to an attack from the normal events. Again this is an interesting result because it shows that the communities that are computed does not overwhelm the analyst with useless information not linked to an attack.

In Figures 4.2 and 4.3, we compare the results of Precision and Recall for each type of attack and for each algorithm.

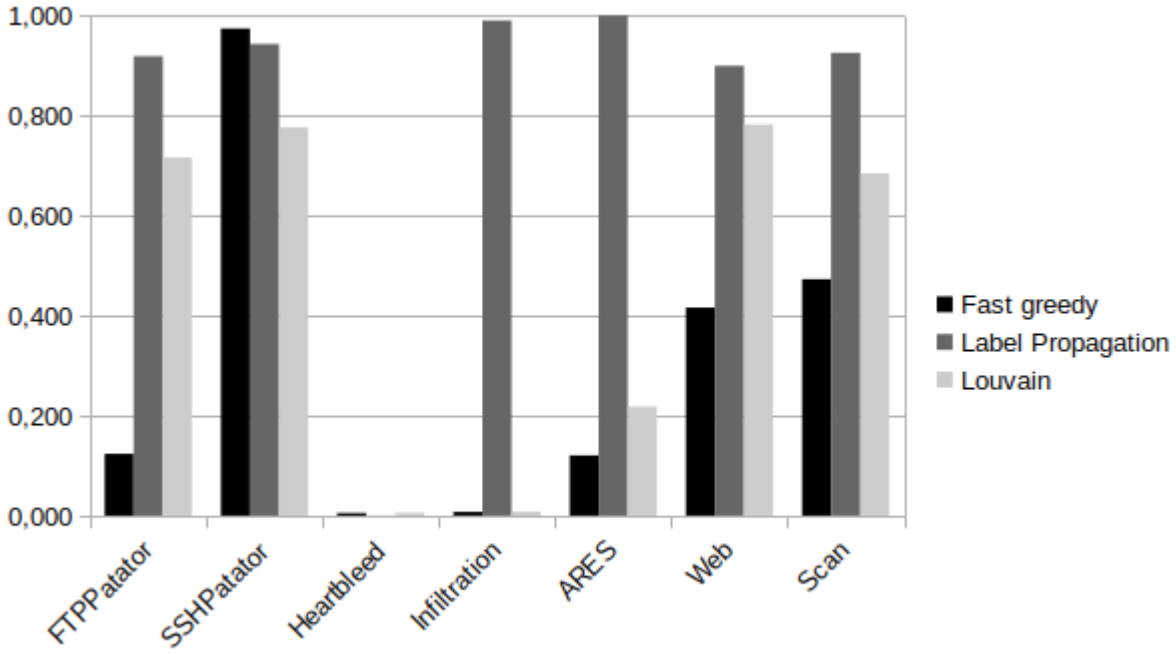

Figure 4.2 – Precision per attack's type for different community detection algorithm

For Precision, the Label Propagation algorithm performs well on all types of attack (precision greater than 0.9) except for FTPPatator and Heartbleed. Moreover, it is the only algorithm that performs well on Infiltration and ARES attack. The Louvain algorithm performs well on FTPPatator, SSHPatator, Web attack, and Portscan. The Fast Greedy algorithm only performs well on the SSHPAtator attack. The Heartbleed attack is the only attack for which no algorithm has shown good results. Indeed, only 29 edges of the graph are related to this attack out of the 132.646 edges representing network connections that take place during the attack. The Louvain algorithm was the most precise in selecting 4.852 edges, i.e., 3,6% of the total edges of the graph.

For Recall, the Louvain algorithm performs well on all types of attack (recall greater than 0.8) and is only outperformed by the Label Propagation algorithm for the Web attack and the PortScan and by the Fast Greedy algorithm for the Infiltration attack. The fact that these attacks are massive explains why the Label Propagation algorithm shows globally better results than

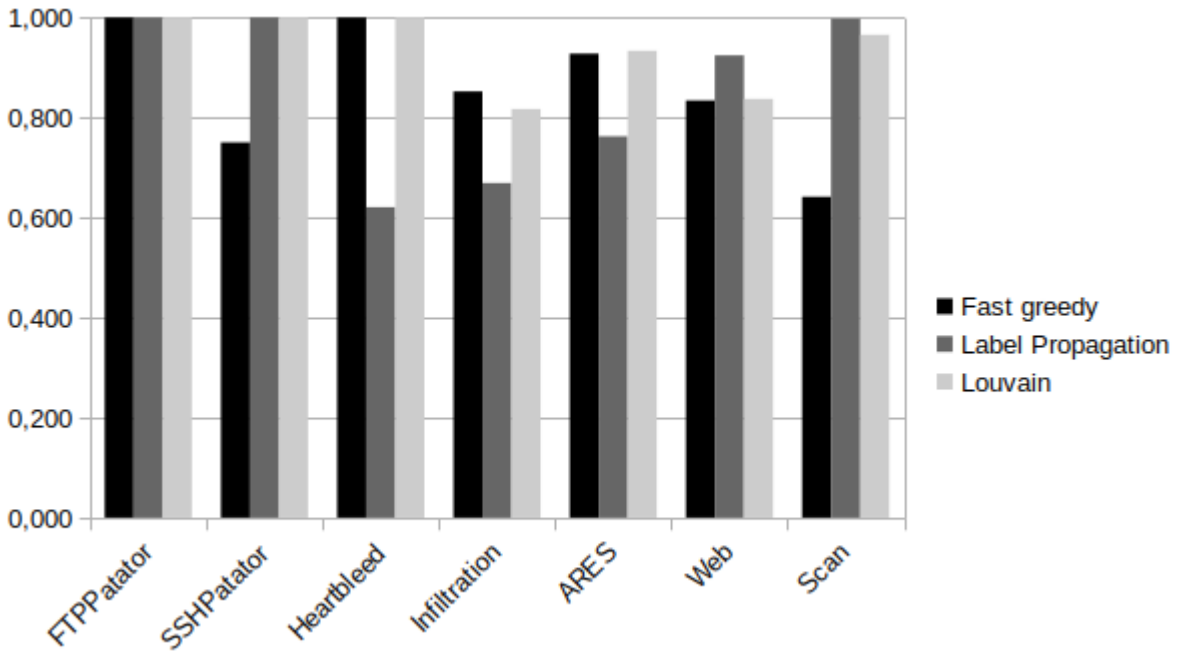

Figure 4.3 – Recall per attack's type for different community detection algorithm

Louvain. However, the Label Propagation shows only average results for the Heartbleed attack (0.621) and Infiltration (0.669). The Louvain algorithm is thus a better choice to consider multiple types of attack.

We note that the fact that we represent the data as a graph brings an additional advantage as it allows us to graphically present results to the analyst. Hence, in Figure 4.4, we show, as an example, the communities selected with the Louvain algorithm. Blue lines correspond to the edges having the *attack* feature set to '1', i.e. correctly selected edges. Red lines correspond to the edges with the *attack* feature set to '0', i.e, the erroneously selected edges. The displayed subgraphs show that the objects build upon the same attack are densely connected. Brute-force (ftp and ssh), web, botnet, and port scan attacks can easily be identified in the set of selected edges while they would have been obfuscated in the whole graph composed of millions of edges. The infiltration attack and the Heartbleed attacks are more difficult to identify in the communities selected by the Louvain algorithm. We surrounded the part representing these attacks with a black rectangle. In Section 4.3.2, we explain why such a result is observed, focusing on the Heartbleed attack.

4.3 Discussion

In this section, we propose to analyze the results obtained in the previous section along two axes: the relevance of the results according to the chosen method and the relevance of the results according to the type of attack that is observed.

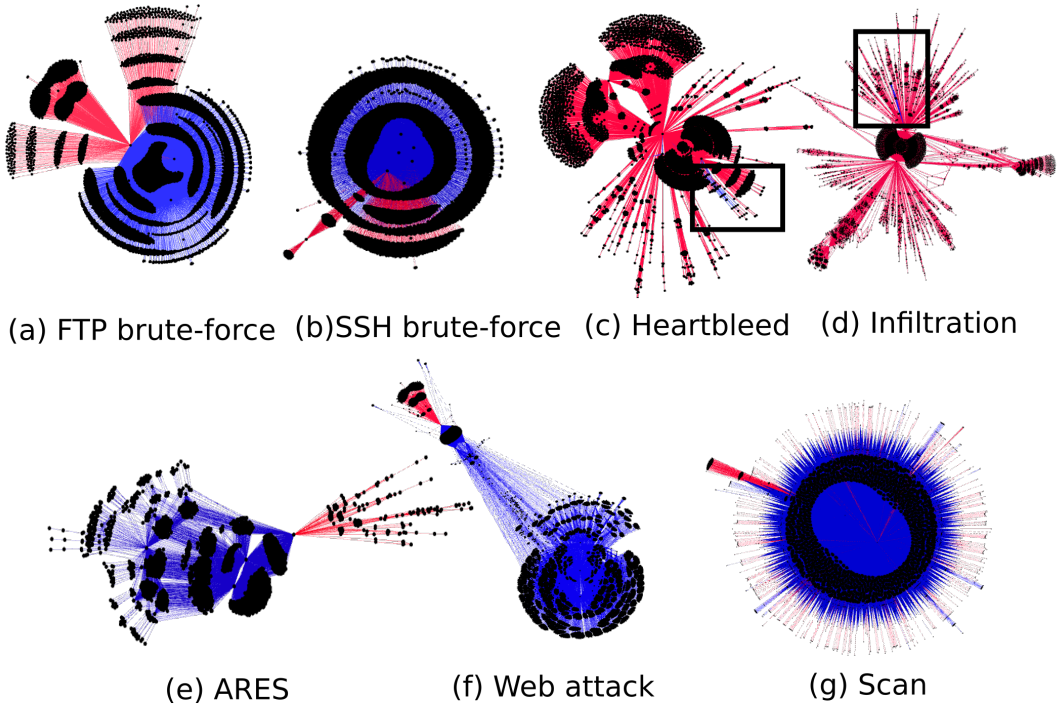

Figure 4.4 – Communities built by Louvain algorithm for various types of attacks. Blue lines corresponds to correctly selected edges (attack-related) and red lines corresponds to wrongly selected edges (normal)

4.3.1 Relevance of the results according to the method

The Fast Greedy algorithm obtains the worst Precision score since it tends to partition graphs into big communities [24]. However, in the security field, attacks are often exceptions compared to normal traffic. Even for massive attacks such as Scan or Brute Force, the partitioning of the Fast Greedy algorithm is not adapted and never reaches a Precision score of 0.5. More surprisingly, even if the communities that are selected are big, the Recall is also the smallest. In particular, the Fast Greedy algorithm gives bad results for Scan and SSH Brute Force while the structure of the corresponding sub-graphs corresponds well to the definition of a community. Blondel *et al.* [24] found that the partitioning quality of Clauset *et al.* for different types of graphs is medium. We obtain the same result here for massive attacks.

Generally speaking, the Label Propagation method obtains the best results for Precision and Recall. However, the Louvain algorithm has a Recall greater than 0.8 for every attack type. To expose on an example the difference between the partitioning of the algorithms, we focus on the Infiltration attack, where the Label Propagation obtains a Precision score of 0.99 whereas the Fast Greedy algorithm and the Louvain algorithm obtain values less than 0.01. Meanwhile, the

Label Propagation method obtains the worst Recall score compared to the two other methods.

In Figure 4.5, we compare the communities selected for the Infiltration attack by (a), the Fast Greedy algorithm, (b), the Louvain algorithm, and (c), the Label Propagation algorithm. Blue lines corresponds to correctly selected edges (attack-related) and red lines corresponds to wrongly selected edges (normal). The rectangle indicates the part of the sub-graph containing the attacks when they are not clearly visible.

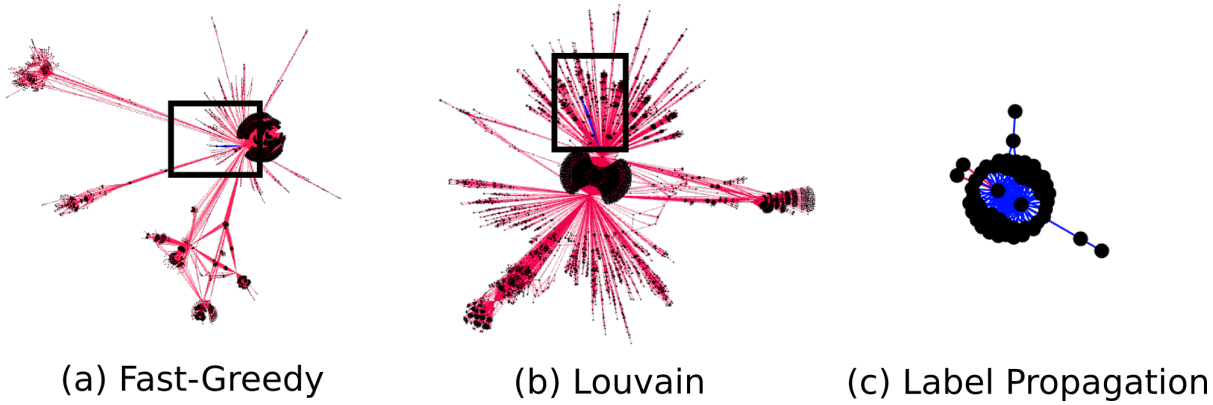

Figure 4.5 – Communities build by three different community detection algorithm in the case of Infiltration attack. Blue lines corresponds to correctly selected edges (attack-related) and red lines corresponds to wrongly selected edges (normal)

In this experiment, the Fast Greedy method selected 16.224 edges while the Louvain method selected 14.193 edges and the Label Propagation method only selected 96 edges. Visually, we can distinguish several sub-structures within the community selected by the Fast Greedy algorithm, while the one selected by the Louvain algorithm seems more densely connected. The Label Propagation algorithm community is very dense but more than 47 edges corresponding to the Infiltration attack have not been correctly selected.

In forensic analysis, when the attack is silent, it may be preferable to have a little more data to assess the context of the attack and compare it with normal traffic. In addition, analysts have one or more indicators within the community to guide them in their analysis. Community detection can thus allow the pre-selection of data to reduce the amount of data to be analyzed. Conversely, an analyst can start from a smaller community and then, using neighborhood queries, expand his or her research.

In brief, the Label Propagation algorithm is a better choice when the goal is to eliminate a large number of false positives and the Louvain algorithm is a better choice when the goal is to not miss any attacks. Of course, these results are related to the data that we used. They will therefore have to be confirmed by the same study on other data. As both methods run only in a few minutes, both methods can be proposed to the analyst.

4.3.2 Relevance of the results according to the type of attack

Figure 4.4, represents the communities selected by the Louvain algorithm for all types of attacks. Blue lines corresponds to correctly selected edges (attack-related) and red lines corresponds to wrongly selected edges (normal). The rectangle indicates the part of the sub-graph containing the attacks when they are not clearly visible. In five out of seven cases (FTP Bruteforce, SSH brute-force, ARES, Scan, and Web attacks), the attack forms a substructure that is quite distinct from the rest of the data, thus validating our hypothesis that attacks can form communities isolated from the rest of the graph. However, this hypothesis is not verified in the case of Heartbleed and Infiltration attacks.

We focus here on the Heartbleed attack, for which none of the methods succeeded in highlighting the security objects related to the attack in a significant way. In Figure 4.6, we see that 29 edges, out of the 4.857 retained by the Louvain algorithm, are related to the Heartbleed attack. The entire graph contains 132,646 edges, therefore the reduction factor between the selected sub-graph and the complete graph is 27. The figure shows that there is one central node in the community and that this central node is one of the security objects related to the attack. The fact that the attack is concentrated on a few edges and that one of its nodes has a high centrality measure means that the subgraph cannot be isolated in one community.

By contrast, for massive attacks like Bruteforce or Scan, there is a high concentration of nodes around one or two nodes. For example, the Port object with the port_value '21' for the FTP bruteforce attack is central for all nodes related to a connexion attempt on this port. Moreover, despite the fact that this port is common to many other connections that are considered normal, its combination with a source IP address makes it possible to isolate brute-force attempts from the rest of the graph.

In all cases, the community detection approach allows us to reduce the number of events to consider for forensic analysis. For massive events, the attack forms a recognizable community. For silent attacks, the structure does not allow by itself to isolate the attack subgraphs but it constitutes the first step to focus on a specific part of the graph.

4.3.3 Limits of the approach and prospects for improvement

In the following, we summarize the limitations of the community detection approach and propose prospects for improvement.

The main hypothesis of this chapter is that attacks are dense sub-graphs isolated in the global graph. We showed that the structure alone can isolate large attacks such as scans, DoS, or brute force attacks. It also allows isolating more discrete attacks forming isolated structures. However, discrete attacks such as Heartbleed or the Infiltration remain more difficult to isolate if the SOs that represent them are also present in normal events.

To overcome this first limitation, two strategies can be considered: the first is to recursively run community detection until a subgraph manageable for forensic analysis is found. The second is to adjust the parameters of the algorithm to control the size of communities. We indeed use the default parameters for all algorithms but some adjustments can be made to control how the

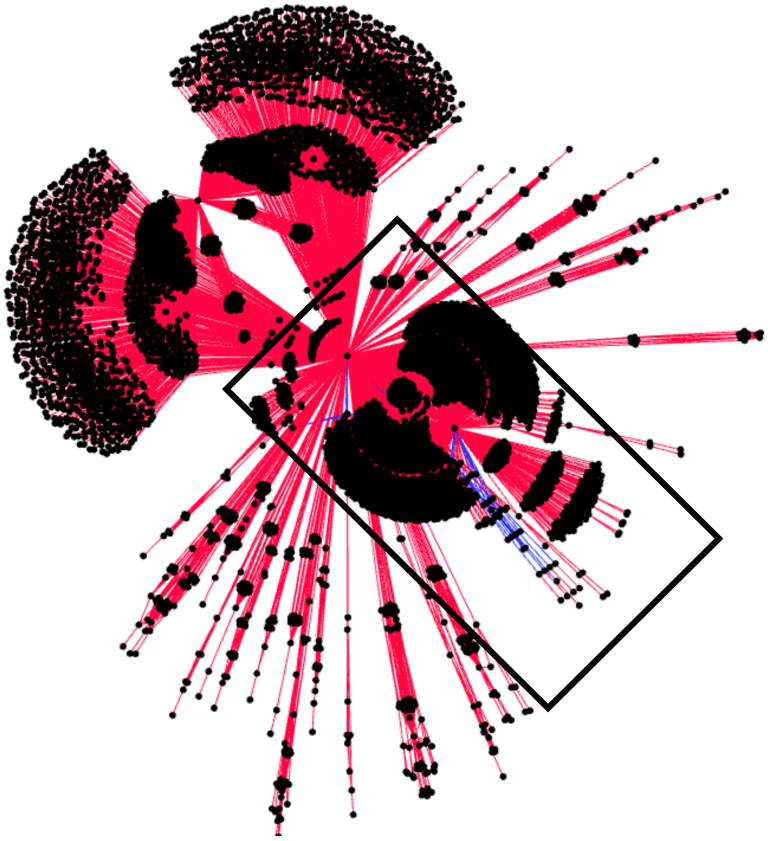

Figure 4.6 – Community selected by the Louvain algorithm for the Heartbleed attack. Blue lines corresponds to correctly selected edges (attack-related) and red lines corresponds to wrongly selected edges (normal)

partition is made. However, both approaches require actions from the analyst to control the parameters. In the first case, he or she must stop the iterations at the right time, and in the second case, he or she must determine the size of the communities.

A second limitation is that no selection criteria allows to select the communities most likely to contain objects related to an attack among all the discovered communities. The strategy we adopted is to select communities containing indicators of compromise or intrusion detection system alerts. However, this assumes that alerts or indicators have already been detected by third-party systems.

The two approaches suggested above to overcome the first limitation do not solve this second limitations. This thus brings us to a third strategy. The community detection approach only considered so far the structure of the graph without considering the content of the graph, i.e. the attributes of nodes and edges. In the next chapter, we will be interested in the properties of each

object in order to group the objects into communities not only using the structure of the graph but also by focusing on the properties of each node. In addition, we will use the structure and content of the graph to detect anomalies and thus eliminate the need for indicators or alerts.

Conclusion

In this chapter, we proposed a process based on community detection to discover security objects linked to an attack identified through an indicator of compromise. We implemented a prototype that discovers communities through different methods and evaluated each of them in the task of selecting relevant subgraphs for subgraphs analysis.

Experiments have shown that this approach allows identifying a very large part of the events related to a given attack, including potential hidden side-events.

Experiments also showed that the graph generation scales to large datasets including millions of events. As long as an analyst is able to discover an IoC, the proposed method offers a way to analyze the corresponding attack.

