POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Automatic Generation of Attack and Remediation Graphs

KÉREN A. SAINT-HILAIRE
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Février 2025

© Kéren A. Saint-Hilaire, 2025.

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Automatic Generation of Attack and Remediation Graphs

présentée par **Kéren A. SAINT-HILAIRE**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Tarek OULD-BACHIR, président
Frédéric CUPPENS, membre et directeur de recherche
Nora BOULAHIA-CUPPENS, membre et codirectrice de recherche
Omar ABDUL WAHAB, membre
Hakima OULD-SLIMANE, membre externe

DEDICATION

To my grandmother, Thérèse Isnardin,
And to my parents, Rébecca and Alfred.

ACKNOWLEDGEMENTS

I want to thank Dr. Frédéric Cuppens, my research director, Dr. Nora Boulahia-Cuppens, my research co-director, Dr. Christopher Neal, Dr. Francesca Bassi, and Dr. Makhlouf Hadji for their guidance and support throughout this research.

I sincerely thank Dr. Tarek Ould-Bachir, Dr. Omar Abdul Wahab, and Dr. Hakima Oul-Slimane for accepting to be part of the jury.

I also sincerely thank Dr. Joaquin-Garcia Alfaro and Dr. Reda Yaich. Their collaboration and insightful advice have significantly enriched this research.

This research was made possible with financial and material support from the Mitacs Accelerate International program, IRT SystemX, and the Cyber Resilience of Transport Infrastructure and Supply Chains (CRITiCAL) Chair.

Last, I thank my family and friends who have been beside me during this journey.

RÉSUMÉ

Depuis la pandémie de COVID-19, les cyberattaques ne cessent de se multiplier en raison du recours croissant à Internet et au télétravail. Les cyberattaques peuvent avoir plusieurs origines. Cette thèse s'intéresse principalement aux cyberattaques exploitant des vulnérabilités dans les réseaux informatiques. Les entreprises doivent prendre le temps de décider quelles vulnérabilités doivent être traitées en priorité. Le nombre d'alertes remontées aux entreprises est énorme. Les experts en cybersécurité ne peuvent en traiter que certaines. Ils doivent par ailleurs décider quelles actions de réponse aux incidents doivent faire partie du plan de réponse aux incidents de l'entreprise.

Les travaux de recherche menés dans cette thèse visent à automatiser la construction du processus de plan de réponse aux incidents en tenant compte des exigences pour l'exploitation des vulnérabilités découvertes, de la composition du système et des contraintes de l'organisation. Cette approche est basée sur des graphes d'attaque. Un graphe d'attaque permet de représenter les chemins qu'un adversaire peut prendre pour causer des dommages au système.

L'objectif principal de cette thèse est de générer un playbook de réponse aux incidents automatisé en temps réel. Pour atteindre cet objectif, des activités de recherche essentielles sont définies. Ces activités sont subdivisées en quatre objectifs de recherche qui constituent la base des contributions de cette thèse.

Le premier objectif de recherche consiste à mettre à jour les graphes d'attaques en temps réel en s'appuyant sur une ontologie de vulnérabilités et une surveillance du système. Cette contribution consiste à développer un outil composé d'un module chargé de corréler les alertes générées par un outil de surveillance intégré à cet outil avec des graphes d'attaques logiques. Ce module déduit quelle vulnérabilité est susceptible d'être exploitée ou est exploitée, puis interroge l'ontologie pour trouver de nouveaux chemins d'attaque possibles menant à l'objectif de l'attaquant. Cela lance alors le processus d'enrichissement du graphe d'attaque avec de nouveaux chemins.

Cette contribution est validée grâce à un scénario de cas d'usage concernant une ville intelligente. Le but de l'attaque est de causer des dommages physiques sur le système de transports publics. Cet objectif est réalisable, car un attaquant obtient des privilèges d'administrateur permettant la modification d'informations sensibles en exploitant la vulnérabilité BlueKeep. L'exploitation de la vulnérabilité active la déduction ontologique d'un autre impact de la vulnérabilité constinuant un autre chemin d'attaque. Ce travail montre que l'adversaire pourrait atteindre l'objectif d'attaque plus rapidement en empruntant cette voie. Cette approche per-

met d'anticiper les chemins d'attaque qui n'étaient pas connus lors de la génération du graphe d'attaque proactif.

Le deuxième objectif de recherche se concentre sur la sélection automatique des contre-mesures de cybersécurité grâce à la correspondance de graphes de connaissance. Cette contribution consiste à faire correspondre le graphe de connaissances de l'ontologie des vulnérabilités avec un graphe de connaissances de contre-mesures pour en déduire des contre-mesures potentielles contre les vulnérabilités du système. Cette approche est évaluée à l'aide de la métrique F1 Score permettant d'évaluer si les contre-mesures sélectionnées sont correctes.

Le troisième objectif de recherche consiste à générer automatiquement des playbooks optimaux de réponse aux incidents. Cette contribution se concentre sur la génération d'un playbook optimal pour chaque vulnérabilité identifiée dans le système, en se basant sur les contre-mesures sélectionnées dans la deuxième contribution. En faisant correspondre les contre-mesures sélectionnées avec un framework de réponse aux incidents, cette solution obtient automatiquement les actions de réponse aux incidents candidates pour la génération du playbook. L'outil élague les actions de réponse aux incidents en fonction des exigences d'exploitation des vulnérabilités, des outils de sécurité du système et des contraintes de l'organisation.

Ensuite, toutes les combinaisons d'actions sont générées en tenant compte de contraintes telles que le nombre minimum d'actions dans un playbook. Un algorithme d'optimisation aide à sélectionner le playbook optimal en effectuant un compromis entre trois objectifs d'optimisation définis. Cette contribution est validée pour un système illustratif, démontrant comment le playbook optimal généré est logiquement efficace. La performance temporelle du processus est évaluée pour le playbook optimal généré pour 40 vulnérabilités. L'efficacité de l'algorithme d'optimisation est également évaluée à l'aide d'une métrique de l'écart en pourcentage entre le nombre de playbooks générés avant et après l'application de l'algorithme d'optimisation sur les playbooks générés pour les 40 vulnérabilités.

Le quatrième objectif de recherche consiste à générer un graphe attaque-défense en temps réel. La contribution se concentre sur l'instanciation de l'action de réponse aux incidents d'un playbook généré sur le graphe d'attaque lorsqu'une alerte générée correspond à un nœud du graphe d'attaque. La solution est déployée sur un poste de travail dans une infrastructure industrielle virtuelle. La configuration permet à un routeur d'envoyer du trafic entre le réseau cible et le réseau dans lequel se trouve l'adversaire vers le poste de travail où l'outil est installé.

L'outil de surveillance intégré à la solution proposée surveille le trafic transitant par le routeur et le trafic venant des différentes interfaces de réseaux. Puis, il est capable de générer des alertes. Lorsqu'une alerte générée correspond à un nœud de graphe d'attaque, l'outil com-

mence à rechercher des contre-mesures qui peuvent être instanciées sur le graphe d'attaque dans un tableau qui met en corrélation les actions d'attaque avec les actions de réponse aux incidents. L'approche est validée pour trois scénarios d'utilisation, en considérant la pertinence des contre-mesures instanciées sur le graphe d'attaque en termes de sécurité, c'est-à-dire leur efficacité et leur applicabilité dans la réalité et la performance temporelle du processus d'instanciation.

Cette thèse répond à plusieurs problématiques de recherche. Cependant, elle présente certaines limitations. Le processus automatisé de correspondance de graphes permet la sélection de contre-mesures pertinentes, mais prend du temps. Il ne peut donc pas être lancé en temps réel. La complexité temporelle du processus de génération du playbook est non-polynomiale. En fonction de la taille et de la complexité du système ainsi que du nombre de vulnérabilités qu'il contient, l'instanciation d'action du playbook optimal sur le graphe d'attaque peut prendre plus de trois minutes. Cependant, en général, un adversaire met plus de trois minutes pour exécuter la prochaine action menant vers l'objectif de l'attaque. L'approche proposée est donc optimale. Elle automatise en temps réel non seulement l'instanciation d'actions de remédiation sur le graphe d'attaque, mais aussi, elle remonte les actions ne pouvant être instanciées sur le graphe d'attaque aux experts en cybersécurité.

ABSTRACT

Cyberattacks have increased since the COVID-19 pandemic because of the increasing use of the internet and home office trends. They can have several origins. This thesis focuses on cyberattacks exploiting vulnerabilities in computer networks. Organizations should spend time deciding which vulnerabilities should be addressed as a priority. The amount of alerts received by organizations is huge. Cybersecurity experts can only deal with some of them. They should also decide which incident response action should be part of the organization's incident response plan.

This thesis aims to automate the incident response plan by building it based on the discovered vulnerabilities' exploitation requirements, system composition, and organization constraints. This approach is based on attack graphs, which help represent the paths an adversary can follow to cause damage to the system.

The main objective of this thesis is to generate an automated incident response playbook in real-time to respond to the cyberattack. Some essential research activities are defined to reach this goal. They are subdivided into four research objectives that constitute a basis for the proposed contributions of this thesis.

The first research objective is updating real-time attack graphs based on a vulnerability ontology and system monitoring. This contribution proposes a tool composed of a module responsible for correlating alerts generated by a monitoring tool integrated into this tool with logical attack graphs. This module deduces which vulnerability is susceptible to be exploited or is exploited and then queries the ontology to find possible new attack paths leading to the attacker's goal. This launches then the attack graph enrichment with new paths.

This contribution is validated thanks to a use-case scenario concerning a smart city. The attack's goal is to cause physical damage to public transportation. This goal is reachable because an attacker gains administrator privileges, allowing the modification of sensitive information by exploiting the BlueKeep vulnerability. The vulnerability exploitation activates the ontology deduction of a new impact of its exploitation, leading to new attack paths deduction. This work shows that the adversary could reach the attack goal faster by taking this path. This approach helps anticipate attack paths that were not known when generating the proactive attack graph.

The second research objective focuses on selecting cybersecurity countermeasures automatically based on graph matching. This contribution consists of matching the knowledge graph

of the vulnerability ontology with a countermeasure knowledge graph to deduce potential countermeasures against the system's vulnerabilities. This approach is evaluated using the F1 Score metric to assess the countermeasures' correctness.

The third research objective is to generate optimal incident response playbooks automatically. This contribution focuses on generating an optimal playbook for each vulnerability identified in the system based on countermeasures selected from the second contribution. The proposed solution automatically generates the candidate incident response actions for the playbook by matching the selected countermeasures with an incident response framework. The tool prunes the incident response actions based on the vulnerability exploitation requirements, the system's security tools, and the organization's constraints.

Therefore, all the combinations of actions are generated considering constraints such as the minimum number of actions in a playbook. An optimization algorithm helps to select the optimal playbook by doing a tradeoff between three defined optimization objectives. This contribution is validated for an illustrative system, demonstrating how the optimal generated playbook is logically effective. The time performance of the process is evaluated for the optimal playbook generated for 40 vulnerabilities. The effectiveness of the optimization algorithm is also evaluated by using a metric of the percentage gap between the number of playbooks generated before and after applying the optimization algorithm over the playbooks generated for the 40 vulnerabilities.

The fourth research objective is to generate an attack-defense graph in real-time. The contribution focuses on instantiating the incident response actions of a playbook generated on the attack graph when an alert generated matches at least a node of the attack graph. The solution is deployed on a workstation in a virtual industrial infrastructure. The configuration enables a router to send traffic between the target network and the network where the adversary network is to the workstation.

The monitoring tool integrated into the proposed solution monitors the traffic passing through the router and traffic coming from different network interfaces. Then, it is able to generate alerts. When a generated alert matches an attack graph node, the tool looks for countermeasures that can be instantiated on the attack graph in a table correlating attack facts with incident response actions. The approach is validated for two use case scenarios, considering the security relevance of the countermeasures instantiated on the attack graph and the time performance of the instantiation process.

