Matching Knowledge Graphs for Cybersecurity Countermeasures Selection

Kéren A Saint-Hilaire, Christopher Neal, Frédéric Cuppens, Nora Cuppens-Boulahia, Makhlouf Hadji

Matching Knowledge Graphs for Cybersecurity Countermeasures Selection

Kéren A Saint-Hilaire[1,2][0009−0003−8139−3992], Christopher Neal[1,2][0000−0002−6953−8728], Frédéric Cuppens[1][0000−0003−1124−2200], Nora Boulahia-Cuppens[1][0000−0001−8792−0413], and Makhlouf Hadji[2][0000−0003−1048−753X]

[1] Polytechnique Montréal Montréal, Canada
[2] IRT SystemX Palaiseau, France

Abstract. As cyberattacks continue to increase, detecting and performing remediation actions against them is essential. This paper presents an approach to automate the countermeasures selection process to deal with a vulnerability exploitation performed by a cyberattack. We propose an approach to match two knowledge graphs, one from a vulnerability ontology, Vulnerability Description Ontology (VDO), and the other is the countermeasures knowledge graph, D3FEND, to mitigate cyberattack impacts. Our approach uses machine learning and an inference system to match entities from VDO and D3FEND to select candidate countermeasures to an attack. Our contribution aims to automatically select countermeasures intended to be part of an incident response playbook for a vulnerability. We show our approach application to a WannaCry use-case scenario. We validate our countermeasures selection approach by comparing the countermeasures automatically selected with those proposed in the literature for a WannaCry attack.

Keywords: Graph Matching · Machine Learning for Cybersecurity · Incident Response

1 Introduction

As cyberattacks increase, advancing our capabilities to thwart them is crucial. In this paper, we consider cyberattacks that are performed by exploiting a vulnerability. Detection is the first step in stopping a cyberattack. An Intrusion Detection System (IDS) generates an alert when it detects a cyberattack. A Security Information and Event Management (SIEM) system correlates the alerts an IDS generates. The next step consists of selecting appropriate countermeasures to manage the attack. As there are many alerts, countermeasures must be automatically activated. Security Orchestration, Automation, and Response (SOAR) systems represent a step towards automation; a SOAR allows one to write playbooks and scripts to execute to cope with an attack. However, presently, experts manually write playbooks, and a SOAR requires extensive configuration.

Integrating an organization's security tools within the SOAR is a substantial task; it is even more complex when an organization has several instances

of tools from different vendors. The SOAR configuration requires a significant investment in time and financial resources, and there is no standard to ease SOAR interoperability. To reduce the manual work experts should do regarding SOAR configuration, we propose to automatically select countermeasures aimed to generate automated playbooks that will be integrated into a SOAR.

Resolving this gap in cyberattack response automation consists of automatically identifying which countermeasures are effective in dealing with a cyberattack. We address this problem in this paper. As a system is monitored in real-time, metadata of generated alerts of detected intrusions is available. By semantically mapping this metadata with system knowledge, such as existing vulnerabilities, it is possible to determine the exploited vulnerabilities. Multiple vulnerability databases exist that contain the required conditions for the exploitation of vulnerabilities as well as the post-conditions of their exploitation.

In order to mitigate adversary actions and block any possible future action, it is necessary to know which countermeasures are related to the consequences of vulnerability. We propose to generate the individuals for each vulnerability description information to create a Knowledge Graph (KG) for the Vulnerability Description Ontology (VDO)[1], proposed by the National Institute of Standards and Technology (NIST). Then, the data can be available and machine-readable for automation tasks. Additionally, a popular KG of countermeasures exists, D3FEND[2]. We propose matching the VDO KG with D3FEND to identify the countermeasures to mitigate an ongoing vulnerability exploitation.

To perform the proposed KG matching we utilize a knowledge base of vulnerabilities, the targets of cyberattacks, and defensive countermeasures.

- **VDO** To represent vulnerabilities, we use VDO, a standardized vulnerability ontology proposed by NIST. VDO represents various attributes for characterizing software vulnerabilities.
- **Digital Artifact Ontology (DAO)**. DAO is used to represent the target of an attack. DAO is an ontology that specifies the concepts needed to classify and represent digital objects of interest for cybersecurity analysis. The use of DAO makes it possible to associate the offensive techniques offered by ATT&CK[3] with the defensive techniques of D3FEND.
- **D3FEND** To select countermeasures, we use D3FEND, a KG created by MITRE that describes specific technical functions within cyber technologies in a common language of defensive techniques. The D3FEND taxonomy inherits artifacts from DAO. ATT&CK is incorporated into the D3FEND KG by mapping its concepts directly to D3FEND's defensive techniques and artifacts model. Throughout this paper, when mentioning D3FEND, we also reference the inherited concepts from DAO and ATT&CK.

Our contributions consist of creating corpora based on entities from VDO and D3FEND, matching each impact and exploitation method of a CVE with an

[1] `https://github.com/usnistgov/vulntology`
[2] https://d3fend.mitre.org/
[3] https://attack.mitre.org/

offensive technique of D3FEND, and selecting the defensive techniques related to the offensive technique for the countermeasure plan construction. We show the application of the proposed model in a real-world situation. We validate the approach by comparing the countermeasures automatically selected for a WannaCry use-case scenario with those proposed for this kind of attack.

The paper is organized as follows: Section 2 covers techniques for matching KGs. Section 3 reviews related works and compares them to our approach. Section 4 details our methodology for matching VDO and D3FEND to select countermeasures. Section 5 evaluates the approach's efficiency and demonstrates its application in a use case. Section 6 discusses the advantages and challenges of our approach. Finally, Section 7 concludes the paper.

2 Background

This section introduces the related concepts used in our approach.

Attack Graph An Attack Graph (AG) represents all the paths an adversary can take to release a detrimental event on an Information System.

Attack-Defense Graph An attack defense graph (ADG) is a directed acyclic graph in which nodes represent threats arising from existing vulnerabilities and countermeasures to mitigate these threats.