We finally showed in this chapter that our approach allows distinguishing information related to normal events from information related to an attack on certain conditions. However, the entry point for this distinction is an IoC. This is why our contribution is related to forensic analysis and not to intrusion detection. However, as the notion of community in graphs of SOs seems to clearly distinguish attacks from normal traffic, we propose in the next chapter to use our graph model to structure input data for an intrusion detection system. Our hypothesis is that graphs of SOs provide a rich description of what happened on the network, and that this rich description could be efficiently exploitable by machine learning mechanisms.

Chapter 5

Novelty detection

Contents

Introduction

To address the limitations of the community detection technique seen in the previous chapter, we investigate in this chapter anomaly detection methods applied to graphs.

These methods often built on supervised machine learning techniques, that require labeled data during the learning phase. However, security experts often do not have such labeled data sets from their event logs, and data labeling is expensive [2].

We thus propose to use an unsupervised anomaly detection technique called “novelty detection” based on an autoencoder. It is generally used when the amount of abnormal data available is insufficient to build explicit models for abnormal classes [117].

In this chapter, we propose a process to efficiently encode security object graphs so that an autoencoder can learn normal patterns and then detect abnormal activities. We also propose different strategies to compute an "anomaly score", i.e., a score that defines how much actual data differ from data learned in the learning phase.

This chapter thus presents the following contributions:

- A method to efficiently encode graph-structured data into values suited to a machine learning algorithm.
- An autoencoder that detects anomalies on graph-structured data with different strategies.
- Experimental results on the CICIDS2017 dataset, showing the effectiveness of our unsupervised method compared to supervised methods.
- Experimental results on the CICIDS2018 dataset, showing the effectiveness of our method compared to other deep learning approaches.

The rest of the chapter is organized as follows. Section 5.1 presents the way we encode both the content and the structure of security object graphs. Section 5.2 presents the building of our novelty detector based on an autoencoder and the detection strategies we chose. Section 5.3 presents the results obtained during the experimentations we performed on the CICIDS2017 and CICIDS2018 datasets.

The experimental results on the CICIDS2017 dataset presented in this chapter were published at the international conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA2020) [87].

5.1 Encoding the graph for machine learning

Preprocessing is often the first step in machine learning techniques. It consists in preparing the data so that it can be used as input for the machine learning algorithm.

To the best of our knowledge, there does not exist a method to encode multi-attributes and heterogeneous graphs that could be considered as generically efficient. For example, an adjacency matrix is inefficient for large graphs. It also carries no information on nodes and edges. In our case, the encoding method must encode both the structure of the graph (i.e., the relations between the nodes) and the specific information associated with the nodes and the edges. Moreover, the result of the encoding should be of reasonable size while containing enough information to detect anomalies. Since there does not exist a single best method to encode our graph, we had to design one tailored to our specific case.

We first remind that our objective is to detect anomalies. A given SO can be linked to several events, normal or abnormal. An edge, by contrast, is only related to the event that led to its construction. Therefore, it is not so much the SOs themselves that can be abnormal (an IP address or a port are not abnormal *a priori*), but the edges that link the SOs together. Consequently, we chose to encode our graph by encoding each of its edges. To preserve the context of the event related to this edge, we chose the following pieces of information to encode an edge: the type of the edge, information about the source node and about the destination node, information about the neighborhood of the source node, and information about the neighborhood of the destination

node. It should be mentioned that, due to the design of our model, a security event cannot be represented, by construction, by a subgraph exhibiting a diameter greater than three. Indeed, the translation method that we defined to convert events to subgraphs never produces a subgraph that has a path between two nodes made of more than three edges.

We need a representation that takes also into account the structure of the graph and the types of the edges. Figure 5.1 illustrates the process to encode an edge in a vector that can be used as an input for an autoencoder. To encode the edge e in the center of the figure, we first encode its attribute *edge_type*. Then, we add the encoding of the source node s represented in blue and the encoding of the destination node d represented in red. Finally, we add the encoding of $Neighbors(s)$ contained in the blue oval and $Neighbors(d)$ represented in the red dotted oval. The vector at the bottom of the figure is the result of the concatenation of all these encodings.

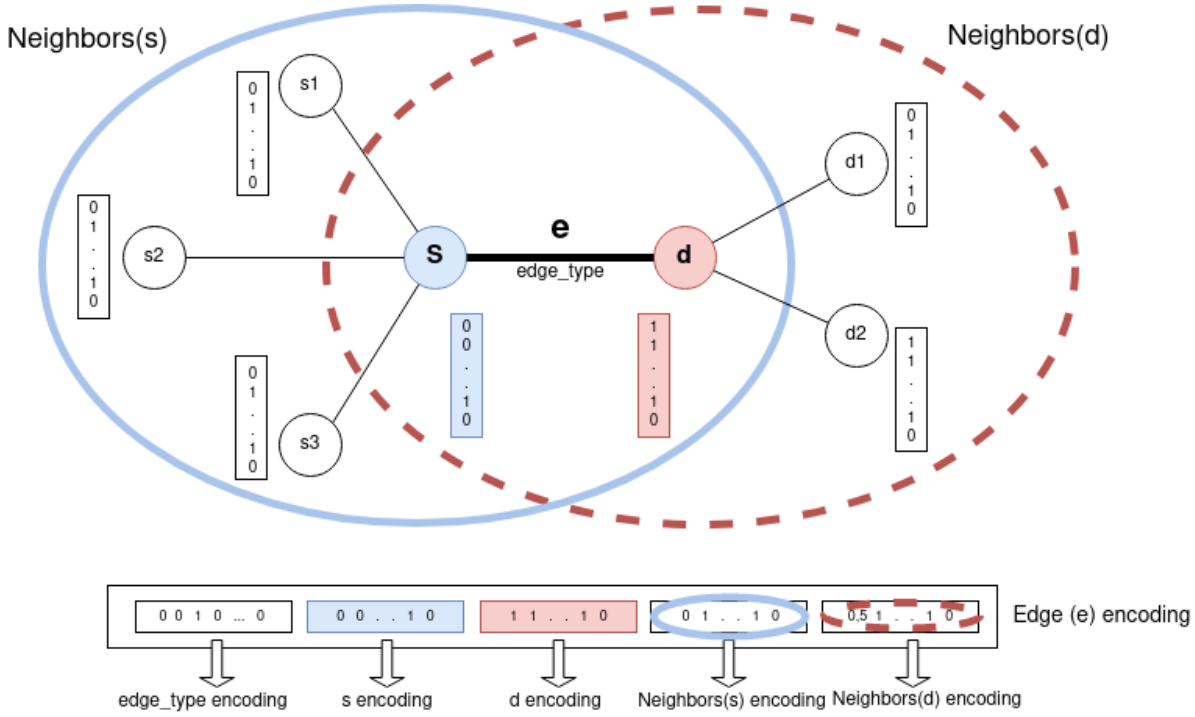

Figure 5.1 – Feeding the autoencoder: from graph to vector

In this section, for the sake of clarity, we first present the possible values of the various SO attributes we have to handle, how we encode them into categories, and finally how we create the vector corresponding to a given edge.

5.1.1 From SO attribute values to categories

Our graph exhibit different kinds of attributes that can have either categorical (version numbers, protocol types, etc.) or continuous (essentially size or duration) values. Anomaly detection

requires to encode these two types of attributes in a unified way (see section 5.2). Therefore, categories must be determined for each attribute, even for continuous ones.

For each *categorical attribute*, we count the number of occurrences of each category, for example the number of times the value "tcp" appears for the attribute "protocol". For single-value attributes such as port_value, the number of occurrences is by construction always equal to one, since we create only one Port SO for a given port number. Single-value attributes are therefore distinguished by counting the number of edges of the node carrying it. Indeed, we consider that the more a node with a given value is linked to other nodes, the more significant this value is for determining the normal behavior of the information system. In both cases, we sort values of attributes in descending order and keep the N most represented values that account cumulatively for 90% of the total number of occurrences or number of edges. Indeed, we consider that values present less than 10% of the time are not significant for determining the normal behavior of the information system. They must be discarded to avoid overfitting. We name categories the remaining values. If we have more than 20 categories, we only keep the 20 most represented categories, so that the vector we will have to build remains of reasonable size.

To build categories from *continuous attributes* values, considering intervals (e.g. [0:10[, [10:20[, etc.) is not an option because this would not take into account the statistical distribution of values and would not be useful for the autoencoder. Therefore, we categorize the continuous data according to the distribution of the attribute values. Since data samples do not necessarily follow a usual probability law, but a law whose density function is a mixed density, we use the classical Gaussian Mixture Model (GMM), assuming that the values of the attributes follow a mixture of a finite number of Gaussian distributions. It has been shown that GMM gives a good approximation of densities [56]. GMM has also already been used for anomaly detection [121, 12].

Two methods exist to infer the Gaussian equation and classify the data. The first one, the expectation-maximization algorithm (EM), is the fastest algorithm for learning mixture models but it requires to define the number of Gaussians components to infer [39]. The second method uses a variational inference algorithm [9]. It does not require to define the number of components but it requires hyperparameters that might need experimental tuning via cross-validation. Therefore, we chose the EM algorithm to control the number of Gaussian and hence control the number of dimensions of our vector, allowing to associate one dimension to one component. The number of Gaussian distributions is determined by the classical Bayes Information Criterion (BIC). The method consists in successively computing mixtures of Gaussians in increasing numbers and choosing the one with the lowest BIC. In practice, we do not mix more than eight Gaussians, because we found in our datasets that the BIC is never significantly smaller with more than eight Gaussians. The result is a mixture of no more than eight Gaussians, which brings us down to a case with no more than eight categories.

5.1.2 Encoding attributes using categories.

Once we have determined all the categories for our dataset, we can encode the nodes as binary vectors. We proceed as follows.

Each attribute is decomposed into categories according to the process described above. For each node and each attribute, we distinguish three cases: either the node has the attribute and its value corresponds to one of the categories of the attribute, or the node has the attribute but its value does not correspond to one of the categories, or the node does not have the attribute. In the first case, the one-hot-encoding technique is used: for each node of the graph, we build a binary vector x of size $N + 1$, N corresponding to the number of categories. Each bit of this vector corresponds to a given category and is thus set to '1' if the attribute value of the nodes is of this category. It is set to zero otherwise. A last bit with the value '0' is added to this vector to represent the category "other". In the second case, each bit is assigned the value '0' and a last bit is added to '1' for the "other" category. Finally, in the last case, all the bits are assigned the value '0'. This method makes it possible to encode all the nodes in a uniform way, despite their heterogeneity. We build a vector for each attribute and we then concatenate all these vectors into a binary vector corresponding to the encoding of the node.

5.1.3 Encoding the structure of the graph.

We encode an edge as a vector resulting of the concatenation of information on (a) the type of this edge, (b) the attributes of its source node, (c) the attributes of its destination node, (c) information about the neighborhoods of its source node and (e) information about the neighborhood of its destination:

- (a): there are 18 types of edges. For each edge, we encode its type using the same one-hot encoding technique that we use to encode the node's attributes.
- (b) and (c): we use the encodings of the source node and the destination node computed as showed previously.
- (d) and (e): for each source node and destination node, we select randomly 10.000 neighbors and compute the mean of their encoding vector. We choose 10.000 nodes because the mean does not change significantly above and this allows us to reduce the computational complexity.

Thus, considering that a l edge between s and d is of type e_{type}, that the s node has $N(s)$ neighbors and the d node has $N(d)$ neighbors, we randomly select 10,000 nodes in $N(s)$ and $N(d)$ in order to constitute a representative sample of the neighborhood $N(s)_{sample}$ and $N(d)_{sample}$. Recall that we already have the encoding of each of these nodes in the form of a binary vector, each having the same size.

We define, then $mean(\overrightarrow{enc}_{N(s)_{sample}})$ and $mean(\overrightarrow{enc}_{N(d)_{sample}})$ as the bitwise average of the vectors encoding each node of $N(s)_{sample}$ and $N(d)_{sample}$ respectively. We thus obtain a compact representation of the neighborhood of the node that is sufficient for the processing we intend to perform on the graph, i.e. the detection of anomalies. In this compact representation, each element of each vector takes a value between 0 and 1, but each element corresponds to a category of a certain attribute. Therefore, this vector of values between 0 and 1 gives an idea of the distribution of the categories in the considered neighborhood.

5.2 Novelty detection with an autoencoder

Novelty detection refers to the detection of data that was not seen during the learning phase, i.e., when the model was first built. This technique is generally used to detect anomalies when there is not enough data to build explicit models for abnormal classes. In the case of intrusion detection, thus novelty detection is especially well-suited since it is impossible to obtain a sample of all existing or future attack variants.

As it is classical for anomaly detection, our approach is based on the assumption that during the learning process the data contains no or very few traces of attacks and that attack-related data deviate sufficiently from the normal data to be detected as novelty.

In this section, we explain how autoencoders allows novelty detection. Then, we explain how we used an autoencoder to detect anomalies in a graph made of security objects.

5.2.1 Using an autoencoder for novelty detection

An **artificial neural network** (ANN) is an ensemble of "neurons" stacked in multiple layers: an input layer, one or several hidden layers, and one output layer, each composed of neurons. The goal of an ANN is to learn a specific function of x such as for a given training dataset x composed of n features such as $x = \{x_1, x_2, ...x_n\}$, we get a result $h_{W,b}(x)$. This result can be for example the probability that an image represents a cat. This function depends on two trainable parameters, the weight W and the bias b.

The training of an ANN is based on two steps that are repeated: the forward propagation and the backward propagation. In the forward propagation, the units of each layer are computed according to the units of the previous layer until we reach the output units, except for the first layer where the entry is the input vector x. In the backward propagation, batch gradient descent is used to update the weight W and the bias b according to the result of a loss function so that the model can improve. Forward and backward propagation steps are repeated several times or until the loss function reaches a minimum.

An autoencoder [83] is a specific type of ANN that learns a representation (encoding) of a set of pieces of data, typically for dimensionality reduction. To do this, the autoencoder learns a function that sets the outputs of the network to be equal to its inputs. An autoencoder is made of two parts: an encoder and a decoder. The encoder aims to compress input data into a low-dimensional representation, and the decoder tries to generate from the reduced encoding a representation that is as close as possible to its original input. The loss function $\mathcal{L}$ of an autoencoder is the average of the reconstruction errors on each dimension of the vector. It is defined as follows, with x the input of the autoencoder, $\hat{x}$ the predicted output and x_i (respectively $\hat{x_i}$) the i-th dimension of vector x (respectively $\hat{x}$):

$$\mathcal{L}(x, \hat{x}) = \frac{1}{n} \sum_{i=1}^{n} \|x_i - \hat{x_i}\|^2$$

Anomaly detection methods based on autoencoders use them to first learn the "normal" behavior by using dataset with benign data. Then, it is assumed that attacks will generate "abnormal"

observations that the autoencoder has never seen. Therefore, it will not be able to reconstruct the data. As a consequence, if the difference between the input and the output of the autoencoder, called reconstruction error, is above a given threshold, one can deduce that an anomaly is present in the data. The way the threshold is determined is of course a problem that we will discuss later.

5.2.2 Building the novelty detector

To detect anomalies in a graph of security objects, we first tailored the loss function to this objective. Then, we considered two detection strategies and determined detection thresholds accordingly. This is explained in the two following sections.

5.2.2.1 Tailoring the loss function

The vectors whose construction was explained in section 5.2 are used as input to our autoencoder. Recall that these vectors encode the following information: edge type, source node attributes, destination node attributes, source node neighborhood, and destination node neighborhood. The first three pieces of information are encoded by binary vectors while the last two are encoded by vectors whose components are between 0 and 1. In a similar case, the authors of [20] showed that it was desirable to have, on the one hand, a single encoding function (that makes it possible to take into account possible correlations between the different types of encoding) and, on the other hand, a decoding function that is specific to each type of information (binary vs. between 0 and 1) or specific to each piece of information. Here, keeping in mind that the output of our autoencoder will be provided to a security analyst, we have chosen to have a decoding function specific to each piece of information. Indeed, we need to be able to calculate a reconstruction error for each piece of information, that indicates to the analyst the potential anomaly thus revealed.

Our autoencoder therefore has five outputs and uses two types of loss functions: binary cross-entropy, a loss function adapted to binary values, and mean-square error, a loss function adapted to continuous values. The result is made of five error values between 0 and 1 to help the analyst diagnose possible anomalies. However, we still need to determine whether there is an anomaly. To do this, we calculate an overall error that is the sum of these five errors and raise an anomaly alert if this overall error reaches a certain threshold according to the strategies we define in the next section.

5.2.2.2 Detection strategies and according thresholds

While classical approaches seek to identify anomalies linked to events, our approach seeks to identify anomalies related to the links between objects. To refer to the case of anomalies on events, we have considered two strategies called *max* and *mean*.

The *max strategy* consists in considering as abnormal any event containing at least one link exceeding a detection threshold. The *mean strategy* consists in computing the mean of the reconstruction errors of all the links associated with an event. If this average exceeds our detection threshold, the event is considered abnormal. In other words, the first strategy supposes that the

anomaly is mostly carried by one link of the subgraph representing an attack event, whereas the second strategy assumes that the anomaly is carried by all the links of the same event. The *mean strategy* takes into account both strong local anomalies and the sum of weak signals. The *max strategy* could outperform the *mean* strategy only if a link with a low anomaly score compensates for the anomaly of a link with a high score.

In both cases, the detection threshold is set experimentally (see 5.3.2.1). Of course, the lower it is, the more alerts are generated, but the greater the risk of false positives. The analyst sets the value of the threshold according to context, lowering its value if it is more important for him or her not to miss any attack than to have to eliminate a large number of false positives.

Notice that to determine the influence of the neighborhood of an edge on the detection of anomalies, we also introduced a complementary strategy that consists in considering only the three following types of information: edge type, source node attributes and destination node attributes. In this configuration, we have only binary vectors. The autoencoder has therefore three outputs, one for each type of information, and uses only one type of loss function, the binary cross-entropy function.

In the following, the first strategy taking into account the five types of information including the respective neighborhoods of the source and destination nodes is called the *neighborhood strategy*. The strategy taking into account only the first three types of information without the neighborhood is be called the *simple strategy*.

5.3 Implementation and experimental results

This section presents our implementation choices, experiments, and analysis. We first describe the technologies we used, the datasets, and the evaluation criteria in Section 5.3.1. We then show how to carefully choose a threshold value for the anomaly detection in Section 5.3.2.1. Finally, we compare the results obtained by our approach to other approaches based on supervised anomaly detection algorithms (in Section 5.3.3) and on deep learning algorithms (in Section 5.3.4).

5.3.1 Experimental setup

As for the community detection implementation seen in Chapter 4, we chose the Python language and used the Gremlin API [127] for the construction of the graph from the event logs and the manipulation of the graph. In addition, we used the Python Keras library for the implementation of the autoencoder.

We used a Janusgraph database with an external index backend, Elasticsearch, and a Cassandra storage backend to store the graph data. We chose these technologies for scalability as they are adapted to large graph databases. Experiments were performed on a Debian 9 machine with 64 GB of RAM.

5.3.1.1 Choice of the datasets

Our experiments are based on two datasets: CICIDS2017 [131] and CICIDS2018 [1].

The CICIDS2017 dataset is presented in Section 3.2. Recall that this dataset contains five days (Monday to Friday) of mixed traffic, benign and attacks. The CICIDS2018 dataset was generated at the Canadian Cybersecurity Institute. It is made of ten pcap files encompassing about sixteen millions of events. It contains ten days of mixed traffic, benign and attacks such as DoS (Slowloris, Hulk, Goldeneye), DDoS, BruteForce, XSS, SQL injection, infiltration, and botnet activities. For both datasets, the normal traffic was generated using the CIC-B-Profile [131] system, which can reproduce the behavior of 25 users using various protocols (FTP, SSH, HTTP, HTTPS, and SMTP). Attacks were executed using classic tools such as Metasploit and Nmap.

The CICIDS2017 and CICIDS2018 datasets are the most recent ones that model a complete network configuration with a wide variety of components. The protocols in the capture (e.g., HTTP, HTTPS, FTP, SSH) are representative of the protocols that are used in a real network, and a variety of common attacks are covered allowing to evaluate our method on realistic datasets. These datasets are also labeled, allowing to quantify the efficiency of the method. Furthermore, they have been used in previous work on anomaly detection, allowing us to compare our model with state of the art approaches. CICIDS2017 has been used mostly for supervised anomaly detection approaches [131, 1], while CICIDS2018 has been used on a study on deep learning methods for anomaly detection [48].