This thesis responds to several research problems. However, it has some limitations. The automated graph-matching process enables the selection of relevant countermeasures but is time-consuming. Therefore, it can not be launched in real time. The time complexity of

the playbook generation process is non-polynomial. Depending on the system's size and complexity and the number of vulnerabilities, the instantiation of incident response actions from the optimal playbook on the attack graph can take more than three minutes. However, an adversary generally takes over three minutes to take his/her next step toward reaching the attack goal. The proposed approach is, therefore, optimal. It automates the instantiation of remediation actions on the attack graph in real-time and reports actions that cannot be instantiated on the attack graph to cybersecurity experts.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENTS iv

RÉSUMÉ v

ABSTRACT viii

TABLE OF CONTENTS xi

LIST OF TABLES xv

LIST OF FIGURES xvi

LIST OF SYMBOLS AND ACRONYMS xix

CHAPTER 1 INTRODUCTION 1
- 1.1 Context and Problem Definition 1
- 1.2 Research Objectives 4
- 1.3 Contributions 4
- 1.4 Publications 5
- 1.5 Thesis Organization 6

CHAPTER 2 LITERATURE REVIEW 7
- 2.1 Background 7
  - 2.1.1 Attack Graph (AG) 7
  - 2.1.2 AND-OR Graph 7
  - 2.1.3 Predicate 8
  - 2.1.4 Logical Attack Graph (LAG) 8
  - 2.1.5 Logical Reasoner 8
  - 2.1.6 Attack-Defense Graph (ADG) 9
  - 2.1.7 Correlation 9
  - 2.1.8 Anti-correlation 9
  - 2.1.9 Incident Response (IR) Playbook 9
  - 2.1.10 Pareto Optimality 10
  - 2.1.11 Ontologies 10

2.1.12 Knowledge Graph (KG) 11
2.1.13 SPARQL Protocol and RDF Query Language (SPARQL) 11
2.1.14 Simplified Agile Methodology for Ontology Development(SAMOD) . 11
2.1.15 Abstractions Translation Ontology Method(ATOM) 12
2.1.16 b-matching of a Graph 12
2.1.17 Graph Matching 13
2.1.18 Natural Language Processing (NLP) 13
2.1.19 Word2Vec 13
2.1.20 RDF2Vec 13
2.1.21 Cosine Similarity 13
2.1.22 Precision 14
2.1.23 Recall 14
2.1.24 F1 Score 14
2.1.25 BlueKeep 14
2.1.26 EternalBlue 14
2.1.27 WannaCry 15
2.1.28 Man-in-the-middle (MITM) attack 15
2.2 Attack Graph Generation 15
2.2.1 Planning-based Attack Graph 16
2.2.2 Logical-based Attak Graph 16
2.2.3 Ontology-based Attack Graph 17
2.3 Countermeasures Selection 18
2.3.1 Non-AG based countermeasures Selection Approaches 18
2.3.2 AG based Countermeasures Selection Approaches 18
2.4 Incident Response Playbooks Generation 20
2.5 Graph Matching in Cybersecurity 22
2.6 Conclusion 24

CHAPTER 3 AUTOMATED ATTACK GRAPH ENRICHMENT 25
3.1 Overview 25
3.2 Generation of the AG 26
3.3 Matching Vulnerability Information with Monitoring Information 26
3.4 Vulnerabilities and Ontologies 27
3.5 Enrichment of AGs 27
3.6 Experimental Approach 28
3.6.1 Use Case Scenario 28

3.6.2 Setup . . . 30
3.6.3 Results . . . 34
3.7 Discussion and Conclusion . . . 36

CHAPTER 4 AUTOMATED COUNTERMEASURES SELECTION . . . 38
4.1 Overview . . . 38
4.2 Experimental Dataset . . . 41
4.3 Graph Matching . . . 41
4.4 Graph Matching Models Validation . . . 42
4.5 Countermeasure Selection . . . 44
4.6 Implementation Results . . . 45
4.6.1 VDO Population . . . 45
4.6.2 Countermeasures Selection Evaluation . . . 45
4.6.3 Illustrative Use Case . . . 48
4.7 Discussion and Conclusion . . . 50

CHAPTER 5 OPTIMAL AUTOMATED PLAYBOOKS GENERATION . . . 51
5.1 Overview . . . 51
5.2 IR Ontology . . . 53
5.3 Optimal Playbooks Generation . . . 55
5.4 Results and Evaluation . . . 58
5.4.1 Illustrative Example . . . 58
5.4.2 Evaluation . . . 66
5.4.3 Other Optimal Playbooks . . . 67
5.5 Discussion and Conclusion . . . 70

CHAPTER 6 ATTACK-DEFENSE GRAPH GENERATION . . . 72
6.1 Overview . . . 73
6.2 Countermeasure Predicates Generation . . . 73
6.3 Attack and Countermeasure Predicates Matching . . . 74
6.4 Countermeasures Instantiation on AG . . . 75
6.5 Complexity Analysis . . . 79
6.6 The ADG generator Implementation . . . 81
6.7 Experimental Setup . . . 83
6.7.1 Architecture 1 . . . 83
6.7.2 Architecture 2 . . . 85
6.8 Use case Scenario . . . 86

6.8.1 Scenario 1 86
6.8.2 Scenario 2 87
6.8.3 Scenario 3 87
6.9 Experimental Results 88
6.9.1 Countermeasures Selection 88
6.9.2 Optimal Playbooks 89
6.9.3 Scenario 1 90
6.9.4 Scenario 2 94
6.9.5 Scenario 3 101
6.10 Relevance Evaluation 110
6.10.1 Scenario 1 110
6.10.2 Scenario 2 111
6.10.3 Scenario 3 113
6.11 Time Performance Evaluation 116
6.12 Discussion and Conclusion 116

CHAPTER 7 CONCLUSION 119
7.1 Summary of Works 119
7.2 Limitations 120
7.3 Future Research 121
7.4 Conclusion 121

REFERENCES 123

LIST OF TABLES

Table 3.1 Comparison between various CVE concerning the same port and protocol based on their CVSS index 27
Table 3.2 Classification of CVE-2002-0392 characteristics 28
Table 3.3 List of impacts inferred. 35
Table 4.1 Word2Vec model prediction evaluation 43
Table 4.2 Evaluation of the prediction for the methods 47
Table 4.3 Evaluation of the prediction for the impacts 48
Table 5.1 List of countermeasures for CVE-2021-21277 from graph matching . . 60
Table 5.2 List of actions with their parameters' values for CVE-2021-21277 . . 63
Table 5.3 Evaluation of this approach 66
Table 5.4 The countermeasures related to the artifact executable file for CVE-2017-0144 68
Table 6.1 The core POS categories tags 74
Table 6.2 An extract of the predicates table generated by correlating AG predicates with countermeasure predicates 77
Table 6.3 The 11 matched AG predicates with the IR actions from the CVE-2021-26828 playbook 92
Table 6.4 Flow value calculation for countermeasures instantiation on the AG for Scenario 2 96
Table 6.5 The 18 matched AG predicates with the IR actions predicates 97
Table 6.6 8 matched AG predicates with the IR actions predicates 100
Table 6.7 The 25 matched AG predicates with the IR actions predicates 104
Table 6.8 Flow value calculation for countermeasures instantiation on the AG for Scenario 3 105
Table 6.9 Time performance for instantiating IR actions on real-time on an ADG 116

LIST OF FIGURES

Figure 1.1 This thesis subdivision 5

Figure 3.1 **Cyber-physical attack scenario**: An attacker exploits the vulnerability linked to CVE-2019-0708 on a Starting Device. Subsequently, the adversary retrieves administrator credentials from the device's memory and utilizes them to gain control over a critical asset. This attack impacts the system's physical and digital components, including individuals and services. 30

Figure 3.2 The AG Enrichment Process 31

Figure 3.3 Sample classes associated to VDO. 33

Figure 3.4 Sample outcomes: (a) AG generated for the scenario involving violence on buses. (b) The AG enriched with inferred ontology data. 35

Figure 4.1 The ADG generation process 39

Figure 4.2 A flow chart of the ADG generation process 40

Figure 4.3 VDO and D3FEND sub-graphs 42

Figure 4.4 Architecture of the proposed solution 42

Figure 4.5 A Maritime Transportation System use case 48

Figure 4.6 An AG generated for the use case scenario 49

Figure 5.1 The ADG generation process 52

Figure 5.2 The optimal playbook generation process 53

Figure 5.3 The incident response playbook ontology structure. The classes are represented in square form, and the circles represent the instances. The directed arrows represent the relationships between a class and another class, and an undirected arrow represents the relationship between a class and itself. The dashed arrows represent the property taken from RDF schema. 54

Figure 5.4 Pareto optimality for playbook generation 57

Figure 5.5 Example System 59

Figure 5.6 List of candidate actions extracted from RE&CT framework for CVE-2021-21277 61

Figure 5.7 Optimal playbook selected when each phase can be repeated only once 64

Figure 5.8 Optimal playbook selected when a category is repeated once for a phase 64

Figure 5.9 Optimal playbook generated for the exploitation of EternalBlue in the use-case scenario introduced in Section 4.6.3 69

Figure 5.10 Default playbook generated for BlueKeep 69
Figure 6.1 The ADG generation process 73
Figure 6.2 The analogy approach to match AG predicates with countermeasure predicates 75
Figure 6.3 An extract of a multi-stage Attack Graph 78
Figure 6.4 Countermeasures applied on some nodes 78
Figure 6.5 The ADG generator architecture 81
Figure 6.6 A virtual industrial architecture composed of a Computer running FactoryIO, a SimAgent VM that pulls sensor values from FactoryIO and pushes docker slaves' output values, a SCADA that starts and stops the entire process, a Master PLC that forwards the start/stop order to other PLCs, a PLC that controls the slave devices, and a crane that puts elements on a conveyor belt. 83
Figure 6.7 A virtual industrial architecture with a computer running FactoryIO, a SimAgent VM for sensor and output management, an operator workstation, SCADA for process control, a master PLC coordinating other PLCs, a slave PLC managing devices, and a crane handling elements on conveyors. 86
Figure 6.8 Optimal IR playbook for CVE-2021-26828 89
Figure 6.9 Optimal IR playbook for CVE-2023-48795 90
Figure 6.10 The AG generated for the first use-case scenario 91
Figure 6.11 The ADG generated for for the packets sent and the brute force attack launched. The orange nodes represent the countermeasures in response to the brute force attack instantiated on the AG generated previously in Figure 6.10 94
Figure 6.12 The AG generated for the second use-case scenario 95
Figure 6.13 The ADG generated for the packets sent and the brute force attack launched. 96
Figure 6.14 The ADG generated when the adversary connects through SSH to the SCADA using root credentials. 99
Figure 6.15 The ADG generated by instantiating IR actions of the default playbooks for the packets sent and the brute force attack launched. 101
Figure 6.16 The ADG generated when the adversary connects through SSH to the SCADA using root credentials. 102
Figure 6.17 The AG generated for the third use-case scenario 103
Figure 6.18 ADG generated when the Bluekeep vulnerability is exploited 106

Figure 6.19 The ADG generated for the scanning of port 8080 on the SCADA . . 107
Figure 6.20 The ADG generated for the scanning of port 22 on the openplc . . . 108
Figure 6.21 The ADG generated when the MITM attack between openplc and conv-prod-in is successful 109
Figure 6.22 The ADG generated when CVE-2021-26828 is exploited successfully on the SCADA 110
Figure 6.23 The ADG generated after the adversary reach the final goal of the attack scenario 111

LIST OF SYMBOLS AND ACRONYMS

ADG	Attack-Defense Graph
ADTrees	Attack-Defense trees
AG	Attack Graph
ATOM	Abstractions Translation Ontology Method
CACAO	Collaborative Automated Course of Action Operations
CAPEC	Common Attack Pattern Enumeration and Classification
CSAF	Common Security Advisory Framework
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
IDS	Intrusion Detection System
IoT	Internet of Things
IR	Incident Response
FN	False Negative
FP	False Positive
KG	Knowledge Graph
LAG	Logical Attack Graph
LAMBDA	Language to Model a database for Detection of Attacks
MTS	Maritime Transportation System
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
OWL	Web Ontology Language
RCE	Remote Code Execution
RDF	Resource Description Framework
SAMOD	Simplified Agile Methodology for Ontology Development
SIEM	Security Information and Event Management
SPARQL	SPARQL Protocol and RDF Query Language
SOAR	Security Orchestration, Automation and Response
STIX	Structured Threat Information Expression
UCO	Unified Cybersecurity Ontology
VDO	Vulnerability Description Ontology
TP	True Positive

CHAPTER 1 INTRODUCTION

The Thesis subject is the automatic generation of attack and remediation graphs. Several attack graph and remediation graph generation approaches exist in the literature. Some of them base the remediation graph generation approach on an attack graph. This work focuses on updating the generated real-time attack graphs representing the attacker's action to cause damage to a system. The update of the attack graph consists of adding new attack paths based on ontology inference. Furthermore, this work proposes to instantiate nodes representing incident response action on the attack graph to get a remediation graph called an attack-defense graph. The term attack-defense graph is used in the rest of this work to refer to the remediation graph as an extension of the attack graph. Section 1.1 explains the identified problem and gives the context of this research subject. Section 1.2 presents the research objectives. Section 1.3 gives an overview of the contributions with respect to these research objectives. Section 1.4 presents the publications. Finally, Section 1.5 provides an overview of this thesis organization.

1.1 Context and Problem Definition

In the last 4 years, cyberattacks have increased because of the rapid use of the Internet globally since the COVID-19 pandemic [1]. People are increasingly working from home and using their personal computers for professional tasks and work computers for personal activities. The most trending cyberattacks are Denial of Service (DoS), Distributed Denial of Service (DDoS), Man-in-the-Middle (MITM), and Ransomware attacks [1–3]. These cyberattacks can have many causes, such as transferring daily life into cyberspace, system errors, and emerging technologies [3]. This work focuses on the causes of system error, particularly attacks caused by vulnerabilities in computer networks.

Organizations can discover several vulnerabilities in their system. However, they can only patch some of them at a time. Patching a vulnerability can impact the organization's business processes. Therefore, cybersecurity experts should conduct a security assessment that considers the impacts of vulnerabilities and the impacts of the mitigation action on the system to decide which actions should constitute the incident response plan and what the perfect timing is for each action release.

Organizations must equip themselves with systems that detect and react to cyberattacks and investigate the following attacks. Cybersecurity operators face difficulties due to the

volume of alerts they must process, their complexity due to technological developments, and the sophistication of attacks. They also face difficulty in analyzing the risks to select countermeasures to create incident response plan.

There are several types of approach for performing countermeasure selection. An approach is based on a risk analysis process; it consists of analyzing each vulnerability one by one using, for example, the Common Vulnerability Scoring System (CVSS). This approach does not consider the prerequisites for exploiting a vulnerability, the topology of the system, or the position of the adversary and its potential movements. The other approach uses attack graphs or attack trees. Both help represent an adversary's paths to compromise a system. They differ in structure; an attack graph is more scalable than an attack tree. Integrating attack graphs to help analyze the risk, considering specific resources and attackers, makes the risk analysis process more precise. The countermeasure selection approach based on attack graphs then makes it possible to select the best countermeasures.