Ontologies An ontology is the concrete and formal representation of a domain. An ontology is a set of terms and the links between them. The ontology ensures that no contradictions exist between these terms. Description Logic (DL) makes it possible to represent an ontology. Ontologies allow automating inference and enabling operability between applications [12]. The open-world assumption governs ontologies and states that what is not known is assumed to be true. It is common to confuse the concepts of ontology and KG, however they have nuanced differences [11]. In the next paragraph, we explain the concept of KG.

Knowledge Graph (KG) A KG is a data graph aimed at accumulating and disseminating knowledge of the real world. A KG uses an ontology as a framework to describe a given instance of a domain. A concrete example would be creating an ontology to describe a countermeasure. The ontology comprises all the characteristics that all countermeasures share. This ontology would, therefore, have classes for the concepts of countermeasure, asset, adversarial action, and properties like mitigates. These terminologies are represented in the TBox. Definition 1 represents the TBox in the DL language. However, a knowledge graph must be created to represent a particular countermeasure, such as **Update software**. The asset mitigated by this countermeasure is a software. Nodes represent entities of interest, such as Software, and edges of potentially different relationships between nodes, such as impacts, mitigates. The constructed KG makes it possible to represent the following reality: *A countermeasure* ***Update software*** *mitigates*

an asset that is a software. This software is impacted by an adversarial action ***Run virtual instance***. This KG does not make it possible to know if **Update software** and **Run virtual instance** are two different entities, namely that the countermeasure and adversarial action classes are disjoint. It is the ontology that makes it possible to define such restrictions.

Definition 1. *Terminologies domain in DL language*

$$Countermeasure \cap AdversarialAction \equiv \bot$$
$$Countermeasure \equiv mitigates.Asset$$

SPARQL Protocol and RDF Query Language (SPARQL) KGs are often modeled using the Resource Description Framework (RDF) in the N-triple format. This is a line-based, plain text serialization format for RDF graphs. For applications to interact with data, queries must be made on KGs. SPARQL is used to query an RDF graph. SPARQL is a query language that allows searching, adding, modifying, or deleting available RDF data.

b-matching of a Graph Based on a, possibly weighted, graph with a positive integer b_v for every vertex v of the graph, a b-matching of a graph is a multiset M of its edges such that, for every vertex v, the number of edges of M incident to v does not exceed b_v [9].

Graph Matching A matching of a graph is a case of b-matching in which $b_v = 1$ for every vertex v. We use a word embedding model to proceed to the b-matching of the two KGs.

3 Related Work

In recent years, researchers have proposed several countermeasure selection approaches. However, they come with limitations that are presented in this section. To fill this gap, we base our approach on graph matching. The research into graph matching has followed several approaches. In this section, we also present the different graph-matching approaches and their limitations and explain how we address them with our work to propose an automated countermeasures selection approach.

In their survey of countermeasures selection approaches [20], Nespoli et al. define a countermeasure strategy. A countermeasure strategy comprises methodologies, procedures, and processes that aim to react to and eradicate security incidents. They present a list of necessary components to define a countermeasure strategy.

Some necessary components [20] are a monitored system, detection tools, and countermeasure knowledge. Others are a system model that synthesizes information gathered from the monitored system, atomic countermeasure options, and a list of possible countermeasures. A threat model representing attack patterns in AGs or ATs is essential. Finally, countermeasures were selected while looking for a trade-off between security level and cost of reaction. A crucial component

is a prediction reward that can lead to a model attacker decision where the threat model is updated based on all previously cited components. The system operator's decision is also important.

In our proposed approach, we consider these components. From a vulnerability report and a system model, we automatically generate an AG that is updated in real time based on system monitoring. Countermeasure knowledge is available in the literature. In this paper, we propose to select a list of possible countermeasures by correlating countermeasures knowledge with vulnerability reports. Other components, such as cost of reaction, are part of our automatically optimal security playbook generation approach [26] that can lead to the update of the threat model.

Nespoli et al. [20] survey the approaches based on 7 comparison features such as attack modeling, countermeasures provision techniques, and used standards. The surveyed approaches [8, 15, 29] highlight the lack of countermeasure standards. This issue is linked to the lack of countermeasure knowledge. The authors only present a limited set of countermeasures used to counteract specific attacks reported to the monitored system. These countermeasure selection processes lose relevance and effectiveness because they do not apply to another type of attack. The solutions also rely on the administrator's knowledge of each threat [10, 29], implying a limitation in the correlation between atomic mitigation steps and attacks.

In [8, 15], the authors use the Common Remediation Enumeration (CRE) standard to develop a countermeasure model. Nespoli et al. argue that CRE can not contribute significantly to the automation of countermeasures selection. Kotenko et al. [15] also assume that the system already has a pool of countermeasures that can be selected for an ad hoc algorithm. This assumption requires much effort from the expert filling the knowledge database of countermeasures. Our countermeasure selection approach is based on countermeasure and vulnerability standards. Using D3FEND, security experts do not have to fill any countermeasures knowledge base.

Our countermeasure selection approach is part of an ADG generation process. Compared with the approach in [28], our approach selects countermeasures for a system monitored in real-time instead of a formal system model. The AG generated in an initial state is updated in real-time based on newly deducted knowledge from VDO [5]. We choose this ontology because it is a standardized ontology proposed by NIST. It allows the representation of vulnerabilities based on the natural language description of a vulnerability in the National Vulnerability Database (NVD).

VDO is an ontology that describes vulnerabilities, the required conditions for exploitation, the consequences, and the product concerned. In our approach, we implement VDO in DL and automatically generate individuals for specific vulnerabilities to create the KG. In order to select appropriate countermeasures to a vulnerability, we need other knowledge bases focusing on countermeasures. We investigate several countermeasure knowledge bases such as the Security

Control Catalogue ITSG-33 (from the Canadian Centre for Cyber Security) [6], D3FEND, and CIS Security Controls [7].

In [6], the Canadian Centre for Cyber Security provides security control definitions suitable for Departmental Security Officers (DSOs), IT security coordinators, and security practitioners. Each security control provides guidance on the best practices the security department should have in developing departmental and domain security control profiles, as well as, defining and implementing departmental IT security functions. In our approach, we focus on alleviating the work of security practitioners by automatically selecting countermeasures aimed to create automatic playbooks that can be integrated into a SOAR. Therefore, we discard these security controls.