As in the two previous chapters, we used the Zeek IDS tool to generate log files from the capture files. As the CICIDS2018 dataset is large, we randomly selected 10% of the dataset (that is 1.600.000 network connections in total) to perform our evaluation and built our model on 200.000 network connections to be consistent with the study of Ferrag *et al.* and thus to compare our approach with the approaches presented in this study.

5.3.1.2 Structure of the autoencoders

The structure of the autoencoder depends on the strategy used (*neighborhood* or *simple* strategy) as well as on the dataset, as both have an impact on the size of the input layers.

As for the strategy type, we used either an input vector containing only the information relative to the type of edges, the attributes of the source node, and the attributes of the destination node, or the whole vector containing in addition the information relative to the attributes of the neighborhood of the source and destination nodes.

Dealing with the datasets, the diversity of the network communication influences the number of attributes: the more varied the communications are, the more attributes will be needed to describe them and the larger our encoding vector will be.

For example, the structure of our autoencoder for the *simple* strategy on the CICIDS2018 is depicted in Figure 5.2. The sizes of both the input layer and the output layers (20+2*276=572 neurons) come from the sizes of our vectors (recall that the output vector should be equal to the input vector). The number of neurons in each layer and the number of hidden layers was

1. `https://www.unb.ca/cic/datasets/ids-2018.html`

determined by experimentation, trying different values looking for a minimum value for the reconstruction error. The autoencoder has three hidden layers: the diversity of the SOs in the graph leads to very diverse encoding and thus this number of hidden layers is suited for learning complex relations between the different bits of the vectors. The intermediate layer between the encoder and the decoder has a size of 125: since the input vectors are sparse, we chose a little number of neurons for this layer. We choose a number of epochs (the number of iterations of the forward and backpropagation phase) of 100 as experiments show that the reconstruction error did not decrease significantly for a larger number of epochs. We choose the Adam optimizer with a learning rate of 0.001 to back-propagate the reconstruction error as it is well-adapted when more than one hidden layer is used.

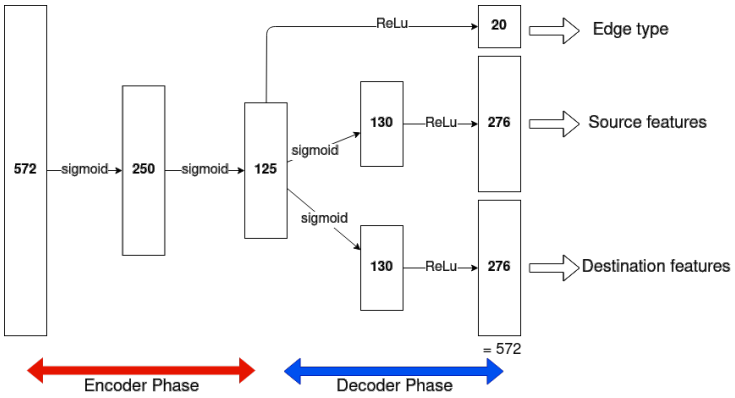

Figure 5.2 – Structure of the autoencoder for *simple* strategy applied to the CICIDS2018 dataset

The structure of the autoencoder for the *neighborhood* strategy on the CICIDS2018 is depicted in Figure 5.3. The sizes of both the input layer and the output layers are 1124 neurons (20+4*276=1124 neurons). We added one hidden layer and doubled the size of the intermediate layer compared to the autoencoder of the *simple* strategy to take into account the new input layer size.

Similarly, we created two autoencoders for the CICIDS2017 dataset. The autoencoder for the *simple* strategy has an input layer of size 678 (20+2*329=678 neurons) and 3 hidden layers and the autoencoder for the *neighborhood* strategy has an input layer of 1336 (20+2*329=1334 neurons) and 4 hidden layers. We adapted the size of the hidden layers proportionally to the size of the input layers to have a structure similar to the CICIDS2018 autoencoders.

To train the CICIDS2017 autoencoders, we used the data captured on Monday for the learning phase, as it is entirely normal. Since there is no full day without attacks in the CICIDS2018 dataset, we used 12 one-hour samples containing no attacks. These two datasets are synthetic. The creators have injected attacks at precise and documented moments. An attack-free sample can therefore be found. We should mention that being certain that a sample from a real-life data does not contain malicious events is still an open issue.

CICIDS2017 and CICIDS2018 are arguably two of the most realistic and reasonably large

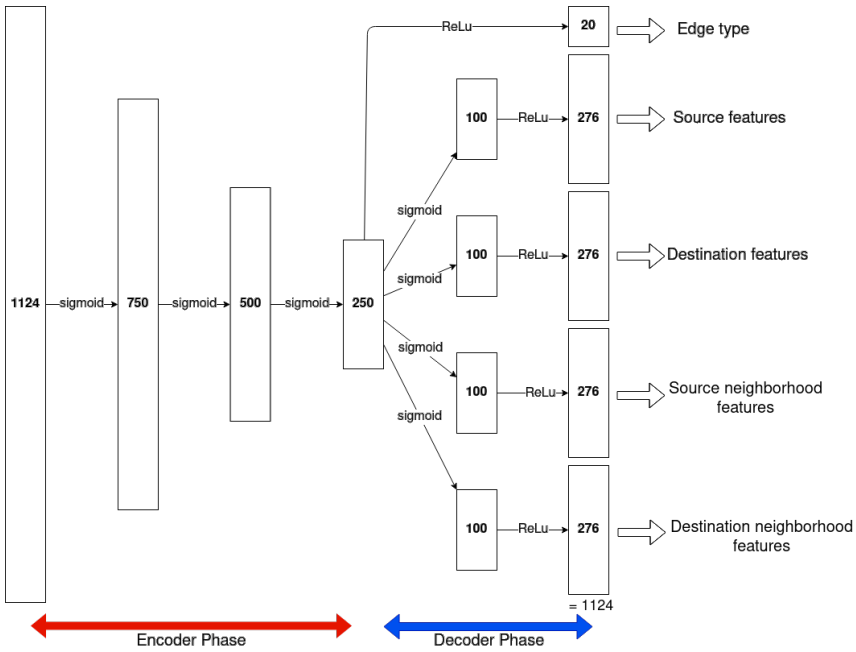

Figure 5.3 – Structure of the autoencoder for *neighborhood strategy* for the CICIDS2018 dataset

synthetic datasets. Nevertheless, CICIDS are labeled datasets. This allowed us to not consider the events generated by the attacks during our learning phases. In reality, of course, we would have unlabeled data potentially containing (fortunately rare) unidentified attacks. These attacks would therefore enter into the learning process, weakening the constructed model. We believe that our approach, based on an autoencoder, on the one hand, not considering the rarest categories (attribute values), on the other hand, allows to overpass this difficulty. Autoencoders indeed learn a general model, not taking into account particular cases such as the (rare) attacks. Future work on a learning dataset with a low attack proportion would nevertheless be necessary to validate this hypothesis. We unfortunately did not have enough time during this thesis to perform the related experiments, nor to define what would be a "small proportion of attacks". These aspects are left as future work.

For our experiments, the CICIDS datasets are split into training sets and validation sets with a validation split of 0.1. This allows us to validate the model on unseen data and thus prevent overfitting. Depending on the various parameters we have, the learning phases took about five hours. In the second phase (anomaly detection phase), we used the whole datasets (Monday to Friday for the CICIDS2017 dataset and the ten days for the CICIDS2018 dataset) to evaluate the detection capacity of our approach.

5.3.1.3 Defining detection evaluation criteria

All the results presented in this chapter are related to events. These are processed by one-hour shifts. For each time slot, we build a graph, then the vectors to feed the autoencoder, and finally we evaluate the novelty of each vector. We thus define False Positives (FP), False Negatives (FN), True Negatives (TN), and True Positives (TP) as follows: FP are network connection events wrongly selected, FN are network connection events that do not appear in the selected objects but that are part of an attack, TN are network connection events not selected in the graph and that are not being part of an attack and finally, TP are network connection events correctly selected.

In addition to the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN), we evaluate our results through the following standard metrics: Precision, Recall, F1-score, False Positive Rate (FPR), True Negative Rate (TNR) and Accuracy. We already presented Precision, Recall and F1-score in Section 4.2.2.

TNR is the proportion of normal events correctly classified among all normal events.

$$TNR = \frac{TN}{FP + TN}$$

FPR the proportion of normal events incorrectly classified among all normal events.

$$FPR = \frac{FP}{FP + TN}$$

Accuracy is the number of events correctly classified (as an anomaly or as normal events) divided by the total number of events.

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

5.3.2 Comparison of the four strategies of sec2graph

To determine the most effective way to apply our approach to intrusion detection, we first need to compare the detection strategies defined previously. In this section, we therefore first propose a process to define the optimal detection threshold for each strategy on each dataset. Then, to validate the so defined values, we study the effect of the threshold values on the detection of various types of attacks. Finally, we compare experimentally the four possible strategies on both datasets to define the best strategy to use.

5.3.2.1 Defining an optimal threshold

In this section, we present the experiments conducted to determine the threshold value to be used as the detection threshold. As noted above, an analyst would set this threshold value according to context, lowering the threshold value if it is more important not to miss any attack

than to have to eliminate a large number of false positives. We consider that minimizing false positives is crucial.

To determine the value of the threshold, we proceed as follows: first, we will consider all the events during a time slot without attack, i.e., Monday for the CICIDS2017 dataset, and the 12 first one-hour slots without attack for the CICIDS2017 dataset. As we seek for the lowest possible false-positive rate, we determine with this data the rate of false positives according to the detection threshold. The curve in Figure 5.4 shows the evolution of the FPR as a function of the detection threshold for strategies *simple_max* (left) and *simple_mean* (right) on the CICIDS2017 dataset. A threshold of 0.0016 leads to an FPR of 0.44% for the *simple_max* strategy while a threshold of 0.0014 leads to an FPR of 0.42% for the *simple_mean* strategy. For both strategies, we see in the figure that the FPR does not decrease significantly when we increase the detection threshold above 0.0016 for the *simple_max* strategy and above 0.0014 for the *simple_mean* strategy. Also, increasing the detection threshold too much can induce a high false-negative rate.

We conclude that a threshold higher than 0.0018 for the *simple_max* strategy and a threshold higher than 0.0014 for the *simple_mean* strategy should be chosen.

Similarly, we determined an optimal threshold value of 0.004 for the *neighborhood_max* strategy and an optimal threshold value of 0.001 for the *neighborhood_mean* strategy. This threshold gives us respectively an FPR of 1.92% for the *neighborhood_max* strategy and an FPR of 1.05% for the *neighborhood_mean* strategy.

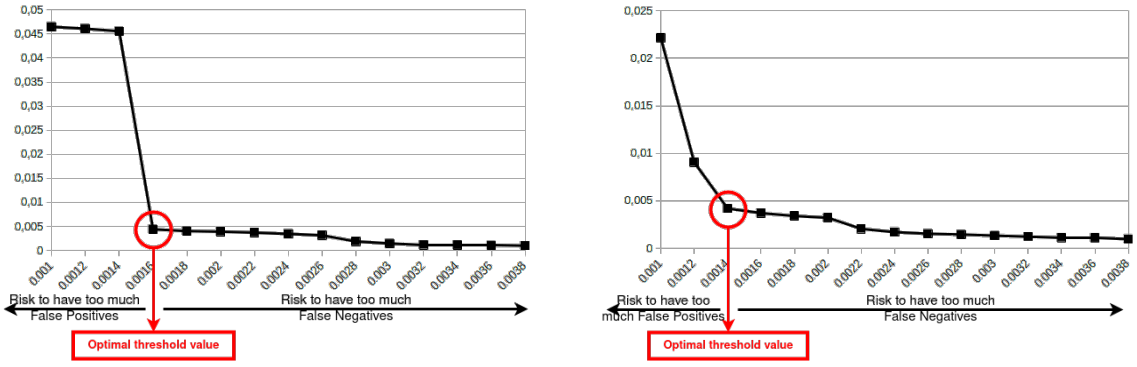

Figure 5.4 – False Positive Rate (FPR) according to the value of the detection threshold for *simple_max* strategy (left) and *simple_mean* strategy (right) on the CICIDS2017 dataset.

For the CICIDS2018 dataset, we follow exactly the same approach, for the four different strategies. We determined an optimal threshold value of 0.0018 for the *simple_max* strategy, 0.001 for the *simple_mean* strategy, 0.0036 for the *neighborhood_max* strategy, and 0.001 for the *neighborhood_mean* strategy. This threshold gives us respectively an FPR of 0.46% for the *simple_max* strategy, an FPR of 0.26% for the *simple_mean* strategy, an FPR of 0.46% for the *neighborhood_max* strategy, and an FPR of 0.34% for the *neighborhood_mean* strategy.

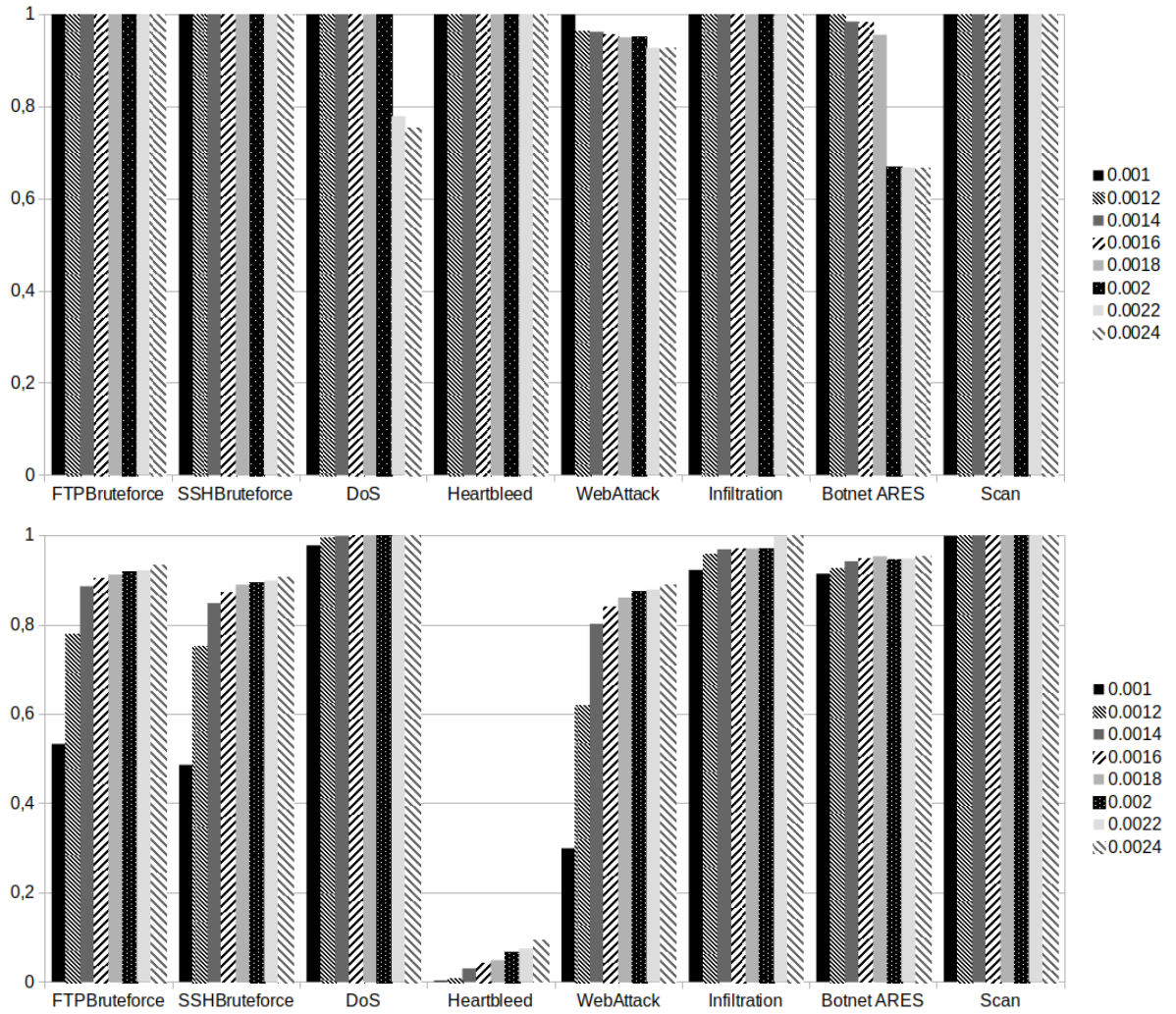

Figure 5.5 – Values of Recall (top figure) and Precision (bottom figure) for the range of variation of the threshold leading to a significant evolution of these values.

5.3.2.2 Validating threshold values

In the previous section, we determined threshold values for our 2 datasets and 4 strategies. Here we seek to validate the values obtained in each of these 8 cases. Indeed, the threshold values determined above allow to minimize false positives; we must now verify that the different types of attacks present in the datasets are indeed detected. We give here the example of the *simple_mean* strategy applied to the CICIDS2017 dataset, for which a threshold higher than 0.0018 should be retained. We only illustrate the validation process for this case, but the same applies to the other cases.

We consider the time slots during which each attack occurs. For example, the FTP Patator attack takes place on Tuesday from 9:20 to 10:20, so we consider the data between 9:00 and 11:00.

During these time slots in which the attacks take place, we want the maximum number of events related to the attacks to be detected as anomalies while keeping the number of false positives as low as possible. In other words, we want to maximize precision and recall. The Figure 5.5 gives the values of recall and precision as a function of the detection threshold for the different types of attacks in the CICIDS2017 dataset. We have made this threshold vary over the entire range for which this variation has a significant impact on recall and precision.

On the curve at the top of the figure, it can be seen that a quasi-optimal recall can be obtained (between 0.95 and 1) for a threshold value of 0.0018. On the curve of the bottom of the figure, we can also see that for this threshold value, the precision is between 0.86 and 1.00, excepted for the Heartbleed attack (0.048). Increasing the threshold further does not significantly increase the precision, but does significantly decrease the recall for the Botnet attack. We, therefore, conclude that we can set the threshold value to 0.0018.

In the case of Heartbleed, the low precision can be explained by the silent nature of this attack. Indeed, if we detect 100% of the events related to the attack for a threshold of 0.003, this represents only eight network connections compared to the nearly 93.000 network connections that took place during the attack.

5.3.2.3 Comparison of the different strategies

An optimal value for the detection threshold for each of the four strategies has been defined. It permits to compare each of these strategies on the CICIDS2017 and CICIDS2018 datasets.

We chose to compare the results of our four strategies against two criteria, namely the False Positive Rate(FPR, the rate of false positives that analysts have to deal with) and the Recall rate (the attack detection rate).

With our approach, an alert can be raised for each event contributing to an attack. As a result, an attack which generates a significant proportion of events relatively to the rest of the traffic will greatly influence precision. On the other hand, an attack that generates few events will have little influence on precision. The overall precision that can be observed is therefore dependent on the nature of the attacks present in the dataset. In order to make our evaluation as generic as possible and not depend on this volume, we do not take into account the precision in our comparison.

The results for the CICIDS2017 dataset are presented in Table 5.1 and the results for CICIDS2018 dataset are presented in Table 5.1.

Evaluation criteria Better if	**FPR (2017)** smaller	**Recall (2017)** greater	**FPR (2018)** smaller	**Recall (2018)** greater
Strategy				
Simple_Max	0.441	60.72	0.462	84.46
Neighborhood_Max	1.929	89.55	0.455	84.44
Neighborhood_Mean	1.049	99.66	0.338	100
Simple_Mean	0.420	99.98	0.257	100

Table 5.1 – Comparison of False Positive Rate (FPR) and Recall results (in %) for different strategies applied on CICIDS2017 and CICIDS2018 dataset

A good detection rate (Recall) is an essential criterion for the evaluation of our approach, as

we want to limit as much as possible the risks associated to non-detected attacks. When the Recall rate is almost identical for two strategies, the false positive rate allows to pick up the best one. Based on the Recall, the strategies *neighborhood_mean* and *simple_mean* obtain the best scores on the CICIDS2017 dataset with respectively 99.66% and 99.98%. However, the *simple_mean* strategy obtains the best FPR rate (0.420% compared to 1.049%). The best strategy to apply with the CICIDS2017 is therefore the *simple_mean* strategy.

For the CICIDS2018 dataset, the strategies *neighborhood_mean* and *simple_mean* obtain also the best scores with a score of 100% for both of them. The *simple_mean* obtains the best FPR rate (0.257% compared to 0.338%) even if they are close. The best strategy to apply with the CICIDS2018 is therefore, as for the CICIDS2017 strategy, the *simple_mean* strategy.