This thesis aims to automate the construction of a incident response plan process based on the vulnerabilities discovered, exploitation requirements, organization constraints, and architecture composition to reduce the decision-making time of cybersecurity experts. The proposed approach is based on attack graphs.

There are different types of attack graphs. Some researchers favor topological attack graphs [4] over logical attack graphs [5] because they provide more information about network elements. Logic-based models represent an attack as a logical predicate that requires successful preconditions for the attack to be possible. This model type accurately represents how cybersecurity operators estimate whether an attack is possible. Topological models provide a higher-level view of possible attacks in an information system, representing an attack as a means of access from one machine to another. However, topological models do not explain how attacks are carried out in detail and, therefore, are less precise [6]. A logical approach will be favored in this thesis.

In order to update logical attack graphs in real time, it is necessary to consider information about exploited vulnerabilities. This information can be saved in vulnerability databases (VDB) such as the National Vulnerability Database (NVD). The Common Vulnerabilities and Exposures (CVE) List [7] is a database of meta vulnerabilities maintained by MITRE [1]. It provides a standard identifier, CVE ID, and vulnerability overview for known weaknesses in various VDBs. The vulnerability information is text written in natural language. This information must be transformed into machine-readable text that an attack graph generator can use. It is, therefore, essential to automate the extraction of vulnerability information

[1] `https://cve.mitre.org/`

to facilitate this process. Hence, artificial intelligence techniques such as Natural Language Processing (NLP) need to be used.

The use of an ontology is necessary to represent the extracted entities and thus ensure their homogeneity. An ontology allows one to represent a domain in a structured way. The ontology facilitates interoperability between information from the attack graph and what is extracted from vulnerability databases. It is possible to make queries on the ontology in order to infer new knowledge necessary for the ontological enrichment of the attack graph, which is the subject of Chapter 3, and for the selection countermeasures, discussed in Chapter 4. A correlation between countermeasures and vulnerabilities makes it possible to determine at which position a given countermeasure should be applied. For this, a countermeasure ontology is also necessary. The automated selection of countermeasures is possible by matching the knowledge graphs of both ontologies. All these concepts are explained in Chapter 2. In this sense, this thesis proposes an approach for the automated selection of countermeasures based on Knowledge Graph matching.

However, the selected countermeasures are not helpful alone. They are not executable without considering the system architecture or organization constraints. This thesis proposes an optimal incident response playbook generation based on the countermeasures selected. Section 2 introduces the incident response playbook concept. These incident response playbook actions are then involved in the attack-defense graph generation process.

This dissertation thus formulates the following problem statement:

> *"An attack graph can provide excellent support for visualizing and understanding an adversary's path to launch an attack on a system and selecting the proper defensive actions to block the attacker. However, a system is dynamic, so updating the attack graph based on the system's state change is essential. Integration system monitoring can lead to obtaining an enriched attack graph based on the system change. The attack graph is limited to represent the attack path. However, it is more advantageous to provide a solution that can also represent the defensive techniques and where they should be applied to block the adversary. The choice of these defensive techniques depends on the infrastructure compositions and the attack success requirement to ensure that they can be executed in real-time without damaging the infrastructure's functioning."*

1.2 Research Objectives

The main research objective is to automate an incident response plan in real-time to respond to cyberattacks. To reach this objective, several research activities, subdivided into specific objectives, are essential.

General objective Generate an automated incident response plan in real time to respond to the cyberattack based on attack and remediation graphs.

Specific objectives

- Update attack graphs in real-time based on an ontology and system monitoring.
- Select cybersecurity countermeasures automatically based on graph matching.
- Generate automatically optimal incident response playbooks.
- Generate attack-defense graph in real-time.

1.3 Contributions

Four contributions, summarized in the following paragraphs to reach the objectives specified in Section 1.2, are proposed. Figure 1.1 represents the dependencies between these contributions.

The first contribution focuses on enriching attack graphs in real-time based on received alerts from monitoring tools integrated into the proposed solution and an ontology inference. The attack graph is generated from the output of a system scan. The system's real-time monitoring enables alert generation when detecting a malicious action. The matching of the alert information with the attack graph enables queries on a vulnerability ontology, leading to the deduction of new attack paths for enriching the attack graphs.

The second contribution focuses on selecting cybersecurity countermeasures automatically by matching the knowledge graph of the vulnerability ontology of the first contribution with a countermeasure knowledge graph. These concepts are introduced in Section 2.2.

The third contribution automatically generates an optimal incident response playbook based on the countermeasures selected in the second contribution. The playbook's generation considers the organization's constraints, vulnerability exploitation requirements, and system architecture.

The fourth contribution consists of instantiating the optimal playbook's incident response actions on the attack graph in real-time when a malicious action is detected. This instantiation process generates an attack-defense graph.

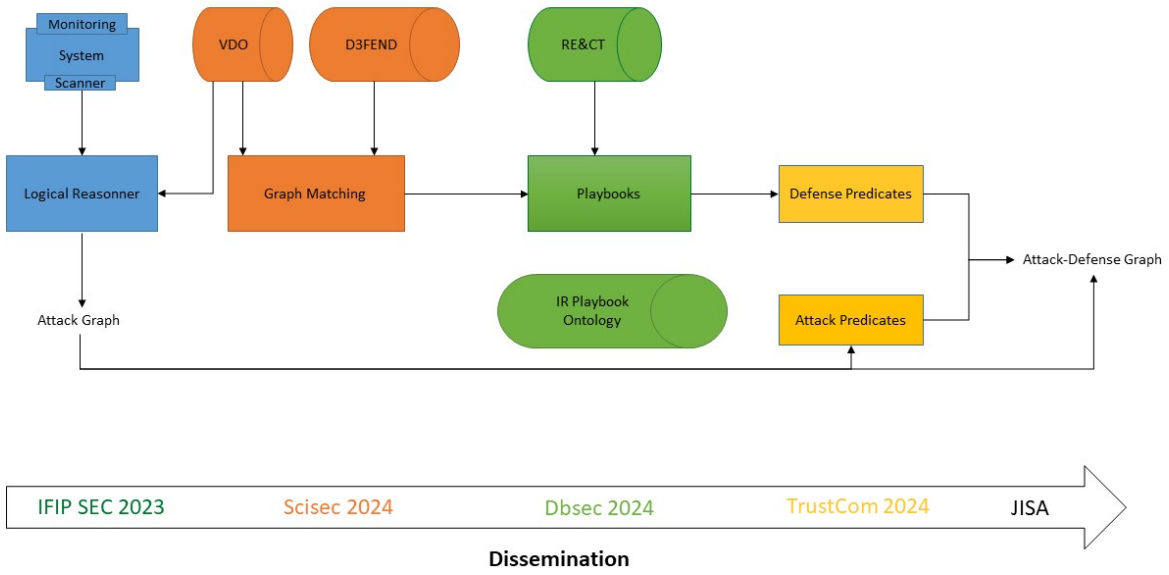


Figure 1.1 This thesis subdivision

1.4 Publications

This section presents a list of scientific publications from this research work. This thesis is a classical thesis. However, Chapter 3, Chapter 4, Chapter 5 and Chapter 6 are based on this scientific publications. Figure 1.1 shows the conferences and the journal where this work is submitted or published for each contribution presented in Section 1.3.

Refereed Conference Papers

1. **Saint-Hilaire, K.**, Cuppens, F., Cuppens, N., Garcia-Alfaro, J. (2024). Automated Enrichment of Logical Attack Graphs via Formal Ontologies. In: Meyer, N., Grocholewska-Czuryło, A. (eds) ICT Systems Security and Privacy Protection. SEC 2023. IFIP Advances in Information and Communication Technology, vol 679. Springer, Cham. https://doi.org/10.1007/978-3-031-56326-3

2. **Saint-Hilaire, K.A.**, Neal, C., Cuppens, F., Boulahia-Cuppens, N., Hadji, M. (2024). Optimal Automated Generation of Playbooks. In: Ferrara, A.L., Krishnan, R. (eds) Data and Applications Security and Privacy XXXVIII. DBSec 2024. Lecture Notes in Computer Science, vol 14901. Springer, Cham. https://doi.org/10.1007/978-3-031-65172-4_12

3. **Saint-Hilaire, K.A.**, Neal, C., Cuppens, F., Boulahia-Cuppens, N., Hadji, M. (2025). Matching Knowledge Graphs for Cybersecurity Countermeasures Selection. In: Zhao, J., Meng, W. (eds) Science of Cyber Security. SciSec 2024. Lecture Notes in Computer Science, vol 15441. Springer, Singapore. https://doi.org/10.1007/978-981-96-2417-1_7

4. **Saint-Hilaire, K. A.**, Neal, C., Cuppens, F., Cuppens-Boulahia, N., & Bassi, F. (2024). Attack-Defense Graph Generation: Instantiating Incident Response Actions on Attack Graphs. TrustCom 2024, Sanya, December 17-21, 2024.

Journal Paper

1. **Saint-Hilaire, K. A.**, Neal, C., Cuppens, F., Cuppens-Boulahia, N., Bassi, F., & Hadji, M. (2024). A Real-time Automated Attack-Defense Graph Generation Approach. Submitted to JISA.

1.5 Thesis Organization

The organization of this thesis follows. Section 2 introduces the essential concepts to understand the proposed contributions of this thesis. Section 3 presents the proposed ontology-based attack graphs enrichment approach. Chapter 4 describes the approach to automatically select cybersecurity countermeasures based on graph matching. Chapter 5 introduces the optimal automated incident response playbook ontologies. Chapter 6 presents the attack-defense graph generation approach. Finally, Chapter 7 summarizes this work's contributions, provides an overview of their limitations, and presents the direction toward future research subjects.

CHAPTER 2 LITERATURE REVIEW

This chapter defines the relevant concepts needed to understand the rest of this work and presents a review of related works to position these contributions in relation to previous works.

This chapter is organized as follows. Section 2.1 defines the essential concepts for understanding this work. Section 2.2 reviews the previous attack graph generation approaches. Section 2.3 compares countermeasures selection contributions with the proposed contribution in this work. Section 2.4 compares the Incident Response (IR) playbook generation with existing ones in the literature. Section 2.5 exposes the current landscape on graph matching in cybersecurity.

2.1 Background

This section introduces the essential concepts to understand this work. This section defines concepts related to attack graphs like AND-OR graphs, predicates, and Logical attack graphs. Then, it defines the concepts necessary to understand the attack defense graph generation, such as correlation and anti-correlation. The attack defense graph generation is based on IR actions instantiated on an attack graph. This action is part of a generated IR playbook. Thus, it explains concepts involved in the IR playbook generation, such as Pareto optimality and ontology. The IR playbook generation is based on countermeasures selected from graph matching. This section then introduces concepts related to the graph-matching process for countermeasures selection, such as b-matching, Word2Vec, RDF2Vec, and cosine similarity.

2.1.1 Attack Graph (AG)

An AG graphically represents the steps an adversary should take to cause damage to a system. An AG represents the event flow in a top-down way. The AG vertices represent the preconditions of an exploit and the exploit itself.

2.1.2 AND-OR Graph

An AND-OR graph is a directed graph in which each vertex v is either an OR or an AND. A vertex represents a sub-objective; according to its type (AND or OR), it requires either the conjunction or disjunction of its children to be achieved. A root vertex v of an AND-OR graph is a precondition as it does not require any other vertex v to be satisfied.

2.1.3 Predicate

A predicate is an atomic formula in the form $f(t_1, ..., t_k)$, where each t_i can be a constant (starting with a lowercase letter), a variable (starting with an uppercase letter), or a wild card ("_"). An example of a predicate follows:

$$netAccess(H, Protocol, Port)$$

Here, netAccess means a user can send packets to port $Port$ on a machine H through a protocol $Protocol$.

2.1.4 Logical Attack Graph (LAG)

A LAG is a type of an AND-OR graph. The vertices represent logical facts describing adversaries' actions or the pre-conditions to carry them out. The edges correspond to the dependency relations between the vertices.

A LAG has two kinds of nodes: fact and derivation nodes.

- A fact node is labeled with a logical statement as a predicate applied to its arguments. A predicate is an atomic formula in the form $f(t_1, ..., t_k)$, where each t_i can be a constant (starting with a lowercase letter), a variable (starting with an uppercase letter), or a wild card ("_").

  There are two kinds of fact nodes:

  - a primitive fact node, representing a pre-condition that does not depend on any node, and a derived fact node.
  - A derived fact node is an OR node depending on one or more derivation nodes. The root node is the attack's final objective.

- A derivation node is an AND node describing the achievement's requirement of all the facts of its children.

2.1.5 Logical Reasoner

A logical reasoner eases the generation of a LAG by processing input information. It uses formal rules and logical principles to infer malicious actions based on knowledge of the system's configuration and identified vulnerabilities.

2.1.6 Attack-Defense Graph (ADG)

An ADG is a graphical representation describing an attacker's measures to attack a system and the defenses a defender can employ to protect the system. The attack nodes represent potential actions an attacker might use to compromise the system. The nodes representing defenses are directed toward the adversarial actions or these actions' preconditions. The edges represent the relationships between the offensive steps and defensive measures.

2.1.7 Correlation

Supposing two actions, a and b.