In [14], Kaloroumakis et al. present a countermeasure knowledge graph, D3FEND. The Digital Artifact Ontology (DAO) is an ontology that specifies the concepts needed to classify and represent digital objects of interest for cybersecurity analysis. The use of DAO makes it possible to associate the offensive techniques proposed by ATT&CK with the defensive techniques of D3FEND based on the relationship of each technique with digital objects.

The CIS controls aim to share insights about attacks and attackers to identify root causes and translate them into defensive action classes. Each protective measure is general advice to enhance an organization's security. In comparison, the D3FEND defensive techniques provide more granularity by proposing specific actions applicable to a specific artifact. Additionally, D3FEND is standardized and machine-readable, which implies that integration for automated countermeasures selection is more accessible than the other knowledge-based methods.

In [22], Pershina et al. present an algorithm for aligning instances in large knowledge bases using Holistic Entity Matching (HolisticEM). Their approach involves generating entity pairs from two KGs and matching them based on attribute similarity. However, this approach's heuristic optimization phase results in varying matches between executions. Our approach aims to obtain a stable and equivalent output from a graph-matching procedure.

In [4], Azmy et al. propose an approach to match entities from two different KGs focusing on ambiguous entities from DBpedia and Wikidata. The approach consists of matching an entity from DBpedia with an entity in Wikidata corresponding to the same real-world entity and vice versa. The datasets are created thanks to existing cross-ontology links (i.e OWL:sameAs predicate) between DBpedia and Wikidata. They focus on entities from the KGs that share the same name, for example, the foaf:name predicate in DBpedia and the rdfs:label predicate in Wikidata. This approach is not adaptable to our needs because there are no existing cross-ontology links between VDO and D3FEND.

In [23], Portish et al. propose an approach to align graphs through a graph embedding algorithm. The ontologies are embedded, and an approach known as absolute orientation aligns the two embedding spaces. To match the two KGs, they use the Euclidian distance to assign each node of a graph to its closest node in the other graph. However, the approach only works well on similarly structured

graphs because they only match entity with entity. In our approach, we are matching a sub-graph from VDO with a sub-graph from D3FEND, and the sub-graphs are structured differently using cosine distance defined in Appendix A.

Currently, there is no contribution in the cybersecurity domain that proposes to match KGs. However, some existing contributions propose the matching of behavioral graphs in cybersecurity and the matching of AGs. In [21], Park et al. propose a malware classification method based on maximal joint sub-graph detection. In [16], Li et al. propose an approach to automatically extract behavior-based AGs from CTI reports and identify the associated attack techniques. They identify the associated techniques by matching the AGs with technique templates.

Hung et al., in [13], also propose an approach to address the radicalization detection problem. They propose graph pattern matching to be used to track individual-level indicators using data merged from available public and government/law enforcement databases. The approach provides a quantification method that allows for checking the occurrence of the indicators that are beneficial in prioritizing investigative efforts and resources for planned attack prevention.

In [22], the resulting matching varies from one case to another. Our approach aims to obtain a stable output from the graph-matching process. The methods proposed in [4] and [23] are promising; however, they are not adaptable to our goal as the approaches are limited to either KGs that share exiting cross-ontology links or KGs structured similarly. In the approaches in [13] and [21], the researchers propose matching behavioral graphs to prevent adversarial actions against a system or a government. In this paper, we propose a solution that can be adapted to KGs representing attacker behavior and countermeasures actions to block attackers' actions.

4 Methodology

4.1 Overview

We propose an approach for selecting countermeasures based on KG matching. This approach allows selecting countermeasure actions for an attack scenario in our ADG generation process, represented in Figure 1. The components in blue in Figure 1 represent the ones involved in our AG enrichment process, part of another contribution [25]. In Step 1, we scan the system. In Step 2, the scanning output serves as input to a logical reasoner, allowing the AG generation to occur in Step 3. When an adversary exploits a vulnerability, as shown in Figure 2, the procedural reasoner matches the alert information with the AG information in Step 4. In Step 5, the vulnerability ontology infers new information. When the procedural reasoner receives the inferred information in Step 6, it releases new rules to the logical reasoner in Step 7 and generates an enriched AG with a new attack path in Step 3.

After, as shown in Figure 2, the ADG generation process can start. The components in gray in Figure 1 are involved in the ADG generation process. Even if

the procedural reasoner does not release the AG enrichment process, the ADG generation process starts as shown in the flow chart presented in Figure 2. An ADG is generated in Step 10 by mapping the AG with an Incident Response (IR) playbook for the exploited vulnerability in Step 9. An IR playbook consists of the steps and procedures an organization should follow when addressing and mitigating occurred incidents. If there is no existing playbook for this vulnerability, our solution automatically generates it from a list of countermeasures as shown in Step 8 of Figure 1.

If the playbook is not generated because of the countermeasures absence for the exploited vulnerability, the automated countermeasures selection process is executed (see Figure 2). The components involved in the countermeasures selection are in white, as well as the output of the countermeasures selected. The dashed box represents the graph-matching process. The graph matching involves VDO and D3FEND KGs. Humans can periodically execute the graph matching process, as well as the scanning of the system. However, it is automatically released for a specific vulnerability exploited when there are no countermeasures available for this vulnerability. It outputs the countermeasures in Step 7 that are required for generating the playbook necessary for the ADG generation. In this paper, we focus on the selection of countermeasures through graph matching. In the next section, we present the dataset involved in the graph-matching process.