Note that the *neighborhood_max* and *simple_max* strategies give very similar results for the Recall value as well as for the FPR value on the CICIDS2018 dataset whereas, the *neighborhood_max* strategy clearly overpasses the *simple_max* strategy on the CICIDS2017 dataset, at the expense of a bigger FPR. Contrary to the two strategies *mean*, it is therefore difficult to conclude for the moment on the two strategies *max*. We have therefore carried out a more detailed comparison by considering these two *max* strategies, but also the two *mean* strategies, for the different types of attacks present in the 2017 and 2018 datasets.

Figure 5.6 presents the methodology we followed to compare, for each dataset, the Recall rate obtained with the different strategies according to the type of attack.

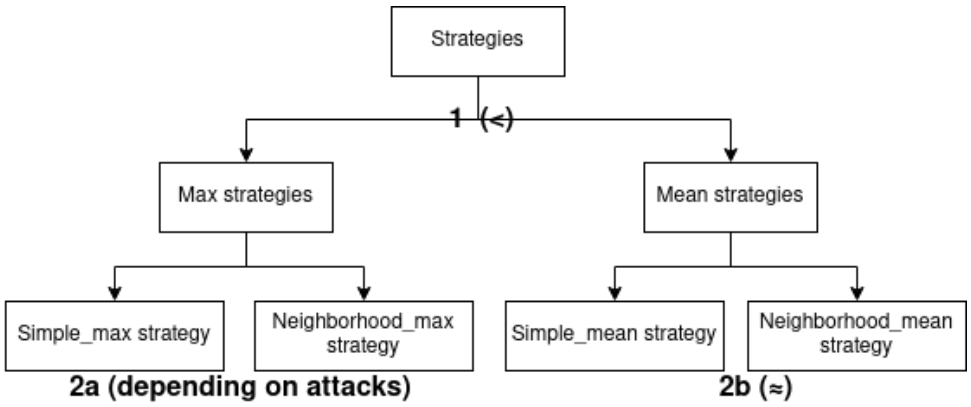

Figure 5.6 – Comparing detection strategies.

We first verify if *mean* strategies give better results than *max* strategies for each type of attack. The comparison **1** therefore focuses globally on the results of the two strategies *mean* and the two strategies *max*. Then, in a second step, we evaluate the impact of taking into account for the detection the neighborhood of the nodes in the graph, for each type of attack. We therefore compare, on the one hand, the *simple_mean* and *neighborhood_mean* strategies (comparison **2a**), on the other hand the strategies *simple_max* and *neighborhood_max* (comparison **2b**). The Figure 5.6 already indicates in a synthetic way the conclusions we draw from these three comparisons (in brackets next to the numbers of these comparisons): *mean* strategies give better results for all

types of attacks, except for the web attack contained in the 2017 dataset. Taking into account the neighborhood on *max* strategies gives results depending on the type of attack with the 2017 dataset. On the other hand, these results are equivalent for the 2018 dataset. Taking into account the neighborhood gives similar results for both *mean* strategies on the two datasets.

The results obtained for the three comparisons (1, 2a, and 2b) are given below: Figure 5.7 corresponds to the 2017 dataset and Figure 5.8 to dataset 2018. We discuss successively each of the three comparisons for each of the two datasets.

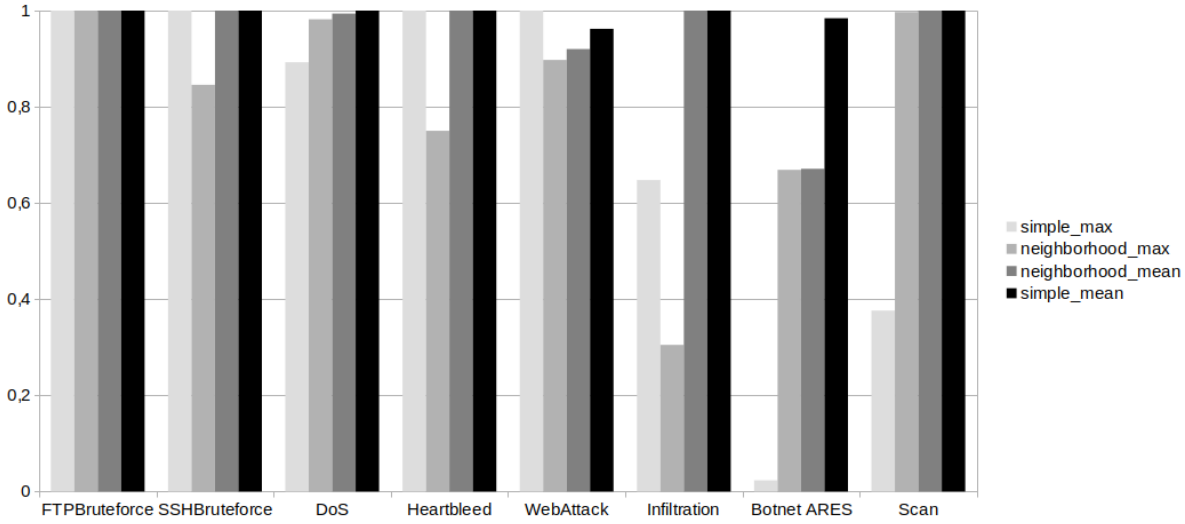

Figure 5.7 – Recall for different types of attack according to different strategies applied on CICIDS2017 dataset.

Comparison 1: *max* versus *mean*. On Figure 5.7, we see that the two *mean* strategies are always equal or better than the two *max* strategies except for the web attack where the *simple_max* strategy overpasses all strategies. We interpret this result as follows: *mean* strategies are more able at highlighting small novelties on several links, novelties which taken together constitute an anomaly. The *max* strategies, since they focus on the most abnormal links, are on the other hand only able to detect an important novelty on these links. For thewWeb attack of the 2017 dataset, we see that despite the fact that the *simple_max* strategy works better than others, the detection rates of the other strategies are greater than 90%. Our explanation is that some links corresponding to the web attack are very close to the learned model, thus lowering the average novelty score. However, *mean* strategies remain preferable in most cases and offer a detection rate for the web attack that is quite acceptable.

Comparison 2a: *simple_mean* versus *neighborhood_mean*. There is no significant difference between the results of the *neighborhood_mean* strategy and of the *simple_mean* strategy, except for the botnet attack where the *simple_mean* strategy is significantly better

than the *neighborhood_mean* strategy. Generally speaking, it seems that taking the neighborhood into account does not bring anything in terms of detection. However, taking the neighborhood into account does not degrade the results either. In the botnet case, taking the neighborhood into account seems to have decreased the novelty score for attack-related edges. We remind that the part of the vector encoding the neighborhood contains continuous values between 0 and 1, whereas the part encoding edge type, source node attributes and destination node attributes contains binary values. The neural network may need more layers or more neurons per layer to learn these continuous values. This would also explain why the FPR is generally higher for the *neighborhood_mean* strategy than for the *simple_mean* strategy. More neurons or more layers would however require more computing resources. Based on all these considerations, we conclude that the *simple_mean* strategy, which has excellent results, is better than the *neighborhood_mean* strategy for the CICIDS2017 dataset.

Comparison 2b: *simple_max* versus *neighborhood_max*. A comparison of the results obtained by the *simple_max* strategy and the *neighborhood_max* strategy does not clearly favour one strategy over the other. Indeed, the *simple_max* strategy is better for SSHBruteforce, Heartbleed, Web, and Infiltration attacks but worst for DoS, Botnet, and Scan attacks. This variability of the results can be explained as follows: if the reconstruction error of an attack is carried essentially by the neighborhood, the strategy *neighborhood_max* will be more efficient whereas if it is carried on the attributes of the source node, destination node or edge type, the strategy *simple_max* can then be the most efficient.

Actually, as the model depends on the dataset to learn the normal behavior and takes as input hundreds of attributes, it is difficult to explain these differences without analyzing the reconstruction error on each attribute for all these attacks. A more complete analysis of the attack-by-attack results taking into account the results for each dimension of the encoding vector would be necessary to decide on the best strategy to adopt. To realize such an analysis, Ribeiro [125] proposes a method consisting in varying the inputs of deep learning models to observe the evolution of the output results. We left this study as a future work.

We now come to the 2018 dataset (see Figure 5.8). We only represent in this figure attacks where there are differences between the strategies, namely FTPBruteForce, DoSSlowHTTP, DDoSLOICHTTP, DDoSLoicUDP, DDOSHOIC, BruteForceWeb, BruteForceXSS, and Infiltration [2].

Comparison 1: *max* versus *mean*. We observe on Figure 5.8 that as for the CICIDS2017 dataset, the two *mean* strategies are always as good or better than the two *max* strategies. Indeed, as already highlighted with the 2017 dataset, *Mean* strategies are more able to highlight small novelties on several links, novelties which together constitute an anomaly. In the case of

2. There is no difference in the Recall values for the following attacks: SSHBruteforce, DoSGoldenEye, DoSSlowloris, DoSHulk, SQLInjection, and Botnet. All strategies have a Recall score of 100% for these attacks.

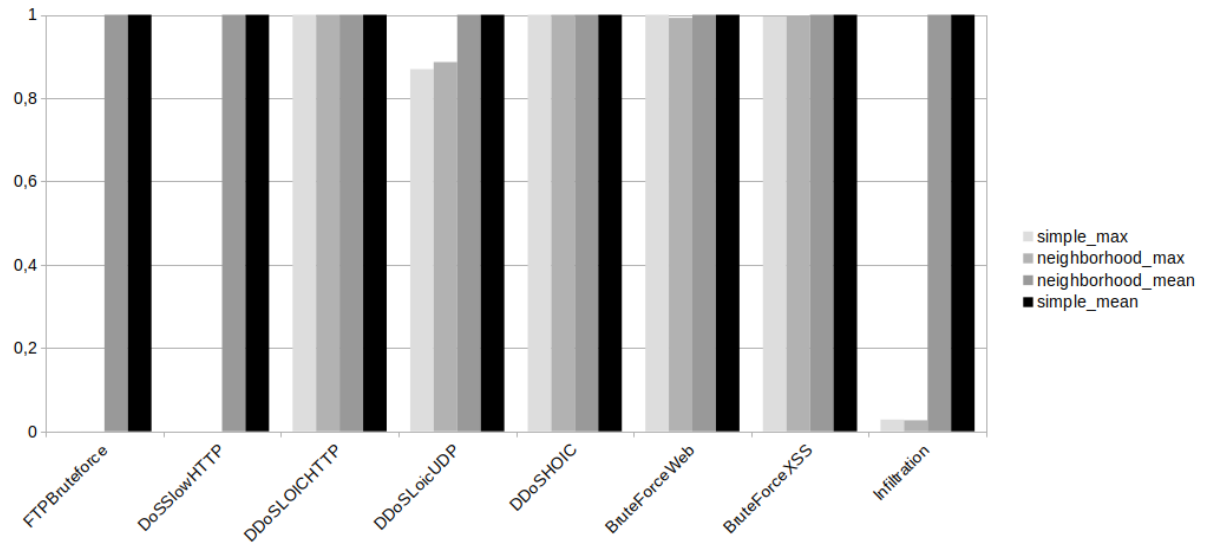

Figure 5.8 – Recall for different types of attack according to different strategies applied on CICIDS2018 dataset.

FTPBruteforce, DoSSlowHTTP, and Infiltration attacks, we are exactly in this situation and the *mean* strategies surpasses even more the *max* strategies.

Comparison 2a: *simple_mean* versus *neighborhood_mean*. There is no significant difference between the results of the *simple_max* and of the *neighborhood_max* strategy for the CICIDS2018 dataset. As said for the comparison using the 2017 dataset, taking the neighborhood into account does not bring anything in terms of detection. However, taking the neighborhood into account does not degrade the results either. Nevertheless, a smaller encoding being preferable, the strategy *simple_mean* has to be chosen.

Comparison 2b: *simple_max* versus *neighborhood_max*. Here also, there is no significant difference between the results of the *simple_mean* strategy and of the *neighborhood_mean* strategy for the CICIDS2018 dataset. Our interpretation is here the same than for comparison 2a, and for the same reason (a smaller encoding being preferable), the strategy *simple_mean* has to be chosen.

Conclusion on the six comparisons. To conclude on the best strategy to chose, we pick the *simple_mean* strategy for both datasets, as it gives the best results in terms of Recall for both cases. The *neighborhood_mean* strategy is very close to the *simple_mean* strategy but it is computationally less efficient as it takes a bigger vector as input than the *simple_mean* strategy.

It is important to note here that this evaluation and the conclusions we draw are relevant only in relation to the datasets we used, i.e., the 2017 and 2018 CICIDS datasets. It would be interesting to realize another evaluation on a dataset containing more silent attacks to see if the

neighborhood_mean strategy provides a better detection rate or if it induces a decrease of the detection rate.

5.3.3 Comparison with other work applied to the CICIDS2017 dataset

We saw in the previous sub-section that the *simple_mean* strategy was the one that, given the datasets used, gave the best results. In this section, we now compare the results of this strategy with those obtained by other researchers on the CICIDS 2017 dataset. Notice that comparisons on the basis of the 2018 dataset are presented in the next subsection.

In order to allow a relevant analysis of the different results, we have selected for our comparison three pieces of work [131, 1, 105] that use evaluation criteria close to ours. Other similar pieces of work exists but we have not retained it for our comparison because it presents poorer results relative to our evaluation criteria. In addition to this, [166] use the criterion of accuracy. Using this unique criterion alone does not give a good idea of the quality of detection. We have therefore also excluded this work from our comparison.

The authors in [131] compares the results of seven supervised classical machine learning algorithms applied to the CICIDS 2017 dataset. These algorithms, all supervised, are: K-Nearest Neighbors (KNN), Random Forest (RF), ID3, Adaboost, Multilayer perceptron (MLP), Naive-Bayes (NB), and Quadratic Discriminant Analysis (QDA). Ahmin *et al.* [1] compares the results of twelve classical or more recent classification algorithms: DecisionTree and rules, WISARD, Forest PA, J48 consolidated, LibSVM, FURIA, REP Tree, NaiveBayes, Jrip, J48, MLP, and Random-Forest. Since the last two algorithms were used in both studies, we retained for our comparison only the best results achieved. The authors in [105] proposes SU-IDS, a semi-supervised method (this is only a part of the dataset which is labelled) based on an autoencoder and a classification method. The proportion of data that is labeled varies from 0.5% to 100%. On our side, we use a totally unsupervised approach: the SU-IDS configuration closest to ours is therefore the one where the least amount of data is labeled, i.e. 0.5%. We thus compare ourselves with this configuration.

All these algorithms were tested using a dataset containing 80 features selected according to their relevance for the detection of attacks using the CICFlowMeter tool [131]. In the case of the first seven algorithms listed above, the authors trained their algorithms on a specifically chosen subset of the 80 attributes using a Random Forest Regressor. These attributes were chosen because they were most likely to help detect the attacks in the dataset and thus improved the performance of the algorithms for these specific types of attacks. In our case, we used all the features contained in Zeek event logs without making a prior selection according to their relevance for each attack to be detected. This is due to the fact that our objective is to measure the ability of the autoencoder to choose the most relevant features to represent a normal behavior in our dataset without targeting specific types of attack.

Table 5.2 gives a comparison of the classical learning machine algorithms listed above against our approach, sec2graph, with the optimal value previously determined for the detection threshold. The values in this table shows that sec2graph, although being an *unsupervised* approach, achieves

Evaluation criteria Better if	**FPR** smaller	**Recall** greater	**Precision** greater	**Accuracy** greater	**F1-score** greater
Algorithm					
KNN	-	96	96	-	96
RF	-	97	98	-	97
ID3	-	98	98	-	98
Adaboost	-	84	77	-	77
MLP	-	83	77	-	76
NaiveBayes	-	84	88	-	84
QDA	-	88	97	-	92
DecisionTree + Rules	1.145	94.475	-	96.665	-
WISARD	2.865	48.175	-	72.655	-
Forest PA	3.550	92.920	-	94.685	-
J48consolidated	6.645	92.020	-	92.688	-
LIBSVM	5.130	54.595	-	74.733	-
FURIA	3.165	90.500	-	93.668	-
REP Tree	4.835	91.640	-	93.403	-
NaiveBayes	33.455	82.510	-	74.528	-
Jrip	4.470	93.400	-	94.465	-
J48	5.040	91.990	-	93.475	-
SU-IDS (0.5% supervised)	5	93.68	-	93.68	
sec2graph (*simple_mean*)	0.420	99.978	98.534	99.668	99.251
Rank	1/12	1/19	1/8	1/12	1/8

Table 5.2 – Comparison of Recall, False Positive Rate (FPR), Accuracy, Precision, and F1-score results (in %) for supervised approaches of literature and sec2graph

better performances compared to those obtained by *supervised* or *semi-supervised* approaches. Given the strategy we have adopted to set the detection threshold, we achieve the best performance in terms of recall, with 99.978% of attack events correctly marked as abnormal (all attacks tested generate marked events). This corresponds to 119 network connections wrongly classified as normal (FN) out of 533.366 network connections related to an attack. With an FPR of 0,42% for the CICIDS2017, the sec2graph approach presents the best results. Indeed, of the 2.422.396 network connections, 1.881.096 were correctly classified as normal (TN) and 7.934 were wrongly classified as abnormal (FP).

Moreover, sec2graph is also the best in terms of precision, which means that the analyst will not be drowned by false positives: 98.534% of alerts are indeed true positives. In addition, contrary to the three pieces of work to which we compare ourselves here, we did not select attributes according to the type of attacks we wanted to detect. We thus believe that our approach should be more able to work for new kinds of attacks.

5.3.4 Comparison with other pieces of work applied to the CICIDS2018 dataset

At the time of this comparison, Ferrag *et al.* [48] are the only one dealing with the CICIDS2018 dataset. In this piece of work, they compare the results of seven supervised and unsupervised deep learning algorithms. The supervised algorithms used in this study are: Deep Neural Network (DNN), Recurrent Neural Network (RNN), and Convolutional Neural Network (CNN). The unsupervised algorithms used are: Restricted Boltzmann Machine (RBM), Deep Belief Network

(DBN), Deep Boltzmann Machine (DBM) and Auto-Encoder (AE). All these algorithms were tested on a dataset restrained to 80 features selected according to their relevance for the detection of attacks. Over the 15 450 706 network connections present in the dataset, 231 127 were used to build the models and 57 782 to test it. In comparison, we build our model on 200 000 network connections and tested it on 1 600 000 network connections.

Table 5.3 provides a comparison of the deep learning algorithms listed above against sec2graph, using the previously determined optimal value for the detection threshold and the *simple_mean* strategy.

	DNN	RNN	CNN	RBM	DBN	DBM	DA	**sec2graph** (*simple_mean*)
TNR for	96.915	98.112	98.914	97.316	98.212	96.215	98.101	99.743
RC for SSH-Bruteforce	100	100	100	100	100	100	100	100
RC for FTP-BruteForce	100	100	100	100	100	100	100	100
RC for Brute Force-XSS	83.265	92.182	92.101	83.164	92.281	92.103	95.223	100
RC for Brute Force-Web	82.223	91.322	91.002	82.221	91.427	91.254	95.311	100
RC for SQL Injection	100	100	100	100	100	100	100	100
RC for DoS-Hulk	93.333	94.912	94.012	91.323	91.712	93.072	92.112	100
RC for DoS-SlowHTTPTest	94.513	96.123	96.023	93.313	95.273	95.993	94.191	100
RC for DoS-Slowloris	98.140	98.220	98.120	97.040	97.010	97.112	97.120	100
RC for DoS-GoldenEye	92.110	98.330	98.221	92.010	97.130	97.421	96.222	100
RC for DDOS-LOIC-UDP	97.348	97.118	97.888	96.148	96.122	96.654	96.445	100
RC for DDOS-LOIC-HTTP	97.222	98.122	98.991	96.178	97.612	97.121	97.102	100
RC for Botnet	96.420	98.101	98.982	96.188	97.221	97.812	97.717	100
RC for Infiltration	97.518	97.874	97.762	96.411	96.712	96.168	97.818	100

Table 5.3 – Comparison of True Negative Rate (TNR) and Recall (RC) for each type of attack and for different methods (in %) and for sec2graph. Sec2graph ranks at the first place for each criteria

The values in this Table 5.3 show that the sec2graph approach with the *simple_mean* strategy offers a 100% detection rate for all types of attacks while having a false positive rate of only $100 - 99.743 \approx 0.25\%$, which is the best results among the panel of tested algorithms. Indeed, of the 1.773.761 network connections, 1.639.966 were correctly classified as normal (TN) and 4.225 were wrongly classified as abnormal (FP). We explain this result as follows: by taking into account more features in a graph structure, we are able to detect more attack-related events than the other approaches and reduce the FPR.