- pre_a represents the system state required for the success of the action a. $post_a$ represents the system state after the action's success a.
- pre_b represents the system state required for the success of the action b. $post_b$ represents the system state after the action's success b.

Attacks a and b are correlated if they have at least one predicate in common in $post_a$ and pre_b. This concept can be applied either to the final objective of the attack or to the sub-objective.

2.1.8 Anti-correlation

Considering a countermeasure a and an attacker's action b:

- $post_a$ is the set of literals of post-condition of a.
- pre_b is the set of literals of pre-condition of b.

A countermeasure a for an attacker's action b is a model of which the post-condition contains a predicate that contradicts one predicate of pre_b.

2.1.9 Incident Response (IR) Playbook

A playbook refers to a documented set of procedures and guidelines specifying how to carry out specific tasks in a given domain. An IR playbook consists of the steps and procedures an organization should follow when addressing and mitigating occurred incidents, allowing the organization to respond to specific requirements of the security policies [8].

2.1.10 Pareto Optimality

Providing a multiobjective optimization background explanation is necessary to understand the following concepts better.

A multiobjective optimization problem aims to find the optimum of a function: $f\colon X \to F$ as $f\colon (x_1, ..., x_n) \to (f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n))$

Where:

- X is a n-dimensional decision space, and F is the m-dimensional objective space with $m \geq 2$.
- $\forall x \in X$, $f(x) \in F$ is an objective vector.

Assuming that $F \subseteq \mathbb{R}^m$ and all objectives $f_i\colon X \to \mathbb{R}$ should be minimized for all $i \in 1, ..., m$. Based on this assumption, the following concepts can now be defined.

Pareto Dominance An objective vector f^A dominates another objective vector f^B, i.e $f^A \prec f^B$, if $f_i^A \leq f_i^B \forall i \in 1, ..., m$ and $f^A \neq f^B$

Incomparability of Vectors The objective vectors f^A and f^B are incomparable, i.e., $f^A \parallel f^B$, if $f^A \neq f^B$, $f^A \nprec f^B$ and $f^B \nprec f^A$

Pareto Optimality The objective vector f^A is Pareto optimal if there is no f such as $f \prec f^A$.

Pareto Front If the objective vectors f^A and f^B are incomparable and there is no f such as $f \prec f^A$ and no f such as $f \prec f^B$, they are both Pareto optimal. They constitute the Pareto front.

2.1.11 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 [9]. 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 [10]. The following paragraph explains the concept of KG.

2.1.12 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. The TBox in the DL language is represented below. However, a knowledge graph must be created to represent a particular countermeasure, such as **Update software**. The asset mitigated by this countermeasure is Software. Nodes represent entities of interest, such as Software, and edges of potentially different relationships between nodes, such as *impacts* and *mitigates*. The constructed KG makes it possible to represent the following reality: *A countermeasure* ***Update software*** *mitigates an asset that is a software. An adversarial action* ***Run virtual instance*** impacts this Software. 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.

Terminologies domain in DL language

$$\boxed{\begin{array}{l} Countermeasure \cap AdversarialAction \equiv \bot \\ Countermeasure \equiv mitigates.Asset \end{array}}$$

2.1.13 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.

2.1.14 Simplified Agile Methodology for Ontology Development(SAMOD)

SAMOD [11] is an iterative methodology, and therefore several iterations are often necessary to obtain the final result. However, the first step is not included in the iterations. It is first about collecting as much information as possible on the domain for which we wish to build our ontology. This is a common point across all methods, but it is a necessary step because we must understand the domain and its concepts if we wish to build an ontology. The following steps are then iterative.

Here are the different steps of this method:

1. Enumerate a list of questions in natural language. These questions should be answerable by the ontology.
2. List the concepts present in the questions.
3. Create a glossary to define each of the concepts.
4. Translate natural language questions into SPARQL queries.
5. Insert concepts into the ontology.

The result of an iteration is called a *modelet*. Each *modelet* generated by an iteration must be merged with the rest of the ontology. The iteration continues until all the competency questions are answered.

2.1.15 Abstractions Translation Ontology Method(ATOM)

Part of ATOM's methodology [12] is similar to SAMOD.

Here are the steps of this method:

1. Define ontology queries in natural language.
2. Translate queries into SPARQL.
3. Specify the raw information. This means using a domain expert's skills to determine which concept can answer the question.
4. Break down the question into intermediate queries. Logically, the previous step generates many concepts that can be broken down into several others.
5. Write the translation rules. This is where the reasoning ability should be analyzed and developed using SWRL rules if necessary.

2.1.16 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 [13].

2.1.17 Graph Matching

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

2.1.18 Natural Language Processing (NLP)

NLP uses machine learning to enable computers to understand and communicate with human language by combining computational linguistics with statistical modeling.

2.1.19 Word2Vec

Word2Vec [14] is a popular word embeddings model used to address the limitations of the bag-of-words model [15], which is a type of vector space model that simplifies text data representation in Natural Language Processing (NLP) and Information Retrieval. 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.

2.1.20 RDF2Vec

RDF2Vec, created by Ristoski et al. [16], 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. Each entity in the KG is represented by a vector of latent numerical features. The similarity of the vectors is calculated using a distance metric to match their embeddings.

2.1.21 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; Their cosine similarity can be calculated using Formula 2.1.

$$\cos(AB) = \frac{A.B}{||A||||B||} \tag{2.1}$$

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.

2.1.22 Precision

Precision measures the number of positive predictions that are correctly predicted. It is calculated by dividing the number of true positive predictions (TP) by all positive predictions, i.e., True Positive (TP) + False Positive (FP).

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

2.1.23 Recall

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

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

2.1.24 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's performance. It is calculated as follows:

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

2.1.25 BlueKeep

BlueKeep is a security vulnerability discovered in the implementation of the Microsoft remote desktop protocol, which allows remote code execution.

2.1.26 EternalBlue

EternalBlue is an exploit that uses a security flaw that exists in the first version of the Server Message Block (SMB) protocol.

2.1.27 WannaCry

WannaCry is a ransomware cryptoworm that targets computers of the Windows operating system by encrypting data and asking for ransom payments.

2.1.28 Man-in-the-middle (MITM) attack

In a MITM attack, an attacker put himself/herself in the middle of the communication between two parties to steal personal information, such as log-in credentials, account details, and credit card numbers.

2.2 Attack Graph Generation

The literature includes several approaches to the generation of AG. These approaches mainly focus on one of three types of AG: topological, bayesian, or logical.

A topological AG is a directed graph $TAG(TN, AS)$ where :

- $TN = \left\{TN\right\}$ is a set of topological nodes: the system assets,
- AS is a set of attack steps; the edges represent an attack allowing an adversary to go from a parent topological node to a child topological node.
  - Each attack step has an attack type describing how the attacker can move between nodes.
  - Based on the attack type, each attack step is linked to a set of conditions.
  - Some attack steps are related to a sensor that can launch an alert when the attack is detected.

Probabilistic models generally use Bayesian networks. In these models, the nodes represent random variables and the arcs of the probabilistic dependencies between these variables [17]. An example of these models is the Bayesian attack model [5]. In Bayesian attack graphs, nodes represent conditions, transitions, and sensors. A Bayesian condition node represents the random variable that describes whether the condition is met. The edges indicate that the child node conditionally depends on the state of its parents.

The LAGs defined in section 2.1 are based on the previously defined AND-OR graphs. LAGs provide advantages that Bayesian attack graphs do not offer [17]. The inference is very complex in a Bayesian attack graph for multistage attacks. It is difficult to track the causes

of the adversary's actions at any given moment. This can, therefore, cause performance problems. The inference is more straightforward in LAGs. The arcs clearly express the causality of the adversary's actions, which are represented by the nodes. LAGs are also more elaborate than topological attack graphs because they provide more details regarding the progress and outcome of the attack. Hence, the choice to favor LAGs in this thesis.

Different techniques can generate LAG. Planning-based, logical-based, and ontological AGs exist. The following paragraphs review works based on these three types of AG.

2.2.1 Planning-based Attack Graph

Gosh and Gosh [18] propose a planning-based approach for minimal AG generation. The planner combines input information, vulnerabilities, and initial network configuration to generate acyclic paths to produce a minimal attack graph. This approach generates an attack graph in polynomial time, regardless of the distribution of vulnerabilities on the attack graph. The initial network configuration and the description of vulnerability exploitation serve as inputs for generating a minimal attack graph through a planner.

Shirazi et al. [19] also present the AG generation problem as a planning problem. However, they only consider the information coming from the vulnerabilities in the first generation of the AG. Information from bugs serves to update the AG. However, this approach does not consider the system's current state, which prevents real-time detection of actions that could result in an attack. There may be a difference in the execution order of actions in the initial and regenerated plans. However, the two plans are similar. It is, therefore, essential that developers compare the two generated graphs before making any decision.

2.2.2 Logical-based Attak Graph

Roschke et al. [20] propose an approach to generate logical AGs based on logic programming. The input information for generating AGs is the system and vulnerability information. They integrate an IDS in the AG generation process, complementing it with data fusion and correlation to improve the quality of the alerts and their correlation. This correlation helps prioritize and label alerts. Approaches based on logic programming offer greater flexibility in terms of semantic correlation with other knowledge bases.

Stan et al. [21] also introduce an AG generation approach based on logic programming. The authors favor logic programming due to its extensibility. This scalability is the capacity of the AG to adapt to a change of magnitude. This type of graph facilitates the representation of attacks of several stages. Generating attack graphs is possible by creating new interaction

rules constituting a knowledge base.

In [22], Hankin et al. introduce a framework that produces an AG for a specific scenario based on Common Attack Pattern Enumeration and Classification (CAPEC) and Common Weakness Enumeration (CWE) standards. They relate CWEs and mitigations to identified attacks through the use of metadata. In this approach, the CWE metric aids in getting the vulnerability impacts involved in populating the vulnerability ontology. This work suggests aligning the vulnerability ontology's knowledge graph with a countermeasure knowledge graph to select countermeasures.

2.2.3 Ontology-based Attack Graph

Recently, ontologies have been used in different AG generation approaches. Falodiya et al. [23] propose an algorithm that goes through a semantic AG to add the information extracted from the AG into an ontology. This ontological approach stores other information, such as the countermeasures available for a vulnerability, their cost, and the CVSS Score. An ontological approach makes analyzing this information considerably more accessible as the network grows.

Lee et al. [24] presents an ontological representation of AGs. They create an ontology representing AGs for a simple network environment. Classes and relationships are created from the multi-requisite graph model. States, vulnerabilities, and prerequisites are shown as classes. This approach enhances the machine readability of large-scale AGs, thereby enabling the automation of network security assessments. Ontological structures facilitate security assessment. Using the ontological AG, experts can access the information needed for risk analysis without analyzing the entire AG. Analyzing the entire AG can be particularly time-consuming when the graph is enormous.

Wu et al. [25] introduce a security analysis approach based on an ontological AG generation. The AG generation approach is based on the ontology ability to infer new knowledge. The reasoning engine then receives the generated AG, enabling the calculation of risk metrics and the development of an optimal security mitigation plan. This approach does not deal with the infrastructure changes during the risk assessment process or in the recommendation of the mitigation plan.

Knowing where the adversary is located on a system is necessary. However, it is essential to take action to impede him/her from going further and causing more damage. Thus, the selection of appropriate countermeasures to the adversarial actions is essential. Automated countermeasures selection is one of the objectives of this thesis. Section 2.3 reviews some

contributions on countermeasures selection, some of them are based on AG.

2.3 Countermeasures Selection

This section presents the related works on countermeasures selection in the literature. Section 2.3.1 presents the countermeasures selection approaches non-based on AG. Section 2.3.2 presents the countermeasures selection approaches based on AG.

2.3.1 Non-AG based countermeasures Selection Approaches

Sawrup [26] proposes a remediation selection approach based on model checking that checks if a formal system model satisfies a given property. The approach involves remediation graph generation. The remediation graphs describe how a system can be corrected so that the system eventually achieves consistent states preferable to the initial state.

In [27], Roy et al. present a countermeasure selection method using attack-defense trees (ADTrees). ADTrees enhance traditional attack trees by integrating defensive measure trees, resulting in a mathematical model representing multi-stage attacks and corresponding protective measures. ADTrees consider both attacks and countermeasures. Similarly, Fraile et al. [28] propose a countermeasure selection approach based on ADTrees. While multiple countermeasures are identified, none is effective in preventing the attack.

Kordy et al. [29] introduce a method for multi-parameter security optimization using ADTrees. They utilize Pareto attributes, which are well-suited for ADTrees that include repeating actions. According to the Pareto principle, around 80% of the outcomes stem from approximately 20% of the causes. However, the proposed optimization concentrates entirely on the attacker's perspective, with countermeasures suggested without examining the actual costs or probabilities.

2.3.2 AG based Countermeasures Selection Approaches

Gonzalez-Granadillo et al. [30] introduce an approach for countermeasures selection based on the automated AGs generation. They develop a tool, StRori, which considers the information system state changes in applying new countermeasures. This approach combines AGs with a cost metric to analyze the countermeasure's impact and better select them, considering the costs incurred by applying each countermeasure. This approach considers many parameters for the countermeasures selection process, such as the security tools available in the system, their impact, the complexity of their execution, and the loss generated by their execution.

This approach ensures a good tradeoff between these parameters and the capacity of the security tools to execute the selected countermeasures.