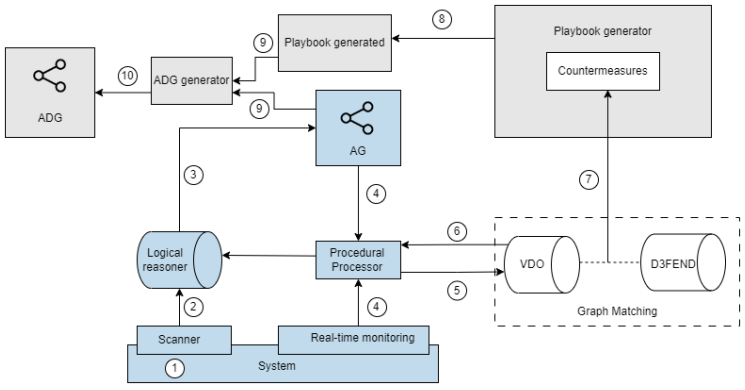

Fig. 1. Our ADG generation process

4.2 Experimental Dataset

The dataset in our approach consists of a sub-graph of VDO KG and a sub-graph of D3FEND KG. VDO describes the pre and post-conditions of a vulnerability exploitation. The countermeasures KG, derived from D3FEND [14], provides corrective actions to perform in the face of exploitations of particular systems.

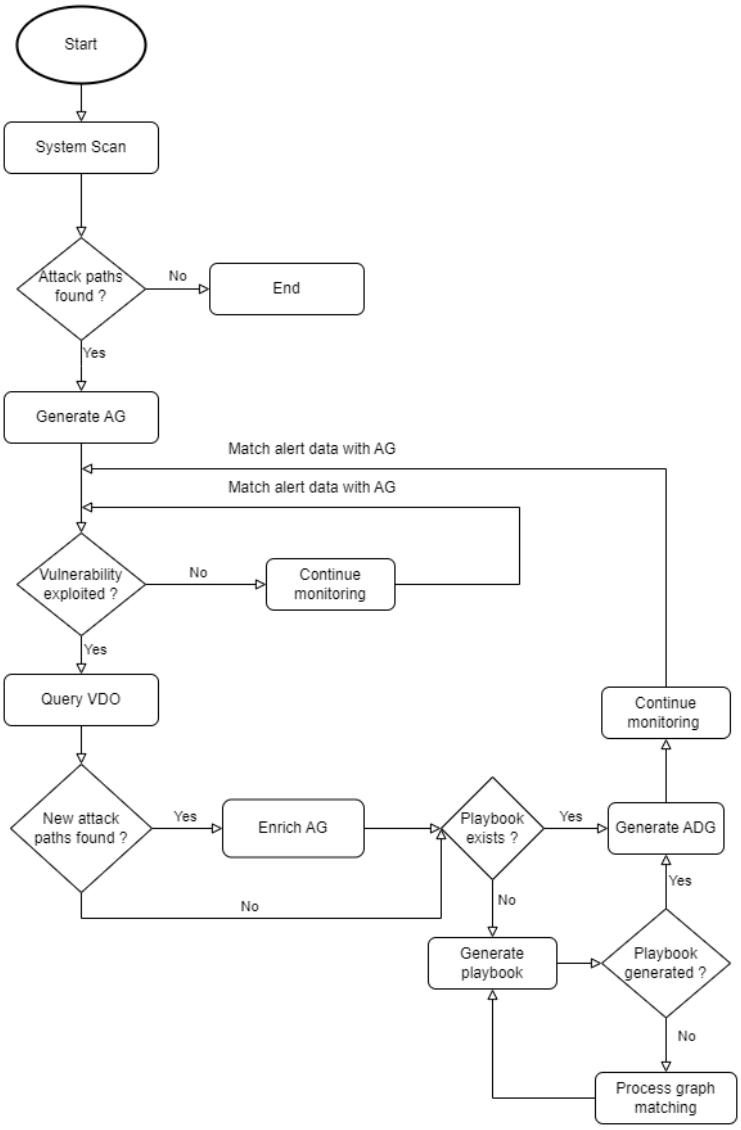

Fig. 2. A flow chart of the ADG generation process

We analyze VDO and D3FEND to choose the classes of interest for the countermeasures selection approach. Figure 3 represents the selected sub-graphs for D3FEND and VDO, respectively.

In D3FEND, these classes are *OffensiveTechnique*, *Artifact*, and *DefenseTechnique*. We select these classes to simplify the graph embedding and avoid noise in the matching process. The class *OffensiveTechnique* refers to the methods an ad-

versary can employ to attack a system and the impacts of the attack. The class *Artifact* refers to the components affected by an offensive technique, and the class *DefensiveTechnique* refers to the countermeasures that should be applied to an artifact. We, therefore, select the sub-graph composed of all the entities whose type is either *OffensiveTechnique*, *DefensiveTechnique* and *Artifact* and the properties allowing the relation between those entities.

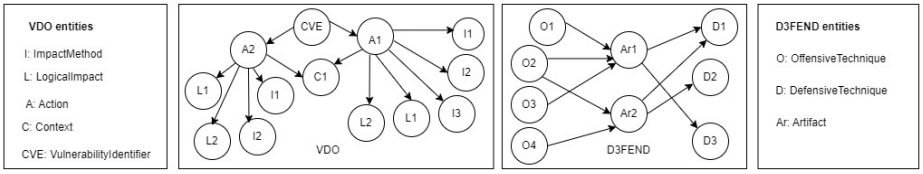

Fig. 3. VDO and D3FEND sub-graphs

In VDO, we choose the classes *LogicalImpact*, *ImpactMethod*, *Action*, *Context*, and *VulnerabilityIdentifier*. We do not choose the other classes to simplify the graph embedding and avoid noise in the matching process. These classes are essential to understanding the assets of a system affected by a scenario, how an adversary can exploit a vulnerability and the consequences of this vulnerability. *LogicalImpact* refers to the consequences of exploiting a vulnerability. *ImpactMethod* relates to the methods applied by an adversary to exploit a vulnerability. *Context* refers to the software and hardware concerned by a vulnerability. *VulnerabilityIdentifier* represents the CVE ID. However, some of the classes are not directly linked. Therefore, we consider the class *Action* to make the complete meaningful sub-graph required for the matching. The class *Action* allows linking an entity of the class *VulnerabilityIdentifier* with entities of the classes *ImpactMethod* and *LogicalImpact*.