It should be mentioned that our auto-encoder takes as input for the *simple_mean* strategy a vector having 572 dimensions and three hidden layers, while the auto-encoder of Ferrag *et al.* [48]has only one hidden layer composed of 100 nodes. It takes about 47 minutes to Ferrag *et al.* to build their model, considering batch size of 1000 and 100 epochs. On similar condition, our autoencoder is more precise but it takes about three hours to learn the model. We consider that this difference of time in the construction of the model, is not a problem as it is realized only once a priori. Even in the case where the model would have to be rebuilt due to changes in the information system, we consider that periodically taking a few hours to build the model is not penalizing. Indeed, changes in an information system are not very frequent and therefore it will not be necessary to reconstruct the model too often. In addition, the autoencoder can learn on new data without starting from the ground up.

Conclusion

We proposed in this chapter an unsupervised technique based on an autoencoder to efficiently detect anomalies using four different strategies to compute the anomaly score, assuming that this dataset contains no attack. This approach can be applied to any dataset without prior data labeling.

Using the CICIDS2017 and CICIDS2018 datasets, we showed that graph structures representation of security data handled by an autoencoder gives better results than common anomaly detection methods (supervised and unsupervised), including deep learning ones.

However, some anomalies characterized by a temporal property are not well suitable for detection by our approach. An example of such an anomaly could be the presence of a given number of events of a certain type (e.g., file access) in a short time window (thus characterizing a high speed data leak). This type of anomaly can of course be important to detect. To further improve our detection results on these kind of anomalies, another kind of autoencoder (LSTM autoencoder [161]) could be used. LSTM autoencoders would allow to take temporal links between events in addition to logical links we already consider.

Another limitation of our approach is relative to evolving traffic. We tested our algorithm on cases where normal traffic does not evolve. In real environments, network activities, devices and behaviors may change over time. Since autoencoders allow for iterative learning, it would be possible to use new data to evolve the model and learn new behaviors. Nevertheless, to cope with evolving traffic generated by new activities, devices or behaviors, changing the number of layers and neurons of the autoencoder may be needed. More precisely, if the number of categories changes due to the evolution of the traffic, it would be necessary to start learning from scratch.

While dealing with millions of events each day is difficult for analysts in a SOC, our graph approach allows to consider a large number of events at once. Moreover, the use of an autoencoder allows a better interpretation of the results. We defined a global reconstruction error corresponding to the sum of reconstruction errors for each dimension of the input vector. By considering the error for each dimension, it should be possible to interpret more precisely the anomaly.

To improve the usability and interpretability of the results by a security analyst, techniques to understand what the model has learnt [125] could also be used. To that aim, data visualization is an interesting path to explore to help the analyst to accurately eliminate false positives or to discover global attack scenarios.

In the next chapter, we thus propose a visualization tool allowing to represent subgraphs (that could correspond to anomalies) in the global graph built from the log files.

Chapter 6

Graph visualization and exploration

Contents

Introduction

To provide the analyst with a presentation of relevant data that will reduce the time required to process alerts, we focused in this chapter on graph visualization techniques. The challenges are to deal with the amount of data that should be represented, to best communicate the data properties and to ease the interpretation phase.

In this chapter, we first propose an immersive visualization of the graph representation. This visualization highlights the relations between security objects and malicious events and/or IoCs. It provides a starting point for the analysts to explore the data and rebuild an attack scenario by following the links between the nodes. We also propose a *dendrogram* representation of security object attributes. Dendrograms are hierarchical representations that help the security analyst to define the criteria to aggregate security objects and to focus only on interesting parts of the graph.

This chapter presents the following contributions:

- A prototype for immersive visualization of the security object graph in 3D.
- An implementation of a dendrogram visualization that enables the analyst to aggregate the security objects of the graph.

The rest of the chapter is organized as follows. Section 6.1 presents the design choices and the prototype of the security object graph visualization. Section 6.2 presents the design and implementation of hierarchical clustering with dendrograms. Finally, Section 6.2.3 concludes this chapter.

The content of this chapter has been published at the International Symposium on Visualization for Cyber Security in 2017 (VizSec2017) [88] and in 2018 (VizSec2018) [86].

6.1 Security objects graph exploration with 3D graph visualization

In this section, we present the 3D graph visualization of the graph representation of security objects presented in Chapter 3. First, we present the design choices we made for our prototype, in particular regarding the graph layout. We then detail the choices made in terms of user interaction. More specifically, we address the problems of navigation in time and space. Finally, we present a prototype explicitly design for virtual reality devices.

6.1.1 The 3D graph visualization

The representation of a graph is not trivial. In this section, we first present the challenges with graph representations and the technical choices made in relation to our constraints and our objectives.

6.1.1.1 Challenges with graph representations

Automatically representing a graph can be described simply: given a set of nodes and a set of edges, compute the position of the nodes and the curve to be drawn for each edge. The arrangement of nodes and edges in a drawing affects its understandability, usability, cost-effectiveness, and aesthetics [144]. There are no well-defined criteria to evaluate a graph drawing algorithm, but generally the success of graph drawing techniques is measured by aesthetic criteria and by their computational efficiency.

The term aesthetic refers to criteria that evaluate some aspects of readability. Among the aesthetic criteria, it is commonly accepted that the representation must minimize crossing edges, distribute nodes equitably in space, make edge lengths uniform and that the symmetries that exist must appear in the visualization [53].

The size of the graph to view is a key factor in graph visualization regarding both aesthetic and efficiency. Comprehension and detailed analysis of data in graph structures is easiest when the size of the displayed graph is small. Large graphs pose several difficult problems. If the number of elements is large, it can compromise performance or even reach the limits of the graphical interface. Even if it is possible to display the elements of the graph, its readability and usability can be questioned if the display does not allow to distinguish nodes and edges. Moreover, displaying

an entire large graph may give an indication of the overall structure but makes it difficult to comprehend. Generally, drawing a graph containing more than a thousand nodes becomes very difficult [63]. Because the layout is so dense, interaction with the graphs become difficult and occlusions make it impossible to navigate in the graph and analyze it. Consequently, a first step in the visualization process is often to reduce the size of the graph to display. We proposed in Chapter 4 and Chapter 5 two methods based respectively on community detection and anomaly detection to reduce the size of the graph to display and analyze. It should be mentioned that we will present two other methods in this chapter to reduce the data to display, respectively in Section 6.1.2 and Section 6.2.

These issues of scalability, aesthetics and computation cost are all the more important for forensic analysis. The analyst must be able to easily see the substructures and follow the links between security objects. In addition, he or she must be able to interact with the data in order to display the relevant details.

6.1.1.2 The graph layout choice

Force-based graph drawing algorithms are the most used techniques [77] among graph drawing algorithms for large graphs because they do not require the graph to have specific properties such as planarity [1].

Force-based graph drawing techniques assigns forces among the set of edges and the set of nodes, according to their relative positions and then uses these forces to simulate the motion of the nodes and edges. Once the forces on the nodes and edges of a graph have been defined, the behavior of the entire graph is simulated as if it was a physical system: the forces are applied to the nodes, pulling them closer together or pushing them further apart. This is repeated iteratively until the system comes to a mechanical equilibrium state, i.e. when their relative positions do not change anymore from one iteration to the next. As a result, all the edges are of more or less equal length and there are as few crossing edges as possible. Graphs drawn with a force-based layout also tend to be more readable, and are able to display symmetries [53] that can be useful for forensic detection. For this reason, we used them to display our graph of security objects.

In our early experiments, we first tried to generate 2D graph representations with a force-based algorithm. However, we had to deal with the fact that numerous lines were crossing one another and that this representation was very difficult to understand. Indeed, our graphs are large and dense. As a consequence, we decided to create 3D graph representations using Unity [149], even if we were aware of the the well-known occultation problem they can cause. The force-directed graph drawing algorithm could easily be extend to a third dimension that gives more space and ease the problem of displaying large structures. Furthermore, the user can navigate to find a view without occlusions as we will see in Section 6.1.2.

While the force-based algorithm makes graphs more readable, it is expensive in computing time. However, the fact that the algorithm is based on successive iterations makes it possible to

1. Planarity is a property exhibited by graphs that can be displayed on a flat layout without having edges that cross one-another.

achieve an intermediate visual rendering: the graph is displayed, while the position of the nodes is not yet stabilized to allow the analyst to start his or her analysis and interpretation of the results. He or she can also stop the calculation at any time to interact with the data.

In our representation, nodes of the graph were first represented as colored sphere. To reduce occlusion, we then used wire-frame colored spheres instead of full spheres to represent the security objects. A color code was used to distinguish the type of each object. For instance, a blue node represents an IP address, a white node is used for a network connection, etc. The palette used is designed for being used by color-blind people [2]. Edges are represented as white straight-line between the nodes.

6.1.1.3 Example of security object graph visualization

In this case study, we use two datasets made of logs from Bro IDS that were generated in the Stratosphere Labs as part of the Malware Capture Facility Project. The objective of this project is to store long-lived real botnet traffic and to generate labeled netflows files.

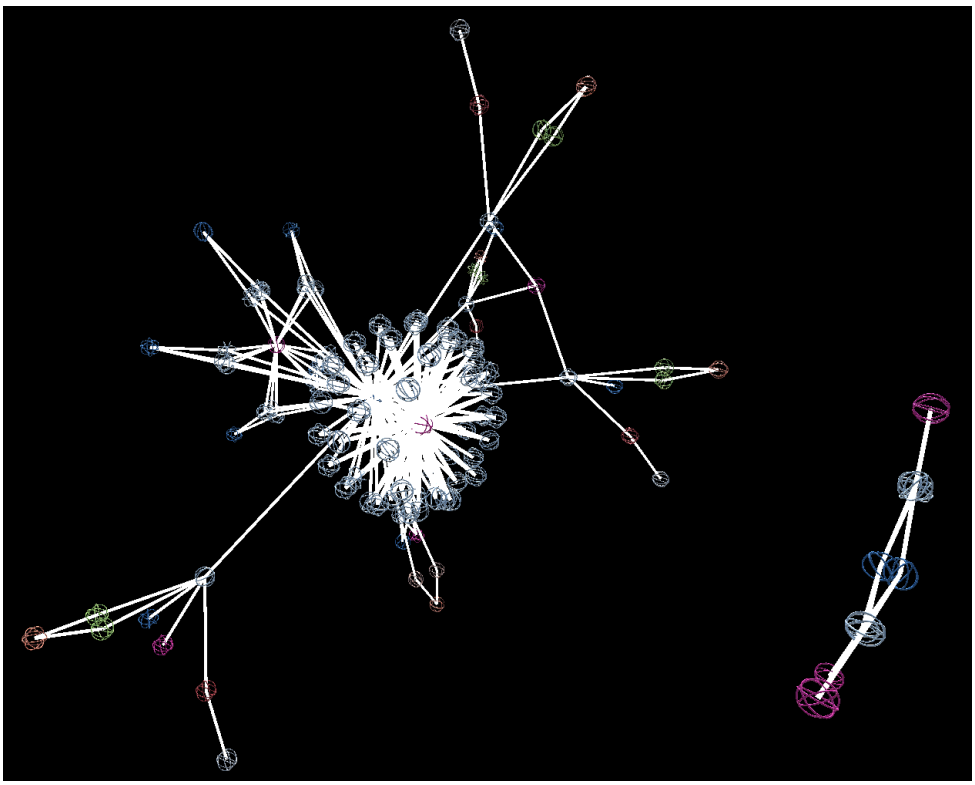

Figure 6.1 – Our representation of CTU-Malware-Capture-Botnets-254-1 dataset

The CTU-Malware-Capture-Botnets-254-1 dataset describes a possible Wannacry attack. The Wannacry attack exploits a samba vulnerability. It should also be mentioned that each piece

2. M. Krzywinski. 15-color palettes for color blindness: `http://mkweb.bcgsc.ca/colorblind/img/colorblindness.palettes.v11.pdf`, retrieved September 2020

of Wannacry malware was programmed to check for a given domain name and to stop when an answer would be obtained. In this capture, a port scan is performed but the vulnerability cannot be exploited because the targeted servers do not provide the required samba service. The capture lasted one hour. Bro output files were generated from this capture. The infected host is 192.168.1.123.

Figure 6.1 shows on the left the representation of the address scan on port 445/tcp. This port is represented in the middle. All rejected network connection attempts can be seen around the port security object.

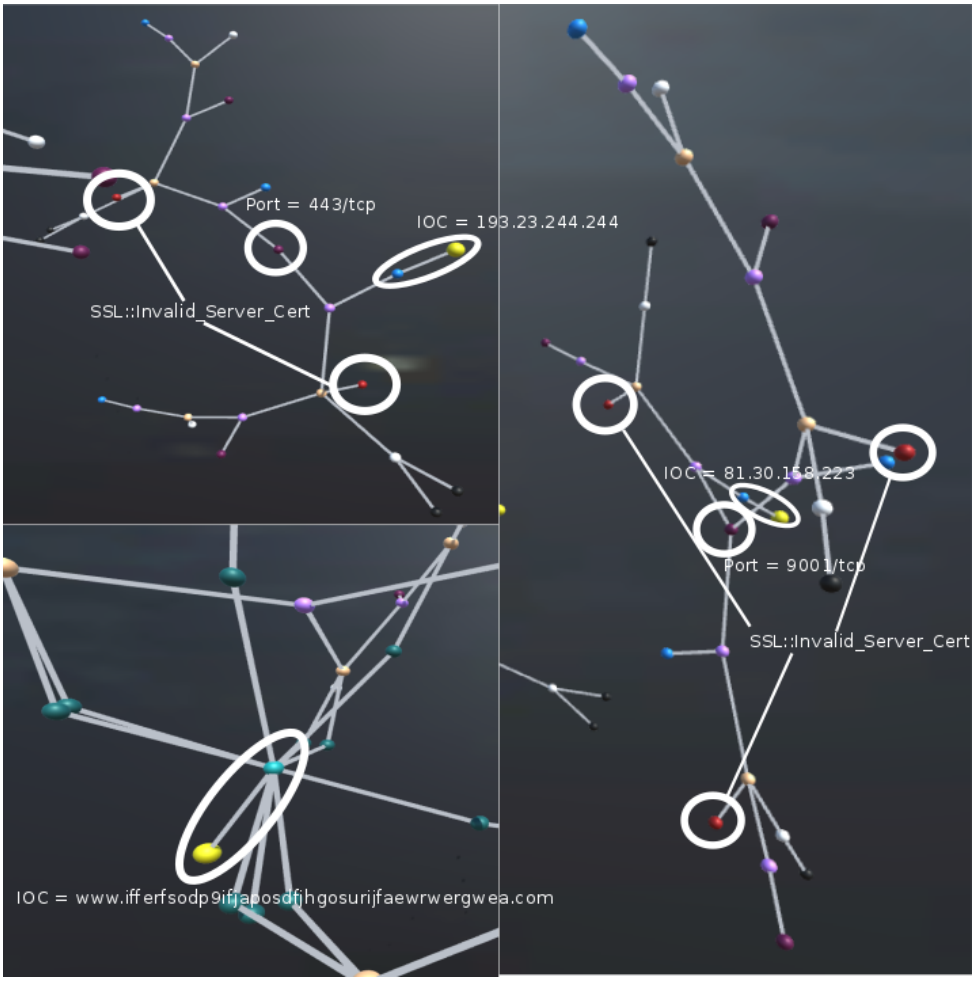

Figure 6.2 – Our representation of selected clusters in dataset CTU-Malware-Capture-Botnets-253-1. The white ellipses indicate Alerts and Indicators

CTU-Malware-Capture-Botnets-253-1 also describes a Wannacry attack. Due to a problem in the DNS server provider, the domain did not resolved the killswitch IP, and therefore the infection went on. The capture lasted 895 seconds. The infected host was 192.168.1.120. This capture used a MITM proxy on all TLS ports and a redirection to a honeypot on port 445/tcp.

This time, we have added external IoC data related to the Wannacry attack. We also used our community detection method to display only the clusters containing IoCs. In this capture, thanks to our proposal, we detected the killswitch `www.ifferfsodp9ifjaposdfjhgosurijfaewrwergwea.com`. We also discovered two malicious addresses, 193.23.244.244 and 81.30.158.223 that were associated to the Wannacry campaign. The representation also shows a scan on port 445/tcp. We focus in Figure 6.2 on clusters with the killswitch and IoCs. In the visualization, the points of interest are concentrated on three clusters. The first cluster concentrates two alerts and one IoC referring to an IP address, the second concentrates three alerts and one IoC referring to an IP address. The last one, but also the more dense contains only one IoC, the killswitch. This domain is directly linked with 9 `DNSResquest` objects. By following the paths coming from the malicious DNS requests, two more DNS requests coming from the same socket can be identified.

6.1.2 User interactivity

This visualization tool is to be used by security analysts. Thus, interactions have to be oriented toward their specific use-cases. In a first part, we tried to identify the main interactions required for the exploration and manipulation of the 3D graph environment and identify mainly the navigation through time and space.

6.1.2.1 Navigation through time

Our tool proposes three modes:

- The static mode.
- The dynamic mode.
- The vanishing mode.

The *static mode* does not take time into account. It shows the whole graph without taking into account the temporal aspects. It proposes an overview of the elements constituting the graph.

The *dynamic mode* allows to visualize the construction of the graph as events occur. This makes it possible to understand what happened and to deduce possible temporal causality between the events. The construction of the graph is based on the timestamp of the events. Each link generated from this event contains the timestamp attribute to know when it should appear in the view. New nodes and edges are added to the graph and linked to pre-existing nodes as they appear in the events. The positions of the nodes in the display space are then automatically readjusted. The user can choose the speed of appearance of the elements of the graph. The timestamp of the first event in the graph is used as the original timestamp. By default the speed corresponds to the real speed of appearance of the events: If ten seconds separate two events, the subgraph representing the first element will be displayed ten seconds later. The subgraph of the second event will appear either disconnected or linked to the first subgraph if the subgraphs contain common objects according to the construction principle seen in Chapter 3. The user can use the interface to increase the speed of events.

The *vanishing mode* also makes it possible to make the events disappear after a certain period of time. So if an edge corresponds to an event that occurred more than `x` minutes ago, `x` being user-defined, this edge disappears. At this time, if a node has no more links, it is removed from the display. This allows to keep only nodes linked to more recent events. This third visualization avoids cluttering up the graph, by progressively removing old elements.

6.1.2.2 Navigation through space

The force-based layout makes global graph analysis easier by distancing isolated sub-graphs and concentrating densely connected nodes. Highly connected data may represent normal behaviors or massive attacks while isolated sub-graphs can be anomalies that can be interesting to analyze. However, our first experiments showed that it was still mandatory to propose navigation through space.

Zoom and pan are traditional interactions in visualization. They are mandatory when large graph structures are explored [63]. They allow the analyst to adjust the desired level of detail. Zooming allows to focus on a sub-section of the graph and analyze the security objects and links one by one while pan allows to see the overall structure of the graph. Navigation also allows us to reduce the occlusion constraints related to the use of 3D. So even with large, densely connected graphs, it is possible to navigate and see every part of them.

The analyst can also click on any node to obtain more information on the related point of interest. The information displayed are the attributes of the nodes such as the address_value for an *IPAddress*, or the number of packets for a *NetworkConnection* for instance.

We took advantage of Unity to implement these features. The navigation through space was developed to immerse the analyst in the data. This immersion was further developed by using virtual reality.

6.1.3 Implementation as an immersive environment

3D graph visualization has significant difficulties. Perceptual and navigational conflicts are caused by the discrepancy of using 2D screens and 2D input devices to interact with a 3D world. Virtual reality helps solving this issue. Immersive environments help the user to understand and explore complex structures. Considering the benefits this could bring to graph exploration and analysis, we used virtual reality to create various tools for the user to explore and manipulate the data.

Inputs in immersive environments are often more limited than on a desktop setup (e.g., a traditional keyboard). Therefore, users must be offered a way to choose among the set of available interactions that he or she will be using. A system of carousel on the controller, shown at the bottom of Figure 6.3, gives the possibility top switch mode by sliding on the pad. The space navigation controls are however available at all time.

This new use-case-specific interactions have been developed considering the security analyst point of view and workflow:

- Repulse neighbors to isolate a particular object.

- Information display about targeted objects.
- Tracking nodes to rebuild a scenario.

Navigation in 3D is achieved either by head motion or virtual flying, following the direction of the controller. Simplified surrounding limits motion sickness that could be induced by smooth artificial locomotion. Another aspect of navigation is the scaling of the scene. By giving the possibility of changing the scale of the graph, the analyst can easily experience a complete view of the graph and recognize patterns or zoom on a specific zone to analyze a small cluster of objects. These two options, directly related to space navigation, are separated from the other tools, and are always available to the user, no matter the chosen interaction mode.