Stan et al. [31] present an optimal countermeasure plan in terms of risk for a given budget based on AGs. Stan et al. use AGs for risk analysis, which makes it possible to consider the attacker's position and potential movements while taking into account the vulnerabilities of the system. However, the AG is not regenerated in analyzing the effectiveness of the countermeasure plan. Nevertheless, the AG update could increase awareness about the adversary's position and actions to block the attacker in real time. This thesis proposes ADG generation by instantiating nodes corresponding to the response of the AG nodes.

Ivanov et al. [32] present an approach for calculating security indicators, risk assessment, and selecting protective measures based on AGs. This approach relies on the system's risk and indicator level to select the protective measures. This process implies the suppression of all the AG arcs connecting any node with the target node, except for a vulnerability that connects this node with the target node. However, this impacts the time performance because the calculation time depends on the AG structure. The more strongly connected the AG components are, the longer the operating time. This thesis proposes a real-time IR plan based on monitoring a system by instantiating the IR response on the AG.

Some countermeasures selection approaches base the AG update after the countermeasure instantiation on anti-correlation. Cuppens et al. [33] introduce the concept of anti-correlation used to determine countermeasures to stop an intrusion. The proposed approach involves selecting and applying defense mechanisms when an intrusion occurs using Language to Model a database for Detection of Attacks (LAMBDA). LAMBDA provides a logical description of attacks and countermeasures.

Describing a LAMBDA attack consists of a pre-condition that defines the required system state for a successful attack, a post-condition that defines the system's state after the attack's manifestation, a detection that describes the alert, and a verification that specifies the conditions for verifying the attack success. Describing a countermeasure with LAMBDA involves a pre-condition defining the required system state for a countermeasure application. This post-condition defines the system state after the countermeasure application. This action establishes a necessary step for mitigation implementation measures and defines a verification process to assess if the countermeasures have been successfully applied.

Kanoun et al. [34] introduce an improvement to the countermeasure selection process based on the effectiveness quantification and the countermeasures selection causing the most negligible negative impact on the information system by adopting an approach of evaluation and risk analysis. This approach uses the LAMBDA language to merge the semi-explicit

correlation and anti-correlation functions. The generated alerts are collected and correlated after detecting the real-time attack. The generated AG enables the visualization of potential future steps an attacker might follow based on the semi-explicit correlation principle and the identification of candidate countermeasures using the anti-correlation principle.

Behbehani et al. [35] introduce a framework for Open Banking API security to identify security threats in FinTech integration via Open Banking API based on Bayesian Attack Graphs to automate the most exploitable attack path predictions. The approach is heuristic; for different maximum budgets, the heuristic algorithm can adjust the priority queues, leading to suboptimal predictions for the countermeasure plan. This thesis proposes a countermeasure plan based on optimal IR playbooks generated [36]. Section 2.4 reviews the related works on IR playbook generation.

2.4 Incident Response Playbooks Generation

NIST [37] describes a playbook as an action plan an organization can follow to contain an attack and recover from it. According to NIST, an IR lifecycle contains the stages of Preparation, Detection & Analysis, Containment, Eradication & Recovery, and Post-incident activities.

The RE&CT framework, inspired by MITRE ATT&CK, provides a manner to describe and categorize actionable IR techniques. In RE&CT, a playbook is a series of executable actions that help respond to an alert. However, it lacks parameters specifying how these actions should be carried out, whether they are automatable or require human intervention, and does not consider potential dependencies between actions. Each security action includes information about the requirements for its execution. These features and limitations have influenced this approach. In contrast, this work formally defines a playbook and offers more detailed information on executing the actions. RE&CT incident actions are also integrated into the IR ontology.

In [38], Islam et al. propose to generate automatic commands necessary for API communication to automate security tool integration. They integrate an ontology in their work, composed of three principal classes: security tool, activity, representing an action, and capacity, corresponding to the capacity of response of a tool. These classes are adopted in the proposed ontology.

Moreira et al. [39] propose using an ontology for security incidents management and processing. This ontology helps document IR actions taken in an organization. The proposed approach reuses two important concepts defined in this ontology: action and event. An action

is something that can be reached, and an event is something that can happen in a numerical context. These concepts exist also in the Unified Cybersecurity Ontology (UCO). [40].

Hutschenreuter et al. [41] propose the application of ontologies in port infrastructure resilience. After adaptation, some concepts in the ontology proposed in [41] as threat and security measures are reused in the ontology. Furthermore, the concept of security measures is not linked to an action workflow, which can limit the automatization.

Some relevant concepts constitute all these ontologies, but others must be more pertinent to the approach. Therefore, reusing or adapting the relevant concept in the ontology created for this solution is more convenient. An ontology helps formalize IR playbooks. By integrating it into a SOAR, such an ontology can address security orchestration, automation, and response (SOAR) limitations.

Ganin et al. [42] propose a multicriteria decision framework for cybersecurity risk assessment. This approach quantifies threats, vulnerabilities, and consequences according to a series of criteria, enabling the prioritization of countermeasures to mitigate the risk considering different criteria. Effectiveness, cost, and time required for a countermeasure execution are some inherent countermeasure criteria. The countermeasures with a lower cost and shorter time requirements have a higher subcriteria score and, thus, a higher overall alternative score.

OASIS proposes a schema and taxonomy for Collaborative Automated Course of Action Operations (CACAO) security playbooks [43]. This framework defines how to create, document, and share playbooks in a structured and standardized form. A CACAO playbook consists of a list of logically performed stages that an event can trigger manually or automatically. Playbooks can reference or include other playbooks. Each action within a playbook represents a security operation. CACAO specifies various workflow steps, linking each playbook component to a specific step, enabling the understanding of which action or playbook should begin or conclude the sequence. Some actions can run simultaneously and be categorized as parallel steps, while others depend on previous actions defined by the if-condition workflow step type.

This work considers the workflow step concept in this approach to define a logical orchestration for the actions composing an IR playbook. CACAO also defines a target object representing an entity or a device that accepts, receives, processes, or executes a command linked to an action. This work defines the concept **Tool** that represents an entity, device, and the concept command that can execute an action. A playbook contains actions; some can be carried out in parallel, and the execution of the actions is logical. Each action is linked to an IR step defined by NIST, and an action requires at least a device or entity to be carried out.

In [44], Empl et al. propose an approach to generate CACAO playbooks Common Security Advisory Framework (CSAF) documents for ICS. A limitation of this approach is that playbook generation is possible only for vulnerabilities in which documentation includes information about assets and components. This approach can generate playbooks for vulnerabilities whose description does not include this information; it is not mandatory to contain information about the adversary's actions and/or the attack impacts. Matching the vulnerability KG from VDO with a countermeasures KG, D3FEND, is proposed to select countermeasures involved in the playbook generation. When unavailable in the vulnerability KG, the information about assets and components can be inferred thanks to the matching with D3fEND KG.

In this approach, each D3FEND countermeasure is related to at least one RE&CT action. All the RE&CT actions linked to the exploited vulnerability are combined through an optimization algorithm, Pareto front defined in [45], ensuring optimal playbook generation. Thus, this approach ensures that whatever the adversary's behavior is, a playbook will be generated to respond to and recover from the attack. Section 2.5 explains how D3DEND is chosen for the countermeasures selection and introduces the related works on graph matching in Cybersecurity.

2.5 Graph Matching in Cybersecurity

In this approach, a standardized National Institute of Standards and Technology (NIST) ontology, Vulnerability Database Ontology (VDO), contributes to AG enrichment. This ontology can infer new paths to correlate the alerts with the information in the logical attack graph nodes. However, VDO does not provide mitigation actions for vulnerabilities. A countermeasures ontology KG with VDO KG must be matched to get mitigation data. Then, other security ontologies focusing on countermeasure actions for the incident response plan are compared.

Three security ontologies [46, 47] are analyzed to choose the most appropriate one for the IR plan. A countermeasure is considered a sub-concept of the device class in the ontology proposed by Hergoz et al. [47]. We cannot reuse the instances for the control and vulnerability classes from the ontology proposed by Fenz et al. ([46]) because they are derived from the German IT Grundschutz Manual ([48]). Our goal is to propose an approach based on a globally recognized standard.

Kaloroumakis et al. [49] propose a cybersecurity countermeasures KG D3FEND [1]. The KG

[1] `https://d3fend.mitre.org/`

represents the essential concepts concerning cybersecurity countermeasures and the relationship necessary to link those concepts together. Digital Artifact Ontology (DAO) is an ontology that specifies the necessary concepts to classify and represent digital objects of interest for cybersecurity analysis. The use of DAO makes it possible to associate the ATT&CK[2] offensive techniques with the D3FEND defensive techniques.

The association between offensive and defensive techniques is possible regarding the relationship of each technique with the digital objects. D3FEND and ATT&CK KGs are used in several approaches [50–52] to standardize attack behavior or rank mitigation actions. Some approaches are based on natural language processing techniques. D3FEND is chosen for the countermeasures selection process because the offensive techniques correspond to the entities of impact method and logical impact in VDO, and the artifact entities correspond to the context entities in VDO.

Currently, no contributions in the cybersecurity domain specifically address knowledge graph matching. However, some existing works focus on matching behavioral graphs and AGs. Park et al. [53] propose a malware classification method for detecting maximal joint sub-graphs. Similarly, in [54], Li et al. present an approach for automatically extracting behavior-based AGs from CTI reports and identifying corresponding attack techniques by matching these AGs with predefined technique templates.

Although some KG matching approaches exist in other domains, they are analyzed to determine their potential adaptability to this method. For example, Azmy et al. [55] propose an approach to match entities between DBpedia and Wikidata, focusing on resolving ambiguous entities. Their method involves matching an entity from DBpedia with its corresponding real-world entity in Wikidata, utilizing existing cross-ontology links (e.g., OWL:sameAs predicate) between the two. However, this approach is unsuitable for the proposed solution needs, as there are no cross-ontology links between VDO and D3FEND.

Pershina et al. [56] introduce an algorithm for aligning instances in large knowledge bases using Holistic Entity Matching (HolisticEM). However, the heuristic optimization phase of this approach causes different matches across executions. In contrast, this work aims to develop a method that ensures consistent and equivalent output from the graph-matching process. Therefore, this approach is discarded, and a novel countermeasure selection process is proposed based on graph matching in a cybersecurity KG.

[2] `https://attack.mitre.org/`

2.6 Conclusion

This chapter introduces the essential concepts for understanding the work in this thesis. These concepts are mentioned in the following chapters, which explain these contributions in detail.

This literature survey on AG generation, countermeasures selection, playbook generation, and graph matching has revealed significant gaps. These gaps are identified as opportunities for novel contributions. This work aims to enrich the field with innovative approaches of AG enrichment, graph matching for countermeasures selection, automated playbook generation, and ADG generation.

The essential concepts to understand this thesis are introduced. Chapter 3 explains this proposed approach to enrich AGs in real time based on alerts generated after threats, misbehavior detection, and inferred knowledge from ontology.

CHAPTER 3 AUTOMATED ATTACK GRAPH ENRICHMENT

Based on the conference paper [57] presented at IFIP SEC 2023 and published in Springer, this chapter presents an approach to automatically generate a logical AG based on network topology and enrich it based on monitoring system information and ontologies.

An AG represents the possible adversary actions to attack a system. Cybersecurity experts can lean on them to make decisions regarding IR plans. There are different AG-building approaches. This contribution focuses on LAGs. Network configuration constantly changes, and new vulnerabilities arise. An AG enrichment approach based on semantic augmentation of the logic predicates is proposed. Mapping AGs with alerts from a monitored system allows for confirming successful attack actions and updating according to network and vulnerability changes. The predicates get periodically updated based on attack evidence and ontology knowledge, allowing the verification of whether changes lead the attacker to the initial goals or cause further damage to the system not anticipated in the initial graphs. This approach is illustrated using a specific cyber-physical scenario affecting smart cities.

This chapter is organized as follows. Section 3.1 gives an overview of this contribution. Section 3.2 presents the logic-based AG generation approach. Section 3.3 demonstrates the need to monitor the information system. Section 3.4 demonstrates how a vulnerability ontology can be constructed for the AG enrichment based on vulnerability description. Section 3.2 describes the proposed AG enrichment approach. Section 3.6 provides the experimental results.

3.1 Overview

This approach generates a logical AG using prior knowledge about the network topology. After the AG generation, networks and network vulnerabilities may evolve (i.e., the configuration of system devices may change, software updates may be enforced, etc.). Therefore, the network is not exposed to the same vulnerabilities as at the beginning of the AG generation process. It is crucial to update the AG in response to system changes. The causality relationships between adversaries and systems must be reflected in the logical statements of nodes and edges to enhance a logical AG. An ontology-based approach for enriching logical AGs is proposed to meet these requirements.

3.2 Generation of the AG

Generating a logical AG requires defining rules that describe causality relationships. For instance, consider code execution: when code is executed on a machine, it grants an adversary access to the host. This scenario is represented by the logical implication outlined in the following rule:

$$execCode(h, a) \rightarrow canAccesHost(h)$$

where *canAccesHost(h)* is a logical predicate describing the accessibility to host h, and *execCode(h,a)* another predicate assessing that an adversary a executed code in h. The example can be extended as follows:

$$execCode(h, a) \quad \wedge \quad hasCredentialsOnMemory(h, u) \quad \rightarrow \quad harvestCredentials(h, u)$$

where *harvestCredentials(h,u)* describes a series of credentials harvesting on host h, *execCode(h,a)* the predicate that an adversary a is executing code on host h, and *hasCredentialsOnMemory(h,u)* the predicate of storing the credentials on the memory of host h, the example describes an adversary harvesting the credentials of a previous user that logged onto the system by finding them in the memory of that precise system.