4.3 Graph Matching

Our graph matching approach takes as input two KGs, O and $O^{'}$. O contains the set of classes $O = C_O^0, C_O^1, .., C_O^i$ and each class of O contains the set of entities $C_O^i = e_0^i, e_1^i, ..., e_n^i$. Similarly, $O^{'}$ contains the set of classes $O^{'} = C_{O'}^0, C_{O'}^1, .., C_{O'}^k$ and each class of O contains the set of entities $C_{O'}^k = e_0^k, e_1^k, ..., e_n^k$. We base our matching approach on graph embedding, word embedding, and SPARQL queries.

Figure 4 gives an overview of our solution architecture. The input is the two subgraphs of D3FEND and VDO KGs. The automated pre-processing phase of the KGs involves parsing the KGs, text processing, and corpora creation. This phase is necessary before starting the training of Word2Vec (see Appendix A) models and the embedding of KGs. Afterward, the matching process starts automatically.

In the matching phase, the cosine similarity is calculated automatically between the offensive techniques from D3FEND and the impact and method from VDO. A SPARQL query then allows the automated retrieval of artifact entities linked to the offensive techniques. If no artifact entity is retrieved, the cosine similarity is automatically calculated between the artifact entities of D3FEND and the context entities of VDO to find the artifact entities necessary to query candidate countermeasures for each CVE. The output from the matching process consists of a list of candidate countermeasures for each CVE.

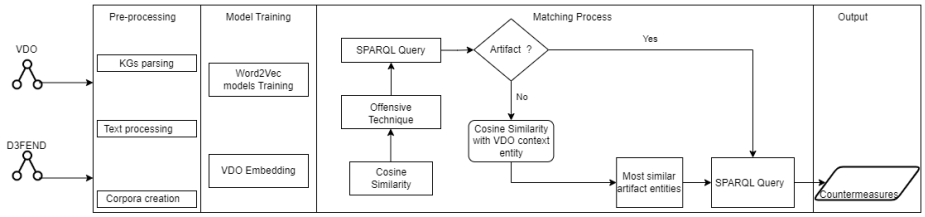

Fig. 4. Architecture of the proposed solution

4.4 Matching Models Validation

The value of the parameters of a Word2Vec model impacts its performance. Window size is the size of the context window used to predict surrounding words. A smaller window size captures a more specific, local context, which leads to more precise word associations. We fix the window size to 5. The value of min_count is a threshold, where words that appear fewer times than the value are ignored during training; we fix its value to 1. The number of workers refers to the number of CPU cores used for parallel processing. A higher workers number leads to faster training. We fix the number of workers to 20. The vector size is the dimensionality of the word vector. Higher dimensions capture more semantic nuances and context but can lead to overfitting. The number of iterations over the training corpus is the number of epochs. Fewer epochs reduce training time but can lead to underfitting. More epochs allow the model to learn better and converge more fully.

We perform different Word2Vec model training for the method and impact corpora by modifying the vector size and the number of epochs. To choose the best Word2Vec model, we consider the number of VDO entities for which we get the correct matches and the similarity score for the matches with the most similar D3FEND entity. Table 1 compares different models based on the number of correct matching for impacts and methods in VDO and the average cosine similarity score for the correct prediction. The blue rows in Table 1 indicate the best models for *LogicalImpact* and *ImpactMethod*, respectively. For 250 vector size and 150 epochs, 100% of the predictions for the entities of *LogicalImpact* are correct. Most importantly, for this model, the average similarity score is

Table 1. Word2Vec model prediction evaluation

Entity class	vector size	epochs	Percentage of correct predictions	Average Similarity Score
LogicalImpact	250	150	100%	0.46
ImpactMethod	250	150	50%	0.5
LogicalImpact	250	50	100%	0.45
ImpactMethod	250	50	100%	0.69
LogicalImpact	250	200	75%	0.38
ImpactMethod	250	200	25%	0.25
LogicalImpact	400	50	100%	0.44
ImpactMethod	400	50	50%	0.33
LogicalImpact	400	150	100%	0.45
ImpactMethod	400	150	75%	0.43
LogicalImpact	400	200	75%	0.38
ImpactMethod	400	200	50%	0.5
LogicalImpact	200	150	100%	0.44
ImpactMethod	200	150	50%	0.5
LogicalImpact	200	50	100%	0.44
ImpactMethod	200	50	50%	0.38
LogicalImpact	200	200	75%	0.5
ImpactMethod	200	200	25%	0.25

higher. The average similarity score is less than 0.5 because the higher similarity score for the match of privilege escalation is 0.33. This value low similarity score for this entity is due to the impact of corpus size. Since the dataset is small, the model lacks variety, leading to poor generalization. For more epochs, the model learns specific patterns and noise from the training data rather than the underlying trends, which results in overfitting. We, therefore, fix the threshold value for the similarity score to 0.33.

4.5 Countermeasure Selection

To extract the entities required for the model's training and the matching process, we parse the KGs using techniques such as SPARQL queries and splitting. We create an impact corpus composed of the entities of *ImpactTechnique* and *PrivilegeEscalationTechnique*, both subclasses of *OffensiveTechnique*, and *LogicalImpact*. We also create a method corpus composed of the entities of the other subclasses of *OffensiveTechnique* and the entities of the classes *ImpactMethod*. We create an artifact corpus with the entities of *Artifact* and *Context*. A general corpus is created with the entities of all the classes.

We train 2 Word2Vec models with the impact and method corpora. We fix the parameters for the best model in Table 1 in Section 4.4 for Word2Vec impact and method models. A Word2Vec model receives a corpus text and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns the vector representation of words. To get the embeddings of the KGs with RDF2Vec, we use the general corpus for the Word2Vec embedder.

We get the literals of the VDO graph embedding for each CVE. Then, for each entity classified as an impact from VDO, we calculate the cosine similarity between this entity and each offensive technique entity of the impact corpus

using the trained model for the impact corpus. As output, we get the most similar offensive technique for the impact. We also calculate the cosine similarity between each entity categorized as a method from VDO and each offensive technique entity of the method corpus using the trained model for the exploitation method corpus. The output consists of the most similar offensive technique for the exploitation method.