As our data are also temporal, an interaction mode on the carousel allows the analyst to control either the playback time, that can be negative if necessary, or directly change the position of the cursor on a timeline using the controller. When the analyst wants to stop and analyse the graph at a precise timestamp, a *freeze* option is available to stop time and movements induced by the force-directed layout. Stopping completely the graph this way allows the analyst to get points of reference, thus enhancing the navigation.

Figure 6.3 – Our Track Node tool: one panel to display the list of selected nodes, one to display the detailed information of one node, and visual feedback of a selected node in the background

The analyst interacts with the security objects by selecting nodes using a laser pointer attached to the controller. A first tool named *Information* on the carousel allows the analyst to get directly the information about a security object upon selection. All the information about the node is

displayed as text on a screen attached to the controller, and the analyst can scroll through it. An arrow pointing at the selected security object acts as visual feedback for the user. This is a very simple tool used to discover the data, get a first idea of both the structure and the events represented through the graph visualization.

Further study requires a more substantial tool. This second tool named *Track Node* allows the analyst to select several nodes that are enlarged in the graph and glow with a yellow light. The analyst is given two panels, one similar to the previous tool, displaying the information about a security object, and another to display the list of the selected security objects with a short description. He or she can scroll through the list and choose one of them. This security object will then have its detailed information displayed on the second screen, and the same arrow as the *Information* tool will point at it. This way, the analyst can select a list of relevant security objects and study their detailed information in order to understand the impact of a set of logged events on the actual attack. The *Track Nodes* tool is showed in Figure 6.3, with the two screens and a selected security object, that is larger and surrounded with a yellow light and a pointing arrow.

Such selection method can lack precision when selected nodes are far away from the analyst. If the high mobility offered by our navigation tools and the freedom of movement would be generally enough to solve this issue, the problem remains in the case of a crowded environment. To cope with this issue, the *Repulse* tool allows to increase and decrease the norm of the repulsion force acting on a pointed node, spreading the nodes and giving access to the potentially hidden ones.

We performed our experiments on a computer having the following characteristics: Intel i7-8700 CPU, NVIDIA GTX 1080 GPU and 32GB RAM. An HTC Vive headset was used for VR, and the tool was implemented using the Unity Editor 2017.1.1f1.

We identified several technical constraints: first, virtual reality requires a high and, more importantly, stable frame rate to ensure comfortable experience. It is generally accepted that our eyes need at least about 45 frames per second to get a fluid rendering. Second, the Vive controllers have a limited number of input buttons: a 1-axis trigger, a 2-axis touch pad, and 2 buttons. Third, usual head up display of information is difficult because of the Vive lenses optical effects on the border of the screen space.

The first challenge comes mainly from the force model used for the graph representation that has a computation complexity of $O(n^2)$ in the worst case. Therefore, given the amount of nodes we are working with, it requires a lot of resources. We adapted the data structure and used methods to limit the amount of computation. The graph data is parsed from a XML file, to create Unity 3D objects, designated as Node and Link. The Link objects contain the timestamps and the reference of the two linked Node objects. The rest of the information is stored in the Node objects; the color of the Node object will notably represent the type of security object. The force-oriented layout is separated into two forces. An attraction force is applied between two linked nodes, depending on the distance between these two nodes. The other force is a repulsion force applied by each node on each other, depending on the distance between them. It is to be noted that interactions between distant nodes are insignificant. Thus, a list of each node closest neighbors is stored as attribute of the Node object and updated regularly. Only the nodes in the list will be considered for the repulsion force computation applied on a node. This reduces the average-case computation

complexity and thus the amount of required computation, at the cost of a little more memory use. Finally, we use different threads for force computation and rendering. A slightly lower refresh rate in the force computation thread is not detrimental to the layout, and this separation ensures that it will not disturb the rendering. Eventually, we managed to respect the 45 frames per second criterion with about a thousand node.

The two last constraints, related to user interaction, were solved using the user interactions presented in the previous section. First, our implementation uses a carousel to change interaction mode, allowing to artificially multiply the controller inputs, solving the first challenge. Second, since the information display had to be attached to an object that the user can focus on, we chose to attach it to the virtual representation of the controller itself, solving the second challenge (cf. Figure 6.3).

6.2 Analyzing visual clusters

We have presented in Section 6.1 an interactive 3D representation of graphs of security objects. Following the use of this tool on several datasets, we found that the graphs were frequently made of "visual clusters" of similar security objects grouped around a common node. Visual clusters are interesting for two reasons:

- On the one hand, they may represent a normal behavior that is frequently repeated. Their analysis allows in this case to identify the characteristics of this normal behavior (i.e., the similar attributes of the nodes forming this cluster) to better identify them afterwards and eliminate them from the analysis, making it faster.
- On the other hand, they can represent massive attacks such as scans, DoS, DDoS or brute force attacks that should be analyzed.

Another advantage is that these visual clusters can easily be identified visually by the analyst. However, when a node has a lot of links and thus neighbors, visual clusters remain difficult to analyze.

To cope with this limitation, a possible strategy, as previously done, is to identify new sub-clusters among these visual clusters. These new sub-clusters, named "syntactic clusters" to distinguish them from the visual clusters, should be composed of nodes sharing the same values for a given subset of attributes. These syntactic clusters will both help the analyst to eliminate some groups of nodes corresponding to normal behavior, and to identify anomalies that will in turn be analyzed.

6.2.1 Identifying syntactic clusters

A good tool to identify syntactic clusters is hierarchical clustering [67], that works as follows:

1. Start with $k = n$ communities (every node is its own clusters, n being the number of security objects in the graph).
2. Compute the distance matrix (distance with each cluster).

3. Merge the two closest clusters according to the distance matrix.
4. Repeat steps 2 and 3 until all nodes are contained in one single cluster.

For this algorithm to work, it must be provided:

- A function to compute the similarity (and therefore the distance) between two clusters.
- A linkage function that defines the way the new attributes values for a new cluster are computed from the attributes values of the two clusters that were merged to create it.

To compute the similarity between two nodes, we use a distance measure defined by the number of dissimilar attribute values. The more two security objects have attribute values that are different, the more distant they are. We use Ward's linkage as the linkage function. It minimizes the variance of the clusters being merged and is well-suited for most applications.

6.2.2 Displaying syntactic clusters

To display the results of hierarchical clustering, i.e., our syntactic clusters, a dendrogram representation is especially suitable.

A dendrogram is a hierarchical tree representation that records the sequences of merges or splits. It also graphically shows the distance between its nodes in a efficient way (see Figure 6.4). By construction, the closer two leaves of a dendrogram are, the more similar the two objects they represent are. Non-terminal nodes of the dendrogram represent a cluster and each branch is a criterion of separation of two clusters. Such a dendrogram can be built for each node of the 3D graph from the characteristics of all the neighbors of this node. Our tool allows the analyst that focuses on a given node (e.g.,an IP address of interest, a node with many neighbors) of the 3D graph to build a dendrogram that corresponds to this node. A first strategy to identify an interesting node is to consider the ones with high degree. The dendrogram obtained for such a node allows the analyst to determine, among the neighbors of the node, those that can be usefully grouped together.

In Figure 6.4, a representation of a dendrogram built upon a visual clusters of 4846 security objects is displayed. The number displayed next to each node of the dendrogram corresponds to the number of SOs contained in the cluster. Starting from the root node on the left, we notice that the first separation clearly separates a single SO from the 4845 other SOs. This can be seen with the length of the edge going towards the single-SO cluster. The second separation indicates a much smaller difference between the two clusters. This indicates that the SOs of these clusters share some similarities. In order to see what are the common attribute values within the same cluster, the analyst can hover a node. On the figure, we can see for example that the cluster containing 111 SOs contains SOs that all have as attribute values the values displayed in the blue rectangle (e.g. conn_state_SF, history_other, proto_tcp, etc.).

6.2.3 Implementation as a web application

We designed and implemented a web application to explore our security-related data, following the principle of Wang *et al.* [157] which states that multiple views can help users understand

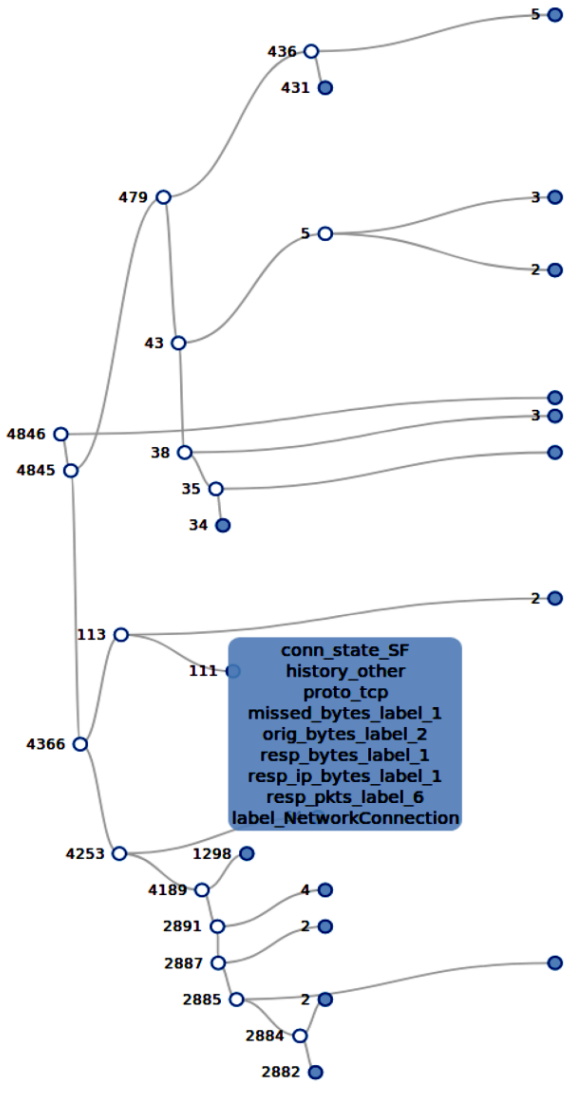

Figure 6.4 – The dendrogram built on the objects with the highest degree: the top branch groups all network connections linked to a network scan whereas the bottom branch groups unrelated network connections.

relationships among different data sets with multiple attributes.

Our tool is made of three views (A, B and C) represented in Figure 6.5.

The 3D-graph view **A** comes from our previous work (see Section 6.1.1). This first view allows the analyst to select the visual clusters of security objects to be analyzed by clicking on the central node of these visual clusters.

The visual clusters are then handled thanks to hierarchical clustering and the corresponding dendrogram **C** is displayed. The analyst can interact with the dendrogram by hovering over the nodes to display the attributes common to a set of nodes or by clicking on the nodes to display

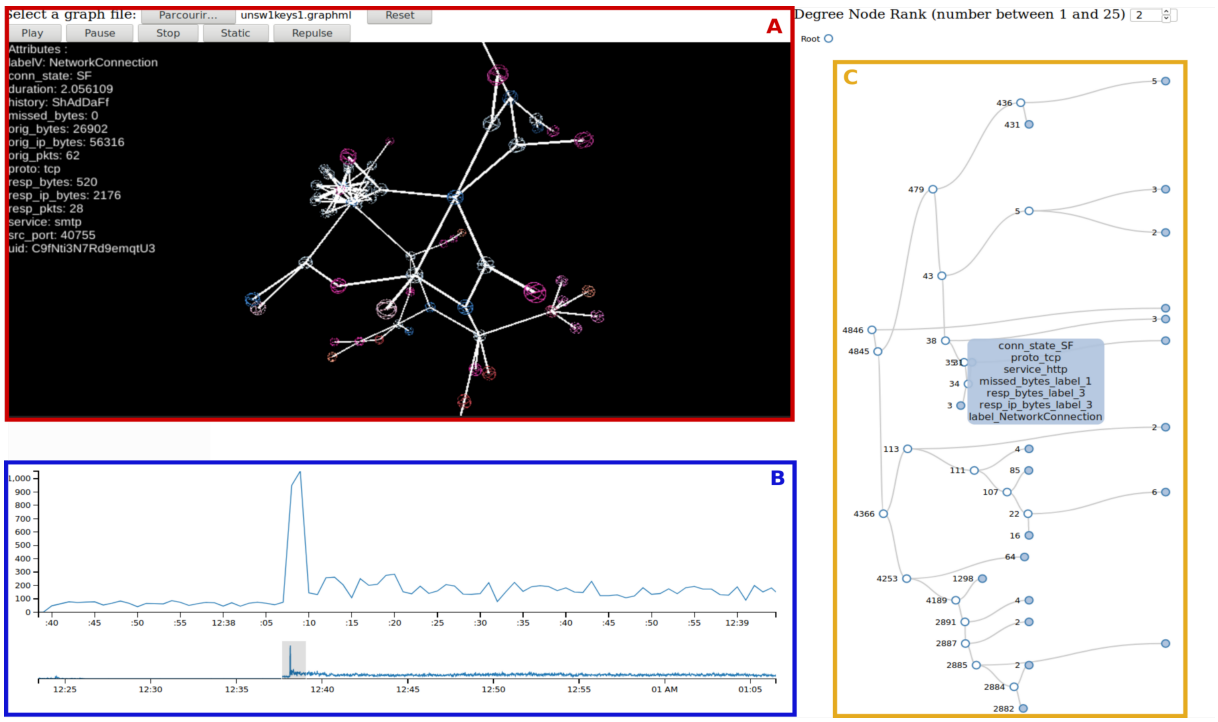

Figure 6.5 – Global Interface of our visualization tool - Control Command and Graph Visualization (A), Timeline Plot(B), Dendrogram (C)

or collapse the sub-graph starting from this node. When the analyst has chosen a criterion for aggregating nodes in a cluster, he or she can double-click on the corresponding dendrogram node.

The 3D-graph of the view **A** is then updated and all the security objects corresponding to the selection criteria are aggregated within the same node called macro-node. This is therefore a mean to cluster the 3D-graph through human analysis that would be hard to perform automatically because it requires contextual knowledge that only analysts have.

View **B** allows to give contextual information on the number of network connections at a given time t. It also allows to select the graph corresponding to this timestamp in view **A**.

Figure 6.6 shows the result of such an aggregation. The aggregation of the nodes makes the rest of the graph much more visible to the analyst. He or she can therefore continue his or her analysis by reducing the risk of occlusion. It should be mentioned that the information corresponding to the aggregated nodes is still present, just not displayed. Clicking on the macro-node displays the attributes common to the aggregated security objects.

Conclusion

We presented in this chapter an immersive data visualization approach and a prototype based on virtual reality. This prototype is voluntarily simple and offers intuitive controls for the user to adapt quickly and reduce fatigue. Navigation and graph manipulation are available to change

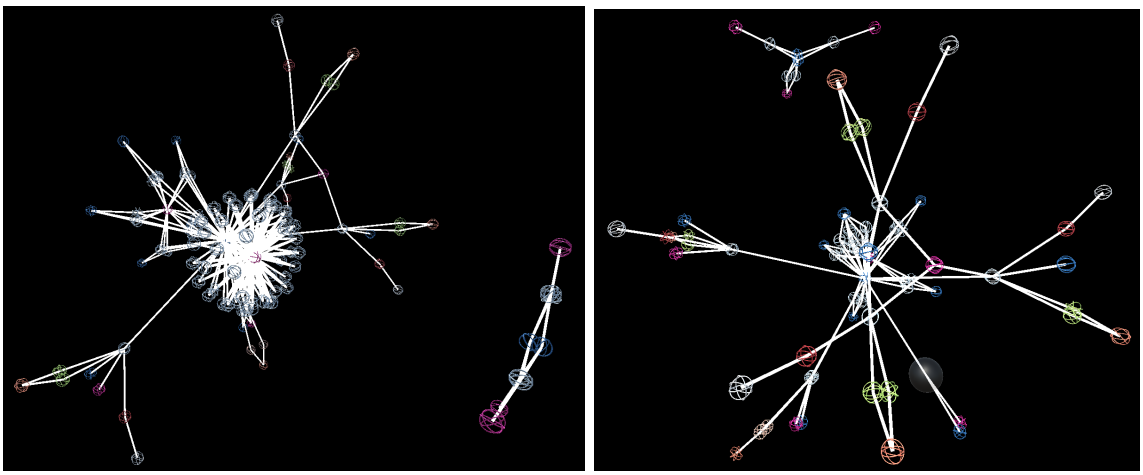

Figure 6.6 – Results of nodes aggregation performed by the analyst. (left) A large number of similar nodes are displayed in the middle obscuring the rest of the graph. (right) The nodes have been aggregated into a single meta node/

the environment scale and the point of view, as well as avoiding occlusion issues. Selection tool with strong visual feedback allows the user to analyze and link the information supported by the graph around them in order to rebuild the scenario of the attack.

However, we believe that virtual reality is useful as a collaborative tool, but not usable on a daily basis by an analyst. As a collaboration tool, functionalities related to interaction and annotation of objects by several analysts would thus bring interesting improvements.

Visualization only allows to display a limited number of objects (in the order of thousands), whereas our graphs contain millions of objects. A pre-selection of objects is thus necessary. In this chapter, we have seen two methods for this pre-selection. First, we have used event temporality to show only objects related to events that took place in a user-defined time window. Second, in order to reduce the number of objects to represent without making the information disappear, we have proposed using hierarchical clustering and a dendrogram representation to aggregate nodes with common characteristics.

To conclude, it remains to evaluate experimentally, from the point of view of analysts, the different forms of node clustering that we have proposed. Such an evaluation requires a real use of the tools we have developed by real analysts, and a systematic collection and analysis of their user experiences. Such a work is already considered in an operational environment, but could not have been performed at the time of writing this thesis.

Conclusion

To respond to the challenges listed in Section 1.1.2 and respond to the requirements of situational awareness, we explored in this thesis the use of graphs for security data analysis. To that end, we proposed a processing pipeline composed of multiple steps, from data modeling to graph visualization. In the remaining of this chapter, we first provide a summary of our contributions and then identify some areas to investigate as future work.

Contributions

To help analysts in intrusion detection and forensic analysis, we first proposed in Chapter 3 a new graph-based model of network events, the graph being made of security objects. This model, based on STIX, is intended to allow analysts to easily link relevant information and thus facilitates the analysis process. It is built from network events and information from threat intelligence according to a process that can be adapted to any type of security data. We proposed an implementation and showed that the approach, that uses data that are classically used in a SOC, is scalable and allows real-time processing, making it usable in a real SOC context. We have also shown the usefulness of graph structures through a use case example.

To enhance forensic analysis, we then proposed in Chapter 4 a process based on community detection to discover, in the security object graph described in Chapter 3, objects linked to an attack identified through a given indicator of compromise. We tested different methods and evaluated their ability to select sub-graphs of interest in a security analysis perspective. These evaluations revealed that community detection allows to identify a large number of events related to an attack. It also allows to distinguish information related to normal events from information related to an attack, when this attack results in a specific structure in the graph.

Related to intrusion detection, we proposed in Chapter 5 an unsupervised technique based on an autoencoder to efficiently detect anomalies using four different strategies. For this purpose, we provided a method to encode both the structure and the content of the security object graph described in Chapter 3. Using the CICIDS2017 and CICIDS2018 datasets, we showed that the security object graph processed by an autoencoder gives better results than common methods of anomaly detection (supervised and unsupervised), including deep learning methods.

Finally, we presented in Chapter 6 an immersive data visualization approach to bring the security experts back into the loop. The implemented prototype offers tools to explore the security object graph in an intuitive way. The visualized graph can be the one corresponding to all the available traffic, subgraphs resulting from the community detection, or subgraphs obtained at the output of the autoencoder. This visualization is thus useful for both forensic analysis and intrusion detection. In particular, our tool allows to navigate the graphs in time and space. The selection

tool also allows the user to analyze and link the information supported by the graph in order to reconstruct the attack scenarios.

All these contributions form a single processing pipeline of security data modeled as a graph and allow end-to-end analysis whether for forensic analysis or unsupervised intrusion detection. With this pipeline, we respond to three of the six functionalities of a SOC presented page 2, i.e., the collect, detect, and display functionalities. Our objective was also, beyond the functionalities themselves, to respond to the challenges identified page 3. We discuss below how our contributions help meeting the requirements relative to each of these challenges:

Diversity of network traffic We addressed the diversity of network traffic by providing a rich model of security data representing various communication protocols in Chapter 3. This model is composed of various security objects (e.g. IPAddress, X509 certificate) and edges, each having multiple attributes. It can be extended to new types of data by adding new objects and relations to the ones that already exist. To address the diversity of the data, we also propose a graph encoding technique in Chapter 5 that preserves the diversity when feeding our autoencoder. The graph encoding keeps the values of all the types of attributes that can be found in the model and transform them into categories, therefore preserving the variability of the data.