3.3 Matching Vulnerability Information with Monitoring Information

Monitoring the information system is necessary to update the AG based on the system's real-time state. The monitoring process output can be continuously mapped with the AG's initial nodes to determine if a vulnerability is being exploited. The mapped information includes the port, IP address, and device protocol. This mapping is prioritized based on the alert's severity and the CVE's Common Vulnerability Scoring System (CVSS) score. This approach prioritizes the CVE with the highest CVSS version 2.0 score for alerts involving multiple vulnerabilities to determine its impacts using a vulnerability ontology.

Table 3.1 represents a system that contains three vulnerabilities concerning remote desktop protocol and exploitable using port 3389 and protocol TCP. This approach prioritizes CVE-2019-0708 due to its higher score. Then, the enrichment process is launched using semantic information about specific vulnerabilities.

Table 3.1 Comparison between various CVE concerning the same port and protocol based on their CVSS index

CVE-ID	**Product**	**Protocol**	**Port**	**CVSS**
CVE-2019-0708	windows remote_desktop_protocol	tcp	3389	9.8
CVE-2012-0152	windows remote_desktop_protocol	tcp	3389	9.3
CVE-2012-0002	windows remote_desktop_protocol	tcp	3389	9.3

3.4 Vulnerabilities and Ontologies

Vulnerability information is essential for an AG generation and enrichment. Standardized databases provide vulnerability descriptions, including preconditions, post-conditions, and practical exploitation methods. This information allows the semantic expression of an adversary's actions and goals, such as pre and post-conditions. Incorporating information about exploited vulnerabilities is crucial for the urgent and real-time updating of logical AGs. Since vulnerability information is typically written in natural language, it must be converted into machine-readable text. An ontology ensures that this machine-readable text is represented homogeneously. An ontology structures the domain, facilitating interoperability between the AG information and the data extracted from vulnerability databases. Queries can be made on the ontology to infer new knowledge necessary for the ontological enrichment of the AG.

Considering CVE-2002-0392 as an example. Its description states: "Apache 1.3 through 1.3.24, and Apache 2.0 through 2.0.36, allows remote attackers to cause a denial of service and possibly execute arbitrary code via a chunk-encoded HTTP request that causes Apache to use an incorrect size." This information can be categorized as shown in Table 3.2. From this table, an ontology can be constructed with the following classes: *CVE-ID*, *Product*, *AttackType*, *Method*, *Impact*, and the properties: *concernsProduct*, *hasRemoteType*, *hasMethod*, *resultsInImpact*.

3.5 Enrichment of AGs

Algorithm 1 describes the proposed method for enriching AGs using a vulnerability ontology and monitoring system information. When a threat is detected on a vulnerable component of the monitored system, it is crucial to examine the vulnerability characteristics to identify its post-conditions. As shown below, new rules are generated based on the attack goal and the deduced impact, enabling the logical reasoner to establish a new path from the resulting consequence to the attack goal.

These post-conditions allow the AG to be enriched with new paths. For an attack targeting physical damage with an exploited vulnerability that can cause a service interruption that

Table 3.2 Classification of CVE-2002-0392 characteristics

CVE-ID.	**Product**	**Attack Type**	**Method**	**Impact**
CVE-2002-0392	Apache	remote	Code Execution	Privilege Escalation
CVE-2023-4194	Linux	local	Authentication Bypass	Privilege Escalation
CVE-2023-40023	Yaklang	local	Trust Failure	Read
CVE-2019-0708	Microsoft Windows	remote	Code Execution	Service Interrupt
CVE-2021-21277	Angular	remote	Code Execution	Privilege Escalation
CVE-2017-0144	Microsoft Windows Server	remote	Code Execution	Privilege Escalation
CVE-2021-26828	OpenPLC ScadaBR	remote	Code Execution	Privilege Escalation
CVE-2023-48795	OpenSSH	remote	Man-in-the-Middle	Modify

can be shut down, reboot, or panic, the enrichment process adds a new path from the node consisting of the vulnerability exploitation and the node expressing the physical damage.

$$\begin{array}{ll} shutdown(host) & \wedge \\ hacl(host, bus, protocol, port) & \rightarrow \\ physicalDamage(bus) & \\ execCode(host, user) \rightarrow shutdown(host) & \end{array}$$

3.6 Experimental Approach

3.6.1 Use Case Scenario

Next, a use case scenario provided by smart city stakeholders involved in this contribution validation is described. A denial-of-service attack is instituted against a municipality network, disrupting communication between machines and sensors. This interruption leads to delays in the city's transportation services. While people are attempting to leave the area, conflict starts, forcing the authorities to close all transportation services. The brutality in public transport on a bus affects the health of several passengers.

Algorithm 1: AG Enrichment Process

```
Data: System state
G = (V, E) ← AttackGraphGenerated;
initialimpact ← impact of exploited vulnerability in pro-active graph;
listimpact ← List of impacts from the SPARQL query;
V={v_0, v_1, ...,v_n} ← List of vertices of G;
E={e_0,e_1,...,e_m} ← List of edges of G;
Result: G'
initialization;
for i=0; i<len(listimpact); i++ do
    if listimpact[i] ≠ initialimpact then
        if impact = Shutdown ∪ impact = Reboot ∪ impact = Panic then
            for z=0; z<len(V); z++ do
                if V[z] is a fact node ∩ attack goal is PhysicalDamage then
                    Add new rules to the logical reasoner;
                    G' ← the AG regenerated;
                    i ← the number of vertices added;
                    V'={v_0, v_1, ...,v_{n+i}} ← List of vertices of G';
                    t ← the number of edges added;
                    E'={e_0,e_1,...,e_{m+t}} ← List of edges of G';
    G' = (V', E');
```

The rationale behind the scenario depicted in Figure 3.1 unfolds as follows: An adversary successfully executes arbitrary code on the starting device by remotely connecting through Remote Desktop Protocol (RDP). A network service enables users to control computers remotely. Subsequently, the adversary gains access to read the starting device memory, where administrator credentials are stored. The adversary then gathers these credentials. Assuming the administrator has remote management access to all machines in the domain, the adversary, with these credentials obtained, logs into the breach point and gains remote access to the critical asset from there.

To listen in network traffic, the adversary executes a DNS Poisoning attack [58]. Additionally, the adversary conducts integrity attacks to alter application-level data, such as bus schedules and routes, disrupting traffic patterns and causing congestion. The traffic disruption and congestion drive citizens to board incorrect buses at inappropriate times. Then, it results in panic and potential brutality. Concurrently, leveraging the domain credentials, the adversary steals additional access keys and mimics other users, as depicted in Figure 3.1 under steps labeled *Access Keys Stealer* and *User Compromise*.

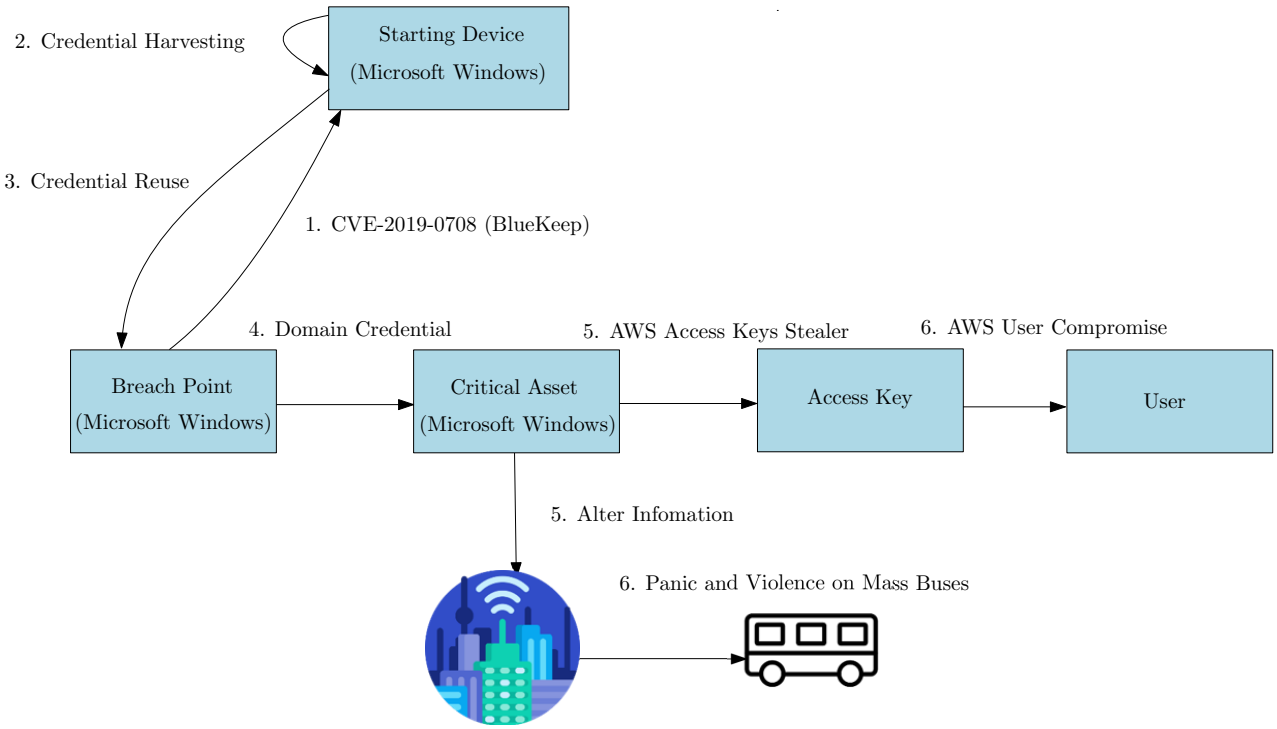


Figure 3.1 **Cyber-physical attack scenario**: An attacker exploits the vulnerability linked to CVE-2019-0708 on a Starting Device. Subsequently, the adversary retrieves administrator credentials from the device's memory and utilizes them to gain control over a critical asset. This attack impacts the system's physical and digital components, including individuals and services.

3.6.2 Setup

The scenario illustrated in Figure 3.1 is implemented to validate this approach. It exemplifies a Cyber-Physical System (CPS). This CPS is monitored by a Security Information and Event Management (SIEM), Prelude-OSS[1]. A machine that is responsible for monitoring CPS is configured. The SIEM and this AG enrichment solution are installed on this machine. The monitoring of the system enables alert generation when adversarial actions are detected. A virtual machine as the scenario's starting device for the simulation is configured. Another virtual machine represents the breach point; a third represents the critical asset.

Referring to Figure 3.2, this process requires using Nessus Essentials[2] scanner (Step 1) to identify and list vulnerabilities within the monitored system. Therefore, the data from Nessus is utilized by MulVAL [59], a logical programming-based reasoning engine (Step 2), to generate a logical AG (Step 3).

[1] `https://www.prelude-siem.com/en/oss-version/`
[2] `https://www.tenable.com/products/nessus/nessus-essentials`

Prelude-ELK[3], an extended version of Prelude-OSS, enables real-time monitoring of the system. When an alert happens, the procedural processor correlates the alert information with details from the AG nodes (Step 4) to ascertain whether the alert relates to a known vulnerability in the system.

In addition, the procedural processor queries a vulnerability knowledge graph (Step 5) to infer the impacts of the exploited vulnerability. Upon deducing a new impact (Step 6), the procedural processor integrates new rules into the logical reasoner (Step 7). Subsequently, the AG is updated to include a new path (Step 8).