The similarity score varies from 0 to 1. When the score is equal to 1 for an entity, this means that this entity is similar to the input entity. However, it is common for an entry entity to have several matches. We choose the one whose score is higher than the given similarity threshold from those entities. Therefore, if all entity scores exceed this threshold, we keep them all. Then, we execute a SPARQL query on D3FEND KG to get the artifact entities related to the offensive techniques. However, it is possible not to get a specific artifact entity from the SPARQL query. In this case, we calculate the cosine similarity for each artifact's entities of D3FEND for the context entity of each CVE using a trained Word2Vec model with the artifact corpus. Then, we get the artifact with the highest score as output. Afterward, we proceed to a SPARQL query to get all the countermeasures related to the artifact entities.

5 Implementation Results

5.1 Countermeasures Selection Evaluation

We use the trained models highlighted in blue from Table 1 for the countermeasures selection process[4]. Our solution automatically executes a SPARQL query on D3FEND for each selected offensive technique to retrieve related artifacts and their defensive techniques. If no artifact is linked to an offensive technique, cosine similarity is used to find the top 5 matching artifacts that match the context entity in VDO most. A SPARQL query follows to obtain the corresponding defensive techniques for each match. We evaluate the efficiency of the countermeasures selected using the precision, recall, and F1 score metrics, defined in Appendix A.

We count the FN, TP, and FP for the defensive techniques matched with each method and impact. Then, we calculate the precision, recall, and F1 score. Finally, we calculate the macro-value for these metrics by calculating the average recall, precision, and F1 score value for the method and impact, respectively. Table 2 and 3 represents the value for these metrics for the methods and impacts respectively.

The average F1 score for the countermeasures selected for both the impacts and the methods is higher than 0.9. Thus, we conclude that our approach for automatically selecting countermeasures is efficient.

Table 2. Evaluation of the prediction for the methods

Metric / Impact Method	Precision	Recall	F1 Score
Code Execution	1	0.75	0.86
Man-in-the-Middle	1	1	1
Authentication Bypass	1	1	1
Trust Failure	1	1	1
Average	1	0.94	0.97

Table 3. Evaluation of the prediction for the impacts

Metric / Logical Impact	Precision	Recall	F1 Score
Manipulation	1	1	1
Discovery	1	0.88	0.93
Shutdown	1	1	1
Privilege Escalation	1	1	1
Average	1	0.97	0.98

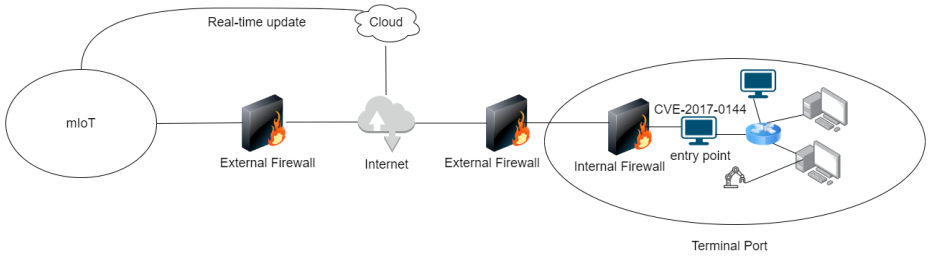

Fig. 5. A Maritime Transportation System use case

5.2 Illustrative Use Case

This section introduces an illustrative use case represented in Figure 5 to validate our approach. We choose a WannaCry use case because ransomware is the attack that has impacted most organizations recently [18, 27]. The use case concerns a Maritime Transportation System (MTS). We choose an MTS because this is a critical sector; in the past, attacks such as the one impacting Maersk [19] have shown how a ransomware attack against an MTS organization can have a severe financial impact.

There is a terminal port with several workstations and a robot hoist that unloads merchandise from ships. The terminal port receives data from the cloud through an internet connection. External and internal firewalls exist between the port network and the internet. The Maritime Internet of Things (IoT) system communicates in real time with the cloud. A user on the entry point machine vulnerable to EternalBlue (CVE-2017-0144) downloads a malicious input with the WannaCry ransomware. The WannaCry propagates to all vulnerable local hosts via port 445, encrypts files, and ransom requests follow. The terminal port must disconnect from the internet and work in degraded mode. There is no

[4] `https://github.com/phDimplKS/graph-matching`

communication between the terminal port and the mIoT system. Let us explain

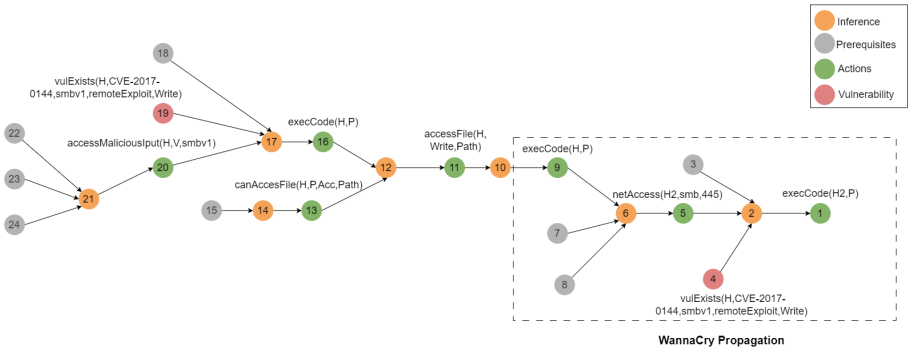

Fig. 6. An AG generated for the use case scenario

how our architecture integration allows us to deal with this scenario. For this use case scenario, an AG is generated first, as shown in Figure 6. The node in the dashed box represents the WannaCry propagation on the vulnerable local hosts. Thanks to detection rules created on an IDS installed over the SIEM, the SIEM allows the detection of EternalBlue exploit attempts : alert tcp any any $->$ any 445 (msg:"SURICATA **SMB** Trans2 Request"; flow:to_server,established; content:"|FF 53 4D 42 32|"; depth:5; content:"|00 00|"; distance:30; within:2; content:"|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|"; distance:10; within:16; reference:cve,2017-0143; classtype:attempted-admin; sid:1000003; rev:1;).