Low amount of attack-related data Having a small amount of or even no data related to attacks is not detrimental to us. Indeed, we specifically chose the autoencoder method that requires only normal data for its learning phase. Therefore, we do not depend on the availability of attack-related data.

High cost of errors Our approach based on an autoencoder reduces the cost of detection errors, as shown by the evaluation we performed using the CICIDS datasets. The 2017 dataset leads to a detection rate of 99.98% and a false positive rate of 0.42% (7.934 network connections wrongly classified as abnormal (FP) out of a total of 2.422.396 network connections), while those performed on the 2018 dataset leads to a detection rate of 100% and a false positive rate of 0.25% (4.225 network connections wrongly classified as abnormal (FP) out of a total of 1.773.761 network connections), outperforming common anomaly detection techniques. These results were obtained without aggregation or correlation processes.

Interpretation of the results by the analyst We proposed a visualization tool in Chapter 6 that allows the analyst to explore security data in an interactive way, thus reducing the problem of data interpretability. The visualization proposed represents the graph of security object in 3D with navigation features allowing the analyst to see events (represented by subgraphs) sequentially, enhancing situation awareness. We also proposed a hierarchical clustering method and an associated visualization in the form of a dendrogram to help the analyst comparing similar nodes.

Evaluation difficulties We chose two of the most recent, realistic and reasonably large synthetic datasets to evaluate our approach for all the evaluation performed during this thesis. This way, we believe our results are close to those that would be obtained on real

data. We recognize, however, that more work is needed to prove this assertion.

Perspectives

Our end-to-end approach has pointed out some research opportunities to further improve intrusion detection and ease the work of analysts. We present them hereafter.

Automatic feature extraction

The graph model is designed to be scalable and suitable to new types of data. Indeed, new objects can be added either to represent new data sources or if the network events evolve (e.g., a new protocol appears).

Taking advantage of this ability, the addition of system and application logs could enrich the model.

However, system and application logs are rarely well structured. Establishing parsing rules for each of these sources would be very costly for the security analyst. Building and updating the model would require the technical expertise of a security analyst and cost can be quite significant. This is why a research work on the automation of feature extraction or at least on an help to guide the analyst would be useful.

To this end, what is done for neural language processing is promising. For example, iACE [92], was designed to automatically extract IoCs from unstructured text. A similar approach would allow the extraction of security objects from security events and would thus help the analyst adapting the graph model according to the available data.

Taking time into account more efficiently

Security object graphs allow to link the various elements in the logs. These graphs represent in a precise and structured way the activities taking place in the information system. Although temporal information is present in the security object graphs, it is only weakly taken into account in the detection of anomalies if not at all, even though it may be important for the detection of an intrusion.

Taking temporal information into account more efficiently is an interesting opportunity to explore to improve detection.

To this end, the possible contributions of certain types of neural networks adapted to the processing of sequential data can be studied. We are thinking here particularly to LSTM (Long-Short Term Memory) [161]).

In addition, since a security object graph is dynamic, it would be interesting to reconsider the way the graph is encoded to feed the autoencoder taking into account the temporal aspect.

Outside the scope of intrusion detection, recent work related to dynamic graphs include studies on the representation of the structure of the graph, its dynamicity and the values of the various

labels associated with the edges and nodes. For example, the propositions of Zheng et al. [167] and Trivedi et al. [147] could be adapted to the intrusion detection field.

Data visualization

We presented a tool to visually represent graphs of security object. This tool allows analysts to navigate through the sub-graphs selected by our community and anomaly detection system. In the first case, the representation of alerts or indicators of compromise allows to perform a forensic analysis by following the paths in the graph. However, in the second case, even if viewing abnormal subgraphs is of great help, an additional help would be useful to the analyst so as to interpret the results of the anomaly detection.

Indeed, the first job of an analyst is to eliminate possible false positives. However, in the case of detection algorithms based on neural networks, interpretation of the results can be difficult. We used an autoencoder allowing the analyst to target attributes that deviate from normal behavior. However, as our data is very rich, a manual attribute-by-attribute analysis would be too costly. A visual help such as heatmap visualization would allow the analyst to validate the detection model and analyze the data more quickly. More generally, adding additional interactive views to our visualization prototype would enhance the security analysis.

Bibliography

[1] Ahmed Ahmim et al., « A novel hierarchical intrusion detection system based on decision tree and rules-based models », *in*: *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, IEEE, 2019, pp. 228–233.

[2] Christoforos Anagnostopoulos, « Weakly Supervised Learning: How to Engineer Labels for Machine Learning in Cyber-Security », *in*: *Data Science for Cyber-Security* (2018).

[3] Giuseppina Andresini et al., « Exploiting the Auto-Encoder Residual Error for Intrusion Detection », *in*: *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2019, pp. 281–290.

[4] M Angelini et al., « MAD: A visual analytics solution for Multi-step cyber Attacks Detection », *in*: *Journal of Computer Languages* 52 (2019), pp. 10–24.

[5] Marco Angelini, Nicolas Prigent, and Giuseppe Santucci, « PERCIVAL: proactive and reactive attack and response assessment for cyber incidents using visual analytics », *in*: *2015 IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2015, pp. 1–8.

[6] ANSSI, *OpenCTI*, 2020, URL: `https://www.opencti.io/fr/`.

[7] Giovanni Apruzzese et al., « On the effectiveness of machine and deep learning for cyber security », *in*: *2018 10th International Conference on Cyber Conflict (CyCon)*, IEEE, 2018, pp. 371–390.

[8] Dustin L Arendt et al., « Ocelot: user-centered design of a decision support visualization for network quarantine », *in*: *Visualization for Cyber Security (VizSec), 2015 IEEE Symposium on*, IEEE, 2015, pp. 1–8.

[9] Hagai Attias, « A variational baysian framework for graphical models », *in*: *Advances in neural information processing systems*, 2000, pp. 209–215.

[10] Stefan Axelsson, « The base-rate fallacy and its implications for the difficulty of intrusion detection », *in*: *Proceedings of the 6th ACM Conference on Computer and Communications Security*, 1999, pp. 1–7.

[11] R Can Aygun and A Gokhan Yavuz, « Network anomaly detection with stochastically improved autoencoder based models », *in*: *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, IEEE, 2017, pp. 193–198.

[12] M Bahrololum and M Khaleghi, « Anomaly intrusion detection system using hierarchical gaussian mixture model », *in*: *International journal of computer science and network security* 8.*8* (2008), pp. 264–271.

[13] Jonathan Baker, Matthew Hansbury, and Daniel Haynes, « The OVAL Language Specification », *in*: *MITRE, Bedford, Massachusetts (url: http://ebookbrowsee. net/ovallanguage-specification-08-08-2011-pdf-d222972411)* (2011).

[14] Paul Barford et al., « Cyber SA: Situational awareness for cyber defense », *in*: *Cyber situational awareness*, Springer, 2010, pp. 3–13.

[15] Sean Barnum and Amit Sethi, « Attack patterns as a knowledge resource for building secure software », *in*: *OMG Software Assurance Workshop: Cigital*, 2007.

[16] Sean Barnum et al., « The CybOX language specification », *in*: *draft, The MITRE Corporation* (2012).

[17] Sean Barnum et al., « The cybox language specification », *in*: *The MITRE Corporation* (2012).

[18] Matthew P Barrett, *Framework for improving critical infrastructure cybersecurity version 1.1*, tech. rep., 2018.

[19] M Bastian, S Heymann, and M Jacomy, « Gephi: an open source software for exploring and manipulating networks. International AAAI Conference on Weblogs and Social Media [Internet]. 2009 », *in*: *There is no corresponding record for this reference.[Google Scholar]* (2015).

[20] Igor LO Bastos et al., « Mora: A generative approach to extract spatiotemporal information applied to gesture recognition », *in*: *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, IEEE, 2018, pp. 1–6.

[21] Desiree Beck et al., *CybOX™ Version 2.1.1. Part 36: Network Connection Object*, OASIS Committee Specification Draft 0, 2016, URL: `http://docs.oasis-open.org/cti/cybox/v2.1.1/cybox-v2.1.1-part36-network-connection.html`.

[22] Desiree Beck et al., *CybOX™ Version 2.1.1. Part 50: Socket Address Object*, OASIS Committee Specification Draft 0, 2016, URL: `http://docs.oasis-open.org/cti/cybox/v2.1.1/csprd01/part50-socket-address/cybox-v2.1.1-csprd01-part50-socket-address.html`.

[23] Jeremiah Blatz, « CSRF: Attack and Defense », *in*: *McAfee® Foundstone® Professional Services, White Paper* (2007).

[24] Vincent D Blondel et al., « Fast unfolding of communities in large networks », *in*: *Journal of statistical mechanics: theory and experiment* 2008.*10* (2008), P10008.

[25] Fabian Böhm, Florian Menges, and Günther Pernul, « Graph-based visual analytics for cyber threat intelligence », *in*: *Cybersecurity* 1.*1* (2018), p. 16.

[26] Yacine Bouzida et al., « Efficient intrusion detection using principal component analysis », *in*: *3éme Conférence sur la Sécurité et Architectures Réseaux (SAR), La Londe, France*, 2004, pp. 381–395.

[27] Shaosheng Cao, Wei Lu, and Qiongkai Xu, « Deep neural networks for learning graph representations », *in*: *Thirtieth AAAI conference on artificial intelligence*, 2016.

[28] Shaosheng Cao, Wei Lu, and Qiongkai Xu, « Grarep: Learning graph representations with global structural information », *in*: *Proceedings of the 24th ACM international on conference on information and knowledge management*, 2015, pp. 891–900.

[29] Eoghan Casey, Greg Back, and Sean Barnum, « Leveraging CybOX™ to standardize representation and exchange of digital forensic information », *in*: *Digital Investigation* 12 (2015), S102–S110.

[30] Eoghan Casey et al., « Advancing coordinated cyber-investigations and tool interoperability using a community developed specification language », *in*: *Digital investigation* 22 (2017), pp. 14–45.

[31] Apache Cassandra, « Apache cassandra », *in*: *Website. Available online at http://planetcassand org/what-is-apache-cassandra* (2014), p. 13.

[32] Brant A Cheikes et al., *Common platform enumeration: Naming specification version 2.3*, US Department of Commerce, National Institute of Standards and Technology, 2011.

[33] Daiki Chiba et al., « Detecting malicious websites by learning IP address features », *in*: *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet*, IEEE, 2012, pp. 29–39.

[34] Aaron Clauset, Mark EJ Newman, and Cristopher Moore, « Finding community structure in very large networks », *in*: *Physical review E* 70.*6* (2004), p. 066111.

[35] Damien Crémilleux et al., « VEGAS: Visualizing, exploring and grouping alerts », *in*: *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2016, pp. 1097–1100.

[36] Roman Danyliw, « RFC 7970: The Incident Object Description Exchange Format Version 2 », *in*: *Internet Engineering Task Force (IETF)* (2016).

[37] Herve Debar, David Curry, and Benjamin Feinstein, « The intrusion detection message exchange format (IDMEF) », *in*: *IETF Request for Comments* 4765 (2007).

[38] Hervé Debar, Marc Dacier, and Andreas Wespi, « Towards a taxonomy of intrusion-detection systems », *in*: *Computer networks* 31.*8* (1999), pp. 805–822.

[39] 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.*1* (1977), pp. 1–22.

[40] Qi Ding et al., « Intrusion as (anti) social communication: characterization and detection », *in*: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 886–894.

[41] 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*, 2017, pp. 1285–1298.

[42] James P Early and Carla E Brodley, « Behavioral features for network anomaly detection », *in*: *Machine Learning and Data Mining for Computer Security*, Springer, 2006, pp. 107–124.

[43] Andreas Ekelhart, Elmar Kiesling, and Kabul Kurniawan, « Taming the logs-Vocabularies for semantic security analysis », *in*: *Procedia Computer Science* 137 (2018), pp. 109–119.

[44] *Elasticsearch*, version 7.9, URL: `https://www.elastic.co/fr/elasticsearch/`.

[45] Mica R Endsley, « Toward a theory of situation awareness in dynamic systems », *in*: *Human factors* 37.*1* (1995), pp. 32–64.

[46] Juan M Estevez-Tapiador, Pedro Garcia-Teodoro, and Jesus E Diaz-Verdejo, « Stochastic protocol modeling for anomaly based network intrusion detection », *in*: *First IEEE International Workshop on Information Assurance, 2003. IWIAS 2003. Proceedings.* IEEE, 2003, pp. 3–12.

[47] Gilberto Fernandes et al., « A comprehensive survey on network anomaly detection », *in*: *Telecommunication Systems* 70.*3* (2019), pp. 447–489.

[48] Mohamed Amine Ferrag et al., « Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study », *in*: *Journal of Information Security and Applications* 50 (2020), p. 102419.

[49] Santo Fortunato, « Community detection in graphs », *in*: *Physics reports* 486.*3-5* (2010), pp. 75–174.

[50] Santo Fortunato and Darko Hric, « Community detection in networks: A user guide », *in*: *Physics reports* 659 (2016), pp. 1–44.

[51] Jérôme François, Shaonan Wang, Thomas Engel, et al., « BotTrack: tracking botnets using NetFlow and PageRank », *in*: *International Conference on Research in Networking*, Springer, 2011, pp. 1–14.

[52] David Freedman, Robert Pisani, and Roger Purves, « Statistics (international student edition) », *in*: *Pisani, R. Purves, 4th edn. WW Norton & Company, New York* (2007).

[53] Thomas MJ Fruchterman and Edward M Reingold, « Graph drawing by force-directed placement », *in*: *Software: Practice and experience* 21.*11* (1991), pp. 1129–1164.

[54] Amirhossein Gharib et al., « An evaluation framework for intrusion detection dataset », *in*: *Information Science and Security (ICISS), 2016 International Conference on*, IEEE, 2016, pp. 1–6.

[55] W Gibb and D Kerr, « OpenIOC: back to the basics », *in*: *OpenIOC: Back to the Basics* (2018).

[56] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT press, 2016.

[57] Aditya Grover and Jure Leskovec, « node2vec: Scalable feature learning for networks », *in*: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[58] Thomas R Gruber, « Toward principles for the design of ontologies used for knowledge sharing? », *in*: *International journal of human-computer studies* 43.*5-6* (1995), pp. 907–928.

[59] Mohamed Hamdi and Noureddine Boudriga, « Detecting Denial-of-Service attacks using the wavelet transform », *in*: *Computer Communications* 30.*16* (2007), pp. 3203–3213.

[60] Will Hamilton, Zhitao Ying, and Jure Leskovec, « Inductive representation learning on large graphs », *in*: *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[61] William L Hamilton, Rex Ying, and Jure Leskovec, « Representation learning on graphs: Methods and applications », *in*: *arXiv preprint arXiv:1709.05584* (2017).

[62] Nick Heard, Patrick Rubin-Delanchy, and Daniel J Lawson, « Filtering automated polling traffic in computer network flow data », *in*: *2014 IEEE Joint Intelligence and Security Informatics Conference*, IEEE, 2014, pp. 268–271.

[63] Ivan Herman, Guy Melançon, and M Scott Marshall, « Graph visualization and navigation in information visualization: A survey », *in*: *IEEE Transactions on visualization and computer graphics* 6.*1* (2000), pp. 24–43.

[64] Elena Hernández-Pereira et al., « Conversion methods for symbolic features: A comparison applied to an intrusion detection problem », *in*: *Expert Systems with Applications* 36.*7* (2009), pp. 10612–10617.

[65] Md Nahid Hossain et al., « SLEUTH: Real-time attack scenario reconstruction from COTS audit data », *in*: *Proc. USENIX Secur.* 2017, pp. 487–504.

[66] Christopher Humphries et al., « Corgi: Combination, organization and reconstruction through graphical interactions », *in*: *Proceedings of the Eleventh Workshop on Visualization for Cyber Security*, 2014, pp. 57–64.

[67] Stephen C Johnson, « Hierarchical clustering schemes », *in*: *Psychometrika* 32.*3* (1967), pp. 241–254.

[68] William Kent, « A simple guide to five normal forms in relational database theory », *in*: *Communications of the ACM* 26.*2* (1983), pp. 120–125.

[69] Elmar Kiesling et al., « The SEPSES Knowledge Graph: An Integrated Resource for Cybersecurity », *in*: *International Semantic Web Conference*, Springer, 2019, pp. 198–214.

[70] Jihyun Kim et al., « Long short term memory recurrent neural network classifier for intrusion detection », *in*: *2016 International Conference on Platform Technology and Service (PlatCon)*, IEEE, 2016, pp. 1–5.

[71] Samuel T King and Peter M Chen, « Backtracking intrusions », *in*: *ACM SIGOPS Operating Systems Review*, vol. 37, ACM, 2003, pp. 223–236.

[72] Thomas N Kipf and Max Welling, « Semi-supervised classification with graph convolutional networks », *in*: *arXiv preprint arXiv:1609.02907* (2016).

[73] Ivan Kirillov et al., *Malware attribute enumeration and characterization*, 2011.

[74] Myungsook Klassen and Ning Yang, « Anomaly based intrusion detection in wireless networks using Bayesian classifier », *in*: *2012 IEEE fifth international conference on advanced computational intelligence (ICACI)*, IEEE, 2012, pp. 257–264.

[75] Marius Kloft et al., « Automatic feature selection for anomaly detection », *in*: *Proceedings of the 1st ACM workshop on Workshop on AISec*, 2008, pp. 71–76.

[76] Satoru Kobayashi et al., « Mining causality of network events in log data », *in*: *IEEE Transactions on Network and Service Management* 15.*1* (2017), pp. 53–67.

[77] Stephen G Kobourov, « Spring embedders and force directed graph drawing algorithms », *in*: *arXiv preprint arXiv:1201.3011* (2012).

[78] Daphne Koller et al., *Introduction to statistical relational learning*, MIT press, 2007.

[79] Christopher Kruegel and Giovanni Vigna, « Anomaly detection of web-based attacks », *in*: *Proceedings of the 10th ACM conference on Computer and communications security*, 2003, pp. 251–261.

[80] Anukool Lakhina, Mark Crovella, and Christophe Diot, « Mining anomalies using traffic feature distributions », *in*: *ACM SIGCOMM computer communication review* 35.*4* (2005), pp. 217–228.

[81] Anukool Lakhina et al., « Structural analysis of network traffic flows », *in*: *Proceedings of the joint international conference on Measurement and modeling of computer systems*, 2004, pp. 61–72.

[82] Pavel Laskov et al., « Learning intrusion detection: supervised or unsupervised? », *in*: *International Conference on Image Analysis and Processing*, Springer, 2005, pp. 50–57.

[83] Yann Le Cun and Françoise Fogelman-Soulié, « Modèles connexionnistes de l'apprentissage », *in*: *Intellectica* 2.*1* (1987), pp. 114–143.

[84] JooHwa Lee and KeeHyun Park, « AE-CGAN Model based High Performance Network Intrusion Detection System », *in*: *Applied Sciences* 9.*20* (2019), p. 4221.

[85] Laetitia Leichtnam et al., « Forensic Analysis of Network Attacks: Restructuring Security Events as Graphs and Identifying Strongly Connected Sub-graphs », *in*: *Workshop on Traffic Measurements for Cybersecurity (WTMC2020)*, 2020.

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

[87] 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.

[88] Laetitia Leichtnam et al., « Starlord: Linked security data exploration in a 3d graph », *in*: *2017 IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2017, pp. 1–4.

[89] Yang Li and Li Guo, « An active learning based TCM-KNN algorithm for supervised network intrusion detection », *in*: *Computers & security* 26.*7-8* (2007), pp. 459–467.

[90] Yinhui Li et al., « An efficient intrusion detection system based on support vector machines and gradually feature removal method », *in*: *Expert Systems with Applications* 39.*1* (2012), pp. 424–430.

[91] Zhou Li et al., « Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures », *in*: *2013 IEEE Symposium on Security and Privacy*, IEEE, 2013, pp. 112–126.

[92] Xiaojing Liao et al., « Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence », *in*: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 755–766.

[93] Yihua Liao and V Rao Vemuri, « Use of k-nearest neighbor classifier for intrusion detection », *in*: *Computers & security* 21.*5* (2002), pp. 439–448.

[94] Tao Liu et al., « Method for network anomaly detection based on Bayesian statistical model with time slicing », *in*: *2008 7th World Congress on Intelligent Control and Automation*, IEEE, 2008, pp. 3359–3362.