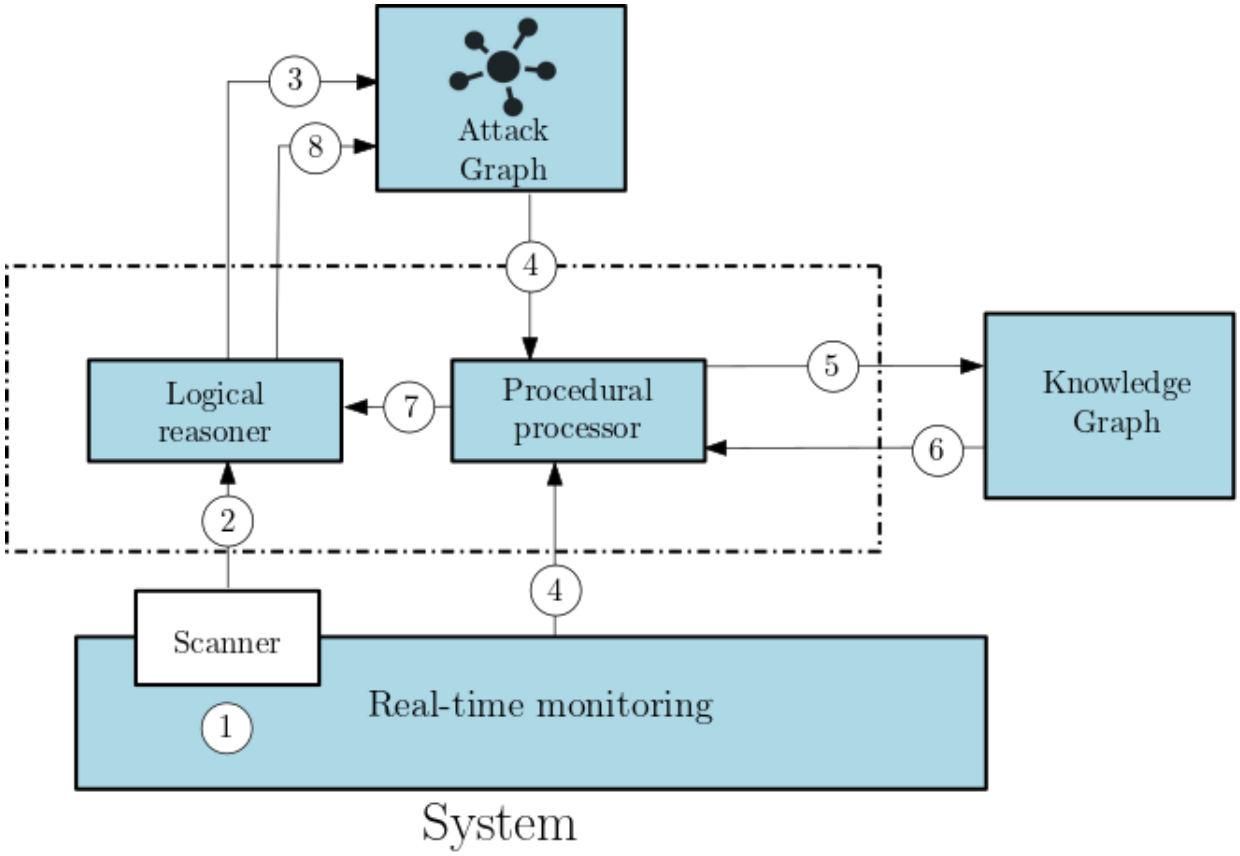


Figure 3.2 The AG Enrichment Process

MulVAL

Based on the scenario depicted in Figure 3.1, input data for MulVAL is generated, and interaction rules are developed based on the vulnerability and the described scenario. The new interaction rules are encoded as Horn clauses [59]. The first line represents the first-order

[3] `https://github.com/Kekere/prelude-elk`

logic conclusion, while subsequent lines denote the enabling conditions. The clauses below correspond to the following statement derived from the scenario in Figure 3.1:

> *"The breach point credentials can be harvested on the starting device only if there is previously an execution code exploit on the starting device and the administrator credentials are saved onto the memory of the starting device."*

```
harvestCredentials(_host, _lastuser) :-
   execCode(_host, _user),
   hasCredentialsOnMemory(_host, _lastuser).
```

The clauses below represent the following facts:

> *"It is possible to execute code on the breach point when these credentials have been harvested and because the breach point and the starting device are on the same network and can communicate through a given protocol and port."*

```
execCode(_host, _user) :-
    networkServiceInfo(_host, _program, _protocol, _port, _user),
    hacl(_host, _h, _protocol, _port),
    harvestCredentials(_h, _user).
```

The following clause depicts this fact:

> *"Physical damage can be caused on the bus because an adversary modified information on a critical asset".*

```
physicalDamage(_bus) :-
    modify(_host, _info))
```

Ontology

The Vulnerability Description Ontology (VDO), an ontology of CVEs developed by NIST, is used. Figure 3.3, adapted from [60], illustrates several attributes within the VDO ontology that characterize software vulnerabilities. Essential features such as *Impact Method* and *Logical Impact* are mandatory in the ontology. *Impact Method* delineates how a vulnerability may be exploited, while *Logical Impact* specifies the potential consequences of successfully exploiting the vulnerability.

There are several types of service interruption. Then, when the vulnerability impact consists of service interruption, one of these types should be selected. It is why the class *ServiceInterruptTypes* is created. Similarly, an adversary can gain different privilege levels; hence, the class *PrivilegeLevel* is created.

The ontology's information classes are populated for each CVE impacting the monitored system based on the CVE's description and metrics.

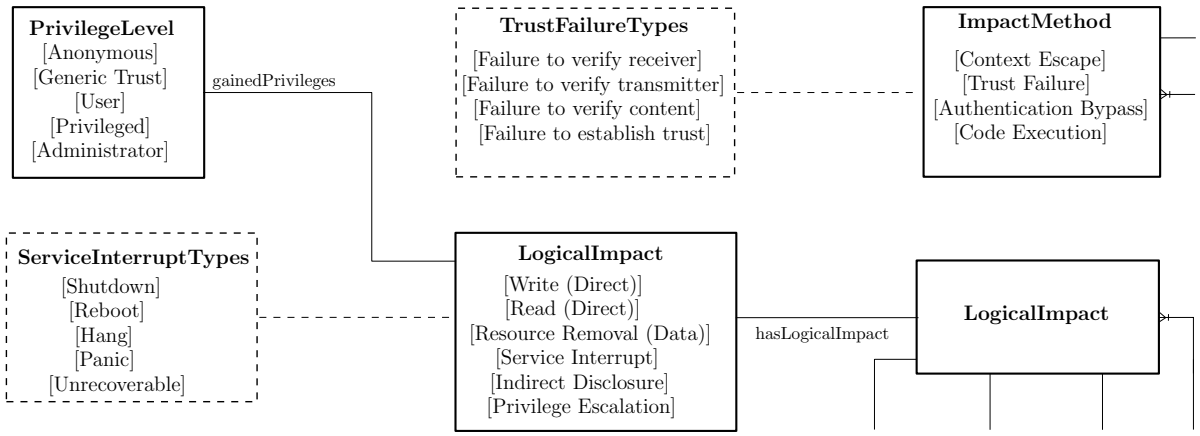


Figure 3.3 Sample classes associated to VDO.

Prelude-ELK

Prelude is a SIEM that collects, normalizes, sorts, aggregates, correlates, and reports all security-related events independently of the product brand or license, giving rise to such events. Thanks to the log processing module Log Management Lackey (LML), Prelude is agentless. It can also recover any type of log (system logs, Syslog, flat files, etc.).

ELK is an acronym for three open-source projects: Elasticsearch, Logstash, and Kibana. Elasticsearch is an open-source RESTful and distributed engine search. Logstash is a pipeline that ingests data from various sources simultaneously and transforms it. Logstash sends the transformed data to Elasticsearch, where it is stored. Kibana allows users to visualize and manage data in Elasticsearch.

An enhanced version[4] of Prelude-OSS's LML (Log Monitoring Lackey) is used along with third-party sensors like Suricata[5] to monitor and analyze Syslog messages generated by various hosts across heterogeneous platforms. Rsyslog Windows Agent[6] and Suricata are installed

[4] `https://github.com/Kekere/prelude-elk`
[5] `https://suricata.io/`
[6] `https://www.rsyslog.com/windows-agent/`

on each virtual machine, enabling monitoring Prelude to achieve this. All privileges to the prelude database are granted to the user "prelude". Logstash ingests the "Prelude_Alert" table by using the JDBC plugin. Logstash ingests the generated alerts into Elasticsearch. The queried information from Elaticsearch is correlated in real-time with data from VDO, facilitating this AG enrichment process.

Procedural processor

A procedural processor with PHP, Javascript, HTML, and Python is developed to input the necessary data to generate the AG. The engine provides a web-based visualization of the AG. It includes a module that automatically queries the alert information from Elasticsearch and correlates the latest alert's IP address, port, and protocol with the AG. When a vulnerability is potentially exploited, the engine refers to the vulnerability ontology to infer additional impacts of the exploit. The tool generates new interaction rules that enrich the AG based on ontology inference.

3.6.3 Results

Figure 3.4 (a) represents the AG generated for the scenario sketched in Figure 3.1. The goal, represented by Node 1, is to cause panic and violence (see the use-case scenario described in Section 3.6.1). A red node represents the existence of a vulnerability on a device. An orange node represents network configuration, e.g., device characteristics, the connection between two devices in the network, etc. When the preconditions are satisfied, a yellow node represents the inference rules leading to a fact. Green nodes represent facts. For instance:

- Node 15 represents the network access to the Starting Device, using Remote Desktop Protocol (RDP) services, from the attacker located on the Breach Point (Node 18).
- Node 20 depicts the vulnerability identified as *CVE-2019-0708* on the Starting Device.
- Node 19 concerns the network configuration of the Starting Device; port 3389 is opened, allowing remote connection to the device using remote desktop service.
- Node 14 represents the rule that leads the adversary to remotely exploit the vulnerability on the Starting Device when preconditions on Nodes 19, 20, and 15 are met.

Alerts are processed with Prelude-ELK (see Section 3.6.2) in real-time. The procedural processor correlates precondition nodes' information with alert information, such as IP address, protocol, and port. Thanks to VDO inference, the procedural reasoner receives other post-conditions associated with *CVE-2019-0708* as represented in Listing 1.

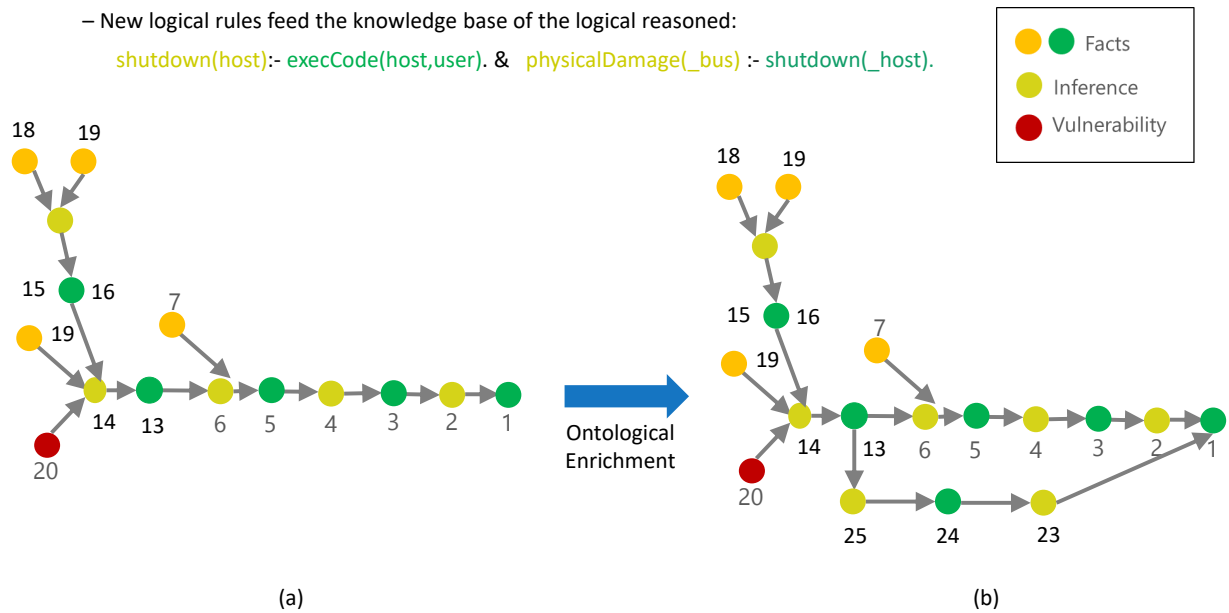


Figure 3.4 Sample outcomes: (a) AG generated for the scenario involving violence on buses. (b) The AG enriched with inferred ontology data.

Table 3.3 represents a list of deduced impacts from the SPARQL query concerning privilege escalation and communication interruption between the affected device and other devices. Delayed information dissemination to citizens, including public transportation users, is a significant consequence. The interruption of service can also lead to brutality on public transportation. Consequently, new paths are added to the AG based on the deduced impacts. As a result, an enriched AG is obtained.

Table 3.3 List of impacts inferred.

CVE-ID	**Logical Impact**
CVE-2019-0708	Privilege Escalation
CVE-2019-0708	Reboot

Figure 3.4 (b) represents such an enriched AG. The three nodes highlighted with the red square correspond to the new nodes added to the enriched AG, thanks to the ontology inference.

- Node 24 describes the service interruption.
- Node 25 is an interaction rule expressing the achievement of Node 24 because the precondition represented by Node 13 is met.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://www.semanticweb.org/keren/ontologies/2022/6/...