Our solution maps the alert information with the AG generated for the scenario. The alert is mapped with the Node 19 representing the EternalBlue vulnerability on the entry point. Then, it launches the countermeasures selection process to get the related countermeasures that can block the attacker from launching the ransomware. This is the list of countermeasures proposed by our solution for CVE-2017-0144:

- There are countermeasures about the smbv1: Asset Vulnerability Enumeration, Software Update, and Restore Software.
- Some countermeasures aim to control files allowed to be downloaded or executed: File Encryption, Local File Permissions, Decoy File, File Analysis, File Removal, File Integrity Monitoring, Restore File, Dynamic Analysis, Emulated File Analysis, Executable Allowlisting and Executable Denylisting.
- Other countermeasures concern the network traffic: Client-server Payload Profiling, Network Traffic Community Deviation, Per Host Download-Upload Ratio Analysis, Protocol Metadata Anomaly Detection, Remote Terminal Session Detection, User Geolocation Logon Pattern Analysis, and Network Traffic Filtering.

– Several countermeasures selected concern a process: Process Spawn Analysis, Hardware-based Process Isolation, Mandatory Access Control, System Call Analysis, and System Call Filtering.

6 Discussion

We compare the countermeasures selected by our approach with the countermeasures proposed in the literature for ransomware [1,3,18] and, more precisely, WannaCry [2] use cases. In the literature, most of the proposed countermeasures concern prevention to avoid infection. The proposed actions consist of setting up spam filters to quarantine suspicious emails and attachments [1] and using predictive models to detect malicious behavior [3]. Other countermeasures proposed frequently are patching the vulnerability [3], excluding kill-switch domains from firewall rules, and blocking the SMB ports [2]. Our approach selects countermeasures that are part of the prevention phase but also from the containment, eradication, and recovery phases. File removal is an action from the eradication phase, which consists of removing the ransomware from the system. The recovery of encrypted files may follow.

All the countermeasures mentioned in Section 5.2 are the candidate countermeasures to the mitigation plan. This paper only aims to select countermeasures, a basis for the playbook generation that is part of another contribution [26]. Thanks to the real-time generation, the AG will help determine which countermeasure actions should be applied quickly based on the attacker's position knowledge. So, based on how far the attacker has gone, only some countermeasures will be instantiated to the AG, leading to the ADG generation, which is part of future work. In the future, we will evaluate our approach performance to select countermeasures in real-time for different system complexity, particularly for large-scale infrastructure for which many security incidents may be generated. We will consider the impact of real-time countermeasure selection performance on the ADG generation process.

7 Conclusion

As part of our ADG generation, we propose a countermeasures selection approach based on graph matching. The approach matches a cybersecurity vulnerability KG, VDO, with a countermeasures KG, D3FEND, thanks to 4 corpora creation for the Word2Vec models training. Graph and word embedding follow to calculate the cosine distance between the entities of the two KGs. The similarity between the entities from the two KGs allows us to select countermeasures. However, some candidate countermeasures only apply to specific assets. The countermeasures are also different in terms of complexity and impact. The selected countermeasures are used for automated playbook generation based on system and organization constraints. Using the AG will help us to determine at which point of the network the playbook actions should be applied to block the adversary from advancing in the system.

Acknowledgements This work was supported by the Mitacs Accelerate International program, IRT SystemX, and the Cyber Resilience of Transport Infrastructure and Supply Chains (CRITiCAL) Chair.

References

1. Adamov, A., Carlsson, A.: The state of ransomware. trends and mitigation techniques. 2017 ieee east-west design test symposium(ewdts), 1-8 (2017)
2. Aljaidi, M., Alsarhan, A., Samara, G., Alazaidah, R., Almatarneh, S., Khalid, M., Al-Gumaei, Y.A.: Nhs wannacry ransomware attack: Technical explanation of the vulnerability, exploitation, and countermeasures. In: 2022 International Engineering Conference on Electrical, Energy, and Artificial Intelligence (EICEEAI). pp. 1–6. IEEE (2022)
3. Alshaikh, H., Ramadan, N., Ahmed, H.: Ransomware prevention and mitigation techniques. Int J Comput Appl **177**(40), 31–39 (2020)
4. Azmy, M., Shi, P., Lin, J., Ilyas, I.F.: Matching entities across different knowledge graphs with graph embeddings. arXiv preprint arXiv:1903.06607 (2019)
5. Booth, H., Turner, C.: Vulnerability description ontology (VDO): a framework for characterizing vulnerabilities. Tech. rep., National Institute of Standards and Technology (2016)
6. CCCS: Security control catalogue (2012), `https://www.cyber.gc.ca/en/guidance/annex-3a-security-control-catalogue-itsg-33`
7. CIS: Cis critical security controls (2024), `https://www.cisecurity.org/controls`
8. Doynikova, E., Kotenko, I.: Countermeasure selection based on the attack and service dependency graphs for security incident management. In: Risks and Security of Internet and Systems: 10th International Conference, CRiSIS 2015, Mytilene, Lesbos Island, Greece, July 20-22, 2015, Revised Selected Papers 10. pp. 107–124. Springer (2016)
9. Emek, Y., Kutten, S., Shalom, M., Zaks, S.: Hierarchical b-matching. In: Bureš, T., Dondi, R., Gamper, J., Guerrini, G., Jurdziński, T., Pahl, C., Sikora, F., Wong, P.W. (eds.) SOFSEM 2021: Theory and Practice of Computer Science. pp. 189–202. Springer International Publishing, Cham (2021)
10. Gupta, M., Rees, J., Chaturvedi, A., Chi, J.: Matching information security vulnerabilities to organizational security profiles: a genetic algorithm approach. Decision Support Systems **41**(3), 592–603 (2006)
11. Hogan, A., Blomqvist, E., Cochez, M., D'amato, C., Melo, G.D., Gutierrez, C., Kirrane, S., Gayo, J.E.L., Navigli, R., Neumaier, S., Ngomo, A.C.N., Polleres, A., Rashid, S.M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., Zimmermann, A.: Knowledge graphs. ACM Comput. Surv. **54**(4) (jul 2021). `https://doi.org/10.1145/3447772`
12. Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C.: A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0. University of Manchester (2004)
13. Hung, B.W.K., Jayasumana, A.P., Bandara, V.W.: Detecting radicalization trajectories using graph pattern matching algorithms. In: 2016 IEEE Conference on Intelligence and Security Informatics (ISI). pp. 313–315 (2016)
14. Kaloroumakis, P.E., Smith, M.J.: Toward a knowledge graph of cybersecurity countermeasures. Tech. rep., Technical report (2021)