[95] Yarden Livnat et al., « Visual correlation for situational awareness », *in*: *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.* IEEE, 2005, pp. 95–102.

[96] Shan Lu and Mieczyslaw M Kokar, « A situation assessment framework for cyber security information relevance reasoning », *in*: *2015 18th International Conference on Information Fusion (Fusion)*, IEEE, 2015, pp. 1459–1466.

[97] Matthew V Mahoney and Philip K Chan, *PHAD: Packet header anomaly detection for identifying hostile network traffic*, tech. rep., 2001.

[98] David Mann, « An introduction to the common configuration enumeration (cce) », *in*: *Technical Report, Department, Tech. Rep.* (2008).

[99] David E Mann and Steven M Christey, « Towards a common enumeration of vulnerabilities », *in*: *2nd Workshop on Research with Security Vulnerability Databases, Purdue University, West Lafayette, Indiana*, 1999.

[100] Robert A Martin, « Making security measurable and manageable », *in*: *MILCOM 2008-2008 IEEE Military Communications Conference*, IEEE, 2008, pp. 1–9.

[101] Robert A Martin and Sean Barnum, « Common weakness enumeration (cwe) status update », *in*: *ACM SIGAda Ada Letters* 28.*1* (2008), pp. 88–91.

[102] Paul Maxwell, Elie Alhajjar, and Nathaniel D Bastian, « Intelligent Feature Engineering for Cybersecurity », *in*: *2019 IEEE International Conference on Big Data (Big Data)*, IEEE, 2019, pp. 5005–5011.

[103] Silvia Metelli, Nicholas Heard, et al., « On Bayesian new edge prediction and anomaly detection in computer networks », *in*: *The Annals of Applied Statistics* 13.*4* (2019), pp. 2586–2610.

[104] Sadegh M Milajerdi et al., « HOLMES: real-time APT detection through correlation of suspicious information flows », *in*: *arXiv preprint arXiv:1810.01594* (2018).

[105] Erxue Min et al., « SU-IDS: A semi-supervised and unsupervised framework for network intrusion detection », *in*: *International Conference on Cloud Computing and Security*, Springer, 2018, pp. 322–334.

[106] Josiane Mothe, Karen Mkhitaryan, and Mariam Haroutunian, « Community detection: Comparison of state of the art algorithms », *in*: *2017 Computer Science and Information Technologies (CSIT)*, IEEE, 2017, pp. 125–129.

[107] Mark EJ Newman, « Fast algorithm for detecting community structure in networks », *in*: *Physical review E* 69.*6* (2004), p. 066133.

[108] Steven Noel et al., « CyGraph: graph-based analytics and visualization for cybersecurity », *in*: *Handbook of Statistics*, vol. 35, Elsevier, 2016, pp. 117–167.

[109] OASIS, *STIXv2.0*, URL: `freetaxii.github.io`.

[110] Iosif-Viorel Onut and Ali A Ghorbani, « A Feature Classification Scheme For Network Intrusion Detection. », *in*: *IJ Network Security* 5.*1* (2007), pp. 1–15.

[111] Cyril Onwubiko, « CoCoa: An Ontology for Cybersecurity Operations Centre Analysis Process », *in*: *2018 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA)*, IEEE, 2018, pp. 1–8.

[112] *OTX Alienvault*, URL: `https://otx.alienvault.com/`.

[113] Mingdong Ou et al., « Asymmetric transitivity preserving graph embedding », *in*: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1105–1114.

[114] Vern Paxson, « Bro: a system for detecting network intruders in real-time », *in*: *Computer networks* 31.*23-24* (1999), pp. 2435–2463.

[115] Kexin Pei et al., « Hercule: Attack story reconstruction via community discovery on correlated log graph », *in*: *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ACM, 2016, pp. 583–595.

[116] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena, « Deepwalk: Online learning of social representations », *in*: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[117] Marco AF Pimentel et al., « A review of novelty detection », *in*: *Signal Processing* 99 (2014), pp. 215–249.

[118] R Pincus, « Barnett, V., and Lewis T.: Outliers in Statistical Data. J. Wiley & Sons 1994, XVII. 582 pp.,£ 49.95 », *in*: *Biometrical Journal* 37.*2* (1995), pp. 256–256.

[119] Pascal Pons and Matthieu Latapy, « Computing communities in large networks using random walks », *in*: *International symposium on computer and information sciences*, Springer, 2005, pp. 284–293.

[120] *Prelude*, URL: `https://www.prelude-siem.com/`.

[121] Ricardo S Puttini, Zakia Marrakchi, and Ludovic Mé, « A Bayesian Classification Model for Real-Time Intrusion Detection », *in*: *22th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering (MAXENT'2002)*, 2002.

[122] Majjed Al-Qatf et al., « Deep learning approach combining sparse autoencoder with SVM for network intrusion detection », *in*: *IEEE Access* 6 (2018), pp. 52843–52856.

[123] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara, « Near linear time algorithm to detect community structures in large-scale networks », *in*: *Physical review E* 76.*3* (2007), p. 036106.

[124] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo, « struc2vec: Learning node representations from structural identity », *in*: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 385–394.

[125] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin, « " Why should I trust you?" Explaining the predictions of any classifier », *in*: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[126] Piazza Rich, Wunder John, and Jordan Bret, « STIX Version 2.0. Part 1: STIX Core Concepts », *in*: *OASIS Committee Specification* (2017), URL: `http://docs.oasisopen.org/cti/stix/v2.0/cs01/part1-stix-core/stix-v2.0-cs01-part1-stix-core.html`.

[127] Marko A Rodriguez, « The gremlin graph traversal machine and language (invited talk) », *in*: *Proceedings of the 15th Symposium on Database Programming Languages*, 2015, pp. 1–10.

[128] Ryan Anthony Rossi, Rong Zhou, and Nesreen Ahmed, « Deep inductive graph representation learning », *in*: *IEEE Transactions on Knowledge and Data Engineering* (2018).

[129] Martin Rosvall and Carl T Bergstrom, « Maps of random walks on complex networks reveal community structure », *in*: *Proceedings of the National Academy of Sciences* 105.*4* (2008), pp. 1118–1123.

[130] Karen Scarfone and Peter Mell, *Guide to intrusion detection and prevention systems (IDPS)*, tech. rep., National Institute of Standards and Technology, 2012.

[131] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani, « Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. », *in*: *ICISSP*, 2018, pp. 108–116.

[132] Sharp, Austin et al., *JanusGraph*, version 0.5.2, URL: `https://janusgraph.org/`.

[133] Nino Shervashidze et al., « Weisfeiler-lehman graph kernels. », *in*: *Journal of Machine Learning Research* 12.*9* (2011).

[134] Hossein Siadati, Bahador Saket, and Nasir Memon, « Detecting malicious logins in enterprise networks using visualization », *in*: *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2016, pp. 1–8.

[135] Dimitrios Sisiaridis and Olivier Markowitch, « Feature Extraction and Feature Selection: Reducing Data Complexity With Apache Spark », *in*: *International Journal of Network Security and its Security Applications (IJNSA)*, vol. 9, 6, 2017.

[136] Robin Sommer and Vern Paxson, « Outside the closed world: On using machine learning for network intrusion detection », *in*: *2010 IEEE symposium on security and privacy*, IEEE, 2010, pp. 305–316.

[137] Stuart Staniford, James A Hoagland, and Joseph M McAlerney, « Practical automated detection of stealthy portscans », *in*: *Journal of Computer Security* 10.*1-2* (2002), pp. 105–136.

[138] Gianluca Stringhini et al., « Marmite: spreading malicious file reputation through download graphs », *in*: *Proceedings of the 33rd Annual Computer Security Applications Conference*, 2017, pp. 91–102.

[139] Blake E Strom et al., « Mitre att&ck: Design and philosophy », *in*: *MITRE Product MP* (2018), pp. 18–0944.

[140] Hudan Studiawan, Christian Payne, and Ferdous Sohel, « Graph clustering and anomaly detection of access control log for forensic purposes », *in*: *Digital Investigation* 21 (2017), pp. 76–87.

[141] Nan Sun et al., « Data-driven cybersecurity incident prediction: A survey », *in*: *IEEE Communications Surveys & Tutorials* 21.*2* (2018), pp. 1744–1772.

[142] Zareen Syed et al., « UCO: A unified cybersecurity ontology », *in*: *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[143] Direction Interministérielle des Systèmes d'Information et de Communication (DISIC), *Référentiel Général d'Interopérabilité v2.0*, 2016.

[144] Roberto Tamassia, Giuseppe Di Battista, and Carlo Batini, « Automatic graph drawing and readability of diagrams », *in*: *IEEE Transactions on Systems, Man, and Cybernetics* 18.*1* (1988), pp. 61–79.

[145] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau, « Guilt by association: large scale malware detection by mining file-relation graphs », *in*: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 1524–1533.

[146] Vincent A Traag and Jeroen Bruggeman, « Community detection in networks with positive and negative links », *in*: *Physical Review E* 80.*3* (2009), p. 036115.

[147] Rakshit Trivedi et al., « Dyrep: Learning representations over dynamic graphs », *in*: *International Conference on Learning Representations*, 2019.

[148] Orestis Tsigkas, Olivier Thonnard, and Dimitrios Tzovaras, « Visual spam campaigns analysis using abstract graphs representation », *in*: *Proceedings of the ninth international symposium on visualization for cyber security*, 2012, pp. 64–71.

[149] *Unity 3D*, version 2018.2.0, URL: `https://unity.com/`.

[150] Jake VanderPlas, *Python data science handbook: Essential tools for working with data*, " O'Reilly Media, Inc.", 2016.

[151] Kalyan Veeramachaneni et al., « AIˆ 2: 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.

[152] Tatiana Von Landesberger et al., « Visual analysis of large graphs: state-of-the-art and future research challenges », *in*: *Computer graphics forum*, vol. 30, 6, Wiley Online Library, 2011, pp. 1719–1749.

[153] Cynthia Wagner et al., « Misp: The design and implementation of a collaborative threat intelligence sharing platform », *in*: *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*, 2016, pp. 49–56.

[154] Daixin Wang, Peng Cui, and Wenwu Zhu, « Structural deep network embedding », *in*: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1225–1234.

[155] Ke Wang and Salvatore J Stolfo, « Anomalous payload-based network intrusion detection », *in*: *International workshop on recent advances in intrusion detection*, Springer, 2004, pp. 203–222.

[156] Wei Wang et al., « Malware traffic classification using convolutional neural network for representation learning », *in*: *2017 International Conference on Information Networking (ICOIN)*, IEEE, 2017, pp. 712–717.

[157] Michelle Q Wang Baldonado, Allison Woodruff, and Allan Kuchinsky, « Guidelines for using multiple views in information visualization », *in*: *Proceedings of the working conference on Advanced visual interfaces*, 2000, pp. 110–119.

[158] Charles V Wright, Fabian Monrose, and Gerald M Masson, « Using visual motifs to classify encrypted traffic », *in*: *Proceedings of the 3rd international workshop on Visualization for computer security*, 2006, pp. 41–50.

[159] 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)*, IEEE, 2019, pp. 340–355.

[160] Keyulu Xu et al., « Representation learning on graphs with jumping knowledge networks », *in*: *arXiv preprint arXiv:1806.03536* (2018).

[161] Yu Yan et al., « A Network Intrusion Detection Method Based on Stacked Autoencoder and LSTM », *in*: *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, IEEE, 2020, pp. 1–6.

[162] Chuanlong Yin et al., « A deep learning approach for intrusion detection using recurrent neural networks », *in*: *Ieee Access* 5 (2017), pp. 21954–21961.

[163] Xiaoxin Yin et al., « VisFlowConnect: netflow visualizations of link relationships for security situational awareness », *in*: *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, 2004, pp. 26–34.

[164] Daokun Zhang et al., « Network representation learning: A survey », *in*: *IEEE transactions on Big Data* (2018).

[165] Jiong Zhang and Mohammad Zulkernine, « Anomaly based network intrusion detection with unsupervised outlier detection », *in*: *2006 IEEE International Conference on Communications*, vol. 5, IEEE, 2006, pp. 2388–2393.

[166] Yong Zhang et al., « Network intrusion detection: Based on deep hierarchical network and original flow data », *in*: *IEEE Access* 7 (2019), pp. 37004–37016.

[167] Li Zheng et al., « AddGraph: Anomaly Detection in Dynamic Graph Using Attention-based Temporal GCN. », *in*: *IJCAI*, 2019, pp. 4419–4425.

Appendices

CybOX HTTP_Network_Connection Instance example

This example is extract from the CybOX Github project [3] and describes how an HTTP Network Connection Instance is represented in the CybOX language.

```
<?xml version="1.0" encoding="UTF-8"?>
<cybox:Observables xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cybox="http://docs.oasis-open.org/cti/ns/cybox/core-2"
    xmlns:cyboxCommon="http://docs.oasis-open.org/cti/ns/cybox/common-2"
    xmlns:AddressObj="http://docs.oasis-open.org/cti/ns/cybox/objects/address-2"
    xmlns:PortObj="http://docs.oasis-open.org/cti/ns/cybox/objects/port-2"
    xmlns:SocketAddressObj=
    "http://docs.oasis-open.org/cti/ns/cybox/objects/socket-address-1"
    xmlns:NetworkConnectionObj=
    "http://docs.oasis-open.org/cti/ns/cybox/objects/network-connection-2"
    xmlns:HTTPSessionObj="http://docs.oasis-open.org/cti/ns/cybox/objects/http-session-2"
    xmlns:example="http://example.com/"
    xsi:schemaLocation="
␣␣␣␣http://docs.oasis-open.org/cti/ns/cybox/core-2␣../core.xsd
␣␣␣␣http://docs.oasis-open.org/cti/ns/cybox/objects/network-connection-2␣../objects
␣␣␣␣/Network_Connection_Object.xsd"
    cybox_major_version="2" cybox_minor_version="1" cybox_update_version="1">
    <cybox:Observable id="example:Observable-1b427720-98d7-4735-b125-754c7e08f285">
        <cybox:Description>
            This Observable specifies an example instance of a Network Connection Object
            with an HTTP Session.
        </cybox:Description>
        <cybox:Object id="example:Object-d1fdd983-530b-489f-9ab8-ed3cb5212c35">
            <cybox:Properties xsi:type="NetworkConnectionObj:NetworkConnectionObjectType">
                <NetworkConnectionObj:Layer3_Protocol datatype="string">
                    IPv4</NetworkConnectionObj:Layer3_Protocol>
                <NetworkConnectionObj:Layer4_Protocol datatype="string">
                    TCP</NetworkConnectionObj:Layer4_Protocol>
                <NetworkConnectionObj:Layer7_Protocol datatype="string">
                    HTTP</NetworkConnectionObj:Layer7_Protocol>
                <NetworkConnectionObj:Source_Socket_Address>
                    <SocketAddressObj:IP_Address>
```

3. https://github.com/CybOXProject/schemas/blob/master/samples/CybOX_Network_Connection_HTTP_Instance.xml

```
                <AddressObj:Address_Value>192.168.1.15</AddressObj:Address_Value>
            </SocketAddressObj:IP_Address>
            <SocketAddressObj:Port>
                <PortObj:Port_Value>5525</PortObj:Port_Value>
            </SocketAddressObj:Port>
        </NetworkConnectionObj:Source_Socket_Address>
        <NetworkConnectionObj:Destination_Socket_Address>
            <SocketAddressObj:IP_Address>
                <AddressObj:Address_Value>198.49.123.10</AddressObj:Address_Value>
            </SocketAddressObj:IP_Address>
            <SocketAddressObj:Port>
                <PortObj:Port_Value>80</PortObj:Port_Value>
            </SocketAddressObj:Port>
        </NetworkConnectionObj:Destination_Socket_Address>
        <NetworkConnectionObj:Layer7_Connections>
            <NetworkConnectionObj:HTTP_Session>
                <HTTPSessionObj:HTTP_Request_Response>
                    <HTTPSessionObj:HTTP_Client_Request>
                        <HTTPSessionObj:HTTP_Request_Line>
                            <HTTPSessionObj:HTTP_Method datatype="string">
                                GET</HTTPSessionObj:HTTP_Method>
                            <HTTPSessionObj:Version>
                                HTTP/1.1</HTTPSessionObj:Version>
                        </HTTPSessionObj:HTTP_Request_Line>
                        <HTTPSessionObj:HTTP_Request_Header>
                            <HTTPSessionObj:Parsed_Header>
                                <HTTPSessionObj:Accept_Encoding>
                                    gzip</HTTPSessionObj:Accept_Encoding>
                                <HTTPSessionObj:Connection>
                                    close</HTTPSessionObj:Connection>
                            </HTTPSessionObj:Parsed_Header>
                        </HTTPSessionObj:HTTP_Request_Header>
                    </HTTPSessionObj:HTTP_Client_Request>
                    <HTTPSessionObj:HTTP_Server_Response>
                        <HTTPSessionObj:HTTP_Status_Line>
                            <HTTPSessionObj:Version>
                                HTTP/1.1</HTTPSessionObj:Version>
                            <HTTPSessionObj:Status_Code>
                                200</HTTPSessionObj:Status_Code>
                            <HTTPSessionObj:Reason_Phrase>
                                OK</HTTPSessionObj:Reason_Phrase>
                        </HTTPSessionObj:HTTP_Status_Line>
                        <HTTPSessionObj:HTTP_Response_Header>
                            <HTTPSessionObj:Parsed_Header>
                                <HTTPSessionObj:Server>
                                    Apache</HTTPSessionObj:Server>
                                <HTTPSessionObj:Transfer_Encoding>
                                    chunked</HTTPSessionObj:Transfer_Encoding>
                            </HTTPSessionObj:Parsed_Header>
                        </HTTPSessionObj:HTTP_Response_Header>
                    </HTTPSessionObj:HTTP_Server_Response>
                </HTTPSessionObj:HTTP_Request_Response>
```

```
                    </NetworkConnectionObj:HTTP_Session>
                </NetworkConnectionObj:Layer7_Connections>
            </cybox:Properties>
        </cybox:Object>
    </cybox:Observable>
</cybox:Observables>
```

DOCTORAT BRETAGNE LOIRE / MATHSTIC

CentraleSupélec

Titre : Détection et visualisation d'anomalies dans des événements réseaux hétérogènes : modélisation des événements sous forme de graphes et détection de communautés et de nouveautés grâce à l'apprentissage automatique

Mot clés : sécurité informatique, détection d'intrusion, analyse forensique, analyse de graphe, visualisation

Résumé : L'objectif général de cette thèse est d'évaluer l'intérêt des graphes dans le domaine de l'analyse des données de sécurité.

Nous proposons une approche de bout en bout composé d'un modèle unifié de données réseau sous forme de graphes, d'un système de découverte de communauté, d'un système de détection d'anomalies non supervisé et d'une visualisation des données sous forme de graphes.

Le modèle unifié est obtenue en utilisant des graphes de connaissance pour représenter des journaux d'évènements hétérogènes ainsi que du trafic réseau. La détection de communautés permet de sélectionner des sous-graphes représentant des événements fortement liés à une alerte ou à un IoC et qui sont donc pertinents pour l'analyse forensique. Notre système de détection d'intrusion basé sur les anomalies repose sur la détection de nouveauté par un autoencodeur et donne de très bons résultats sur les jeux de données CICIDS 2017 et 2018. Enfin, la visualisation immersive des données de sécurité permet de mettre en évidence les relations entre les éléments de sécurité et les événements malveillants ou les IoCs. Cela donne à l'analyste de sécurité un bon point de départ pour explorer les données et reconstruire des scénarii d'attaques globales.

Title: Detecting and visualizing anomalies in heterogeneous network events: modeling events as graph structures and detecting communities and novelties with machine learning

Keywords: computer security, intrusion detection, forensic analysis, graph analysis, visualization

Abstract: The general objective of this thesis is to evaluate the interest of graph structures in the field of security data analysis.

We propose an end-to-end approach consisting in a unified view of the network data in the form of graphs, a community discovery system, an unsupervised anomaly detection system, and a visualization of the data in the form of graphs.

The unified view is obtained using knowledge graphs to represent heterogeneous log files and network traffics. Community detection allows us to select sub-graphs representing events that are strongly related to an alert or an IoC and that are thus relevant for forensic analysis. Our anomaly-based intrusion detection system relies on novelty detection by an autoencoder and exhibits very good results on CICIDS 2017 and 2018 datasets. Finally, an immersive visualization of security data allows highlighting the relations between security elements and malicious events or IOCs. This gives the security analyst a good starting point to explore the data and reconstruct global attack scenarii.