15. Kotenko, I., Doynikova, E.: Countermeasure selection in siem systems based on the integrated complex of security metrics. In: 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. pp. 567–574. IEEE (2015)
16. Li, Z., Zeng, J., Chen, Y., Liang, Z.: Attackg: Constructing technique knowledge graph from cyber threat intelligence reports. In: European Symposium on Research in Computer Security. pp. 589–609. Springer (2022)
17. Ma, L., Zhang, Y.: Using word2vec to process big text data. In: 2015 IEEE International Conference on Big Data (Big Data). pp. 2895–2897. IEEE (2015)
18. Malik, A.W., Anwar, Z., Rahman, A.U.: A novel framework for studying the business impact of ransomware on connected vehicles. IEEE Internet of Things Journal **10**(10), 8348–8356 (2023). https://doi.org/10.1109/JIOT.2022.3209687
19. Mos, M.A., Chowdhury, M.M.: The growing influence of ransomware. In: 2020 IEEE International Conference on Electro Information Technology (EIT). pp. 643–647 (2020). https://doi.org/10.1109/EIT48999.2020.9208254
20. Nespoli, P., Papamartzivanos, D., Gómez Mármol, F., Kambourakis, G.: Optimal countermeasures selection against cyber attacks: A comprehensive survey on reaction frameworks. IEEE Communications Surveys & Tutorials **20**(2), 1361–1396 (2018). https://doi.org/10.1109/COMST.2017.2781126
21. Park, Y., Reeves, D., Mulukutla, V., Sundaravel, B.: Fast malware classification by automated behavioral graph matching. In: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research. CSIIRW '10, Association for Computing Machinery, New York, NY, USA (2010)
22. Pershina, M., Yakout, M., Chakrabarti, K.: Holistic entity matching across knowledge graphs. In: 2015 IEEE International Conference on Big Data (Big Data). pp. 1585–1590. IEEE (2015)
23. Portisch, J., Costa, G., Stefani, K., Kreplin, K., Hladik, M., Paulheim, H.: Ontology matching through absolute orientation of embedding spaces. In: The Semantic Web: ESWC 2022 Satellite Events: Hersonissos, Crete, Greece, May 29–June 2, 2022, Proceedings, pp. 153–157. Springer (2022)
24. Ristoski, P., Paulheim, H.: Rdf2vec: Rdf graph embeddings for data mining. In: The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I 15. pp. 498–514. Springer (2016)
25. Saint-Hilaire, K., Cuppens, F., Cuppens, N., Garcia-Alfaro, J.: Automated enrichment of logical attack graphs via formal ontologies. In: Meyer, N., Grocholewska-Czuryło, A. (eds.) ICT Systems Security and Privacy Protection. pp. 59–72. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-56326-3_5
26. Saint-Hilaire, K., Cuppens, F., Cuppens-Boulahia, N., Hadji, M.: Optimal automated generation of playbooks. 38th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSec'24) (2024)
27. Šulc, V.: Current ransomware trends. International Days of Science **31** (2021)
28. Swarup, V.: Remediation graphs for security patch management. In: IFIP International Information Security Conference. pp. 17–28. Springer (2004)
29. Viduto, V., Maple, C., Huang, W., López-Peréz, D.: A novel risk assessment and optimisation model for a multi-objective network security countermeasure selection problem. Decision Support Systems **53**(3), 599–610 (2012)
30. Zhang, Y., Jin, R., Zhou, Z.H.: Understanding bag-of-words model: a statistical framework. International journal of machine learning and cybernetics **1**, 43–52 (2010)

Appendix A Additional Background Definitions

Word2Vec Word2Vec [17] is a popular word embeddings model used to address the limitations of the bag-of-words model [30], which is a type of vector space model that simplifies text data representation in Natural Language Processing (NLP) and Information Retrieval (IR). A bag-of-words vector represents text describing the occurrence of words within a document. In Word2Vec, each token becomes a vector with the length of a determined number.

RDF2Vec RDF2Vec, created by Ristoski et al. [24], is an unsupervised technique built on Word2Vec. RDF2Vec first creates sentences that can be fed to Word2Vec by extracting walks of a certain depth from a KG to make the embedding. A vector of latent numerical features represents each entity in the KG. We calculate the similarity of the vectors to match their embeddings using a distance metric.

Cosine Similarity Cosine similarity is a metric for measuring distance when the magnitude of the vectors does not matter. Mathematically, cosine similarity calculates the cosine of the angle between two vectors projected in a multi-dimensional space. Considering two vectors A and B; we can measure their cosine similarity using Formula 1.

$$\cos(AB) = \frac{A.B}{||A||||B||} \tag{1}$$

Where, $A.B$ is the dot product of the vectors A and B, $||A||$ and $||B||$ are the length (magnitude) of the two vectors A and B, and $||A||||B||$ is the regular product of the vectors A and B.

If $A = B$, $\cos(AB) = 1$; in this case A and B are fully similar. If $A.B$=0, then A and B are in opposite directions, so $A.B$ is negative, and one or both vectors are zero vectors, $\cos(AB) = 0$; in this case A and B are opposite.

Precision Precision measures the number of positive prediction correctly predicted. So, it is calculated by dividing the number of true positive prediction (TP) by all positive prediction i.e. True Positive (TP) + False Positive (FP).

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

Recall Recall gives a percentage of true positives instances by a model. It is the number of well predicted positives divided by the total number of positives (True Positive + False Negative (FN)).

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

F1 Score Either precision and recall can not evaluate a machine learning model separately. F1 score allows combining precision and recall. So, it can provide a good evaluation of a model performance. It is calculating as follow:

$$F1Score = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision}$$