SELECT DISTINCT ?v ?impmethod ?log
WHERE {
    ?vul :hasIdentity ?vulid .
    ?vulid :value ?v .
    ?vul :hasScenario ?sce .
    ?sce :hasAction ?ac .
    ?sce :affectsProduct ?prod .
    ?ac :resultsInImpact ?im .
    ?im :hasLogicalImpact ?log .
    ?ac :hasImpactMethod ?imp .
    ?imp :value ?impmethod .
    FILTER (str(?v) ="CVE-2019-0708") .
}
```

Listing 1: Query allowing to deduce other post-conditions of CVE-2019-0708

- Node 23 leads to the violence on buses scenario (i.e., by inference, Node 24 targets Node 23, a new rule concerning the attack goal).

In Figure 3.4, the two extra nodes represent a new path the attacker can take to cause panic and violence. The new path is shorter than the former one. The adversary can attain the goal, represented by Node 1, much sooner than expected. This difference would make operators conscious that applying an IR plan is more urgent. Experts can then prioritize incident response actions that prevent the more impactful attacker's actions.

3.7 Discussion and Conclusion

An ontology-based approach for enriching AGs is proposed. In this approach, attacks are represented with logical graph model predicates. Successful validation of preconditions indicates successful attack execution. Contrary to similar approaches, such as topological and probabilistic AGs, this approach facilitates inference by ensuring that the edges indicate causality. This approach is implemented using existing software and validated with a cyber-physical use case provided by smart city stakeholders. This validation encompasses the entire process, from generating an initial AG through network vulnerability scans to updating the graph by

mapping monitoring alerts and ontology semantics in real-time. The predictions from the initial graph are successfully integrated into the enriched graph through inferences made by the ontology, leveraging both expert knowledge and monitoring data. Chapter 4 focuses on selecting countermeasures for exploited vulnerabilities.

Some future work can be considered. Automated generation of interaction rules from the information from MITRE ATT&CT techniques and techniques is a future direction for the research. Interaction rules generation from information in the input file while generating AGs when no attack path is found based on the input information is another possible future work.

CHAPTER 4 AUTOMATED COUNTERMEASURES SELECTION

Based on the conference paper [61] presented at Scisec 2024, this chapter presents an approach to automate the countermeasures selection process for an exploited vulnerability from a cyberattack. An approach to match a knowledge graph from a vulnerability ontology, Vulnerability Description Ontology (VDO), and the countermeasures knowledge graph, D3FEND, is proposed to mitigate cyberattack impacts. This approach uses machine learning and inference techniques to match entities from VDO and D3FEND to select candidate countermeasures to an attack. The aim is to automatically select countermeasures intended to be part of an IR playbook for a vulnerability. This approach application is demonstrated in a WannaCry use-case scenario. This countermeasures selection approach is validated by comparing the countermeasures automatically selected with those proposed in the literature for a WannaCry attack.

The chapter is organized as follows. Section 4.1 provides an overview of the proposed contribution. Section 4.2 describes the dataset involved in the countermeasures selection process. Section 4.3 details the methodology for matching VDO and D3FEND to select countermeasures. Section 4.4 shows the validation of graph-matching models. Section 4.5 explains the techniques involved in selecting the countermeasures. Section 4.6 evaluates and applies the approach's efficiency in a use case. Section 4.7 discusses the advantages and challenges of this approach and then concludes this chapter.

4.1 Overview

A method for selecting countermeasures based on KG matching is proposed. This method enables the selection of countermeasure actions for an attack scenario in this AG generation process, sketched in Figure 4.1. The blue components in Figure 4.1 represent elements involved in this AG enrichment process presented in Chapter 3.

In Step 1, the system is scanned. In Step 2, the scanning output is ingested into a logical reasoner, facilitating AG generation in Step 3. When an adversary exploits a vulnerability, as illustrated in Figure 4.2, the procedural reasoner correlates the alert information with the AG data in Step 4. In Step 5, the vulnerability ontology infers new information. After receiving the inferred information in Step 6, the procedural reasoner releases new rules to the logical reasoner in Step 7, resulting in an updated AG with a new attack path in Step 3.

Following this, as illustrated in Figure 4.2, the ADG generation process can begin. The

components in gray in Figure 4.1 are involved in this process. Even though the procedural reasoner does not trigger the AG enrichment procedure, the ADG generation begins, as shown in Figure 4.2. In Step 10, an ADG is created by correlating the AG to an Incident Response (IR) playbook for the exploited vulnerability identified in Step 9. An IR playbook outlines an organization's steps and procedures to address and mitigate incidents. If no existing playbook is available for the identified vulnerability, this solution automatically generates one from a list of countermeasures, as demonstrated in Step 8 of Figure 4.1.

If the playbook is not generated due to the absence of countermeasures for the exploited vulnerability, the automated countermeasures selection process is executed (see Figure 4.2). The components involved in selecting countermeasures are in white, along with the output of the chosen countermeasures. The dashed box represents the graph-matching mechanism involving the VDO and D3FEND Knowledge Graphs (KGs). While humans can periodically execute the graph-matching process and scan the system, it is automatically triggered for a specific exploited vulnerability when no countermeasures are available. This process outputs the necessary countermeasures in Step 7, which are required to generate the playbook necessary for ADG generation. This contribution focuses on the selection of countermeasures through graph matching. The following section presents the dataset used in the graph-matching process.

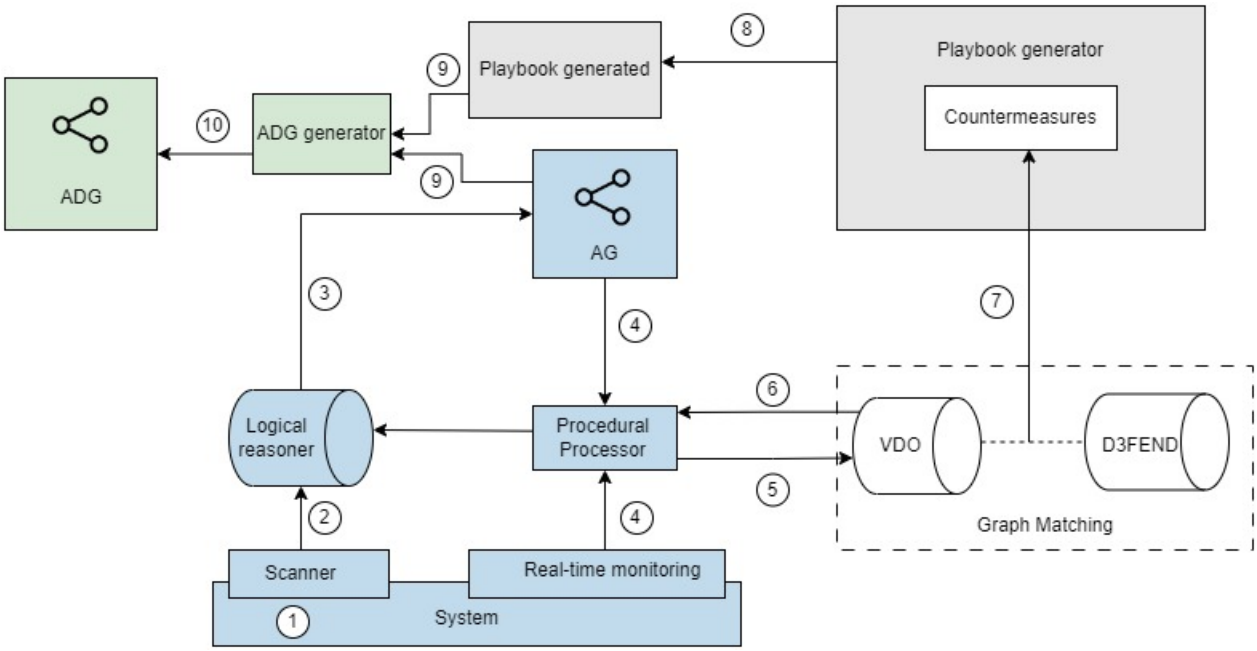


Figure 4.1 The ADG generation process

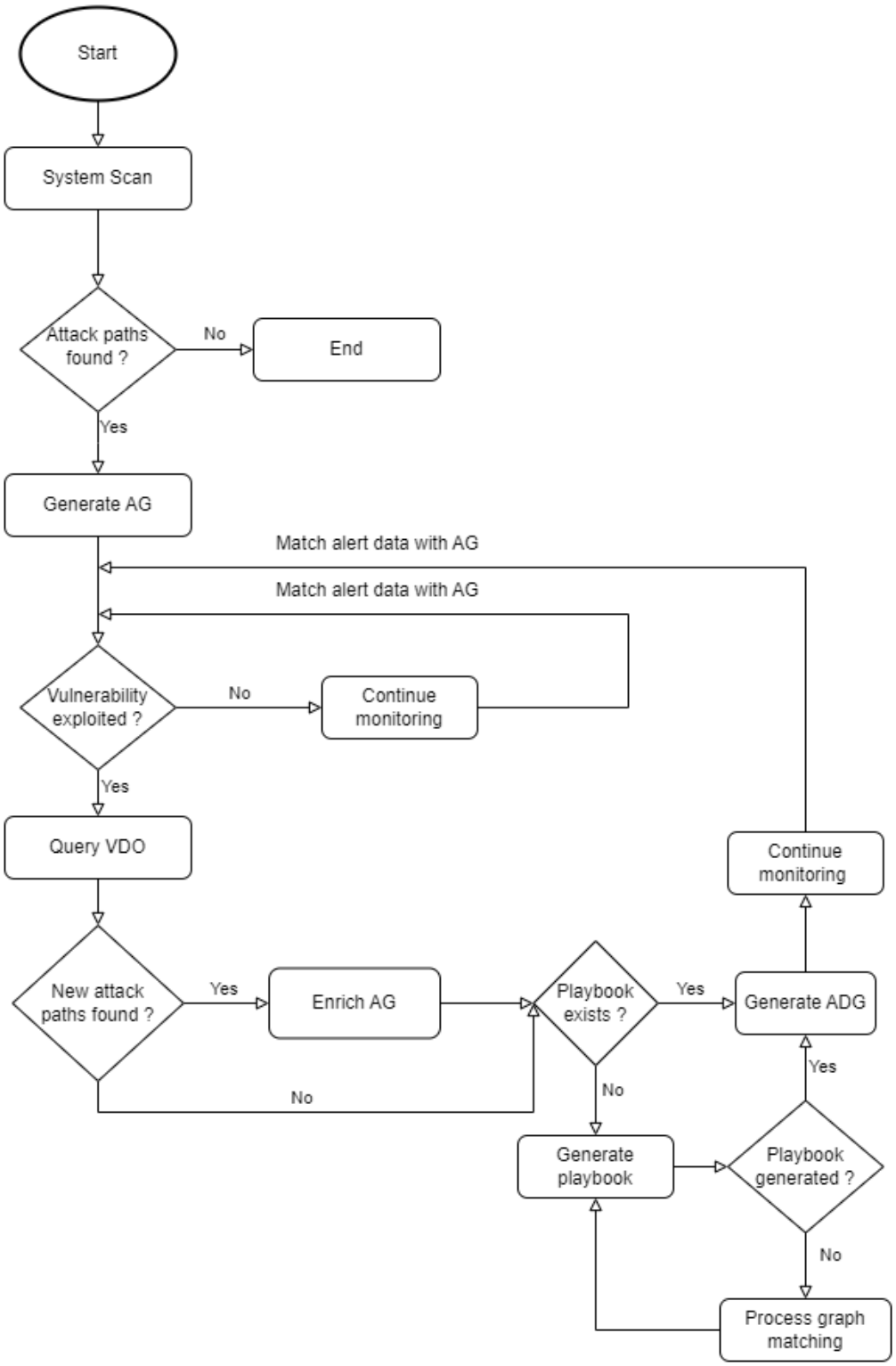


Figure 4.2 A flow chart of the ADG generation process

4.2 Experimental Dataset

The approach dataset comprises a VDO Knowledge Graph (KG) sub-graph and a D3FEND KG sub-graph. VDO outlines vulnerability exploitation pre-conditions and post-conditions, whereas D3FEND [49] offers corrective actions for specific system exploitations. VDO and D3FEND are analyzed to identify the relevant classes for this countermeasures selection method. Figure 4.3 shows the chosen sub-graphs for D3FEND and VDO.

In D3FEND, the relevant classes are *OffensiveTechnique*, *Artifact*, and *DefenseTechnique*. These classes are chosen to streamline the graph embedding process and minimize noise in the matching process. The *OffensiveTechnique* class refers to the methods adversaries use to attack a system and the impacts of those attacks. The *Artifact* class indicates the components affected by an offensive technique, while the *DefensiveTechnique* class invokes the countermeasures that target an artifact. Consequently, the sub-graph composed of all entities of types *OffensiveTechnique*, *DefensiveTechnique*, and *Artifact*, along with the properties that define the relationships between these entities are selected.

In VDO, the classes *LogicalImpact*, *ImpactMethod*, *Action*, *Context*, and *VulnerabilityIdentifier* are selected. Only these classes are chosen to facilitate graph embedding and avoid noise during matching. These classes are crucial to comprehending the assets of a system affected by a scenario, how an adversary can exploit a vulnerability and the consequences of this vulnerability. *LogicalImpact* designates the consequences of exploiting a vulnerability. *ImpactMethod* refers 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 associated. Therefore, the class *Action* is considered to complete meaningfully the sub-graph required for the matching. The class *Action* allows correlating an entity of the class *VulnerabilityIdentifier* with entities of the classes *ImpactMethod* and *LogicalImpact*.

4.3 Graph Matching

The proposed 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 $C_{O'}^k = e_0^k, e_1^k, ..., e_n^k$. The approach is based on graph embedding, word embedding, and SPARQL queries.

Figure 4.4 outlines this solution architecture. The two subgraphs of D3FEND and VDO KGs constitute the input. The automated pre-processing phase of the KGs comprises parsing the

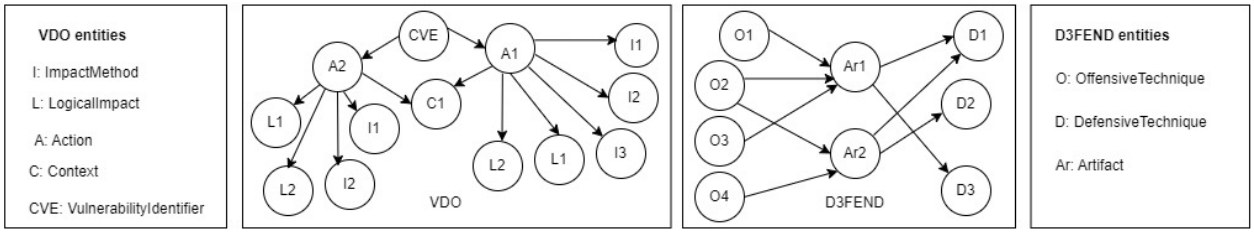


Figure 4.3 VDO and D3FEND sub-graphs

KGs, text processing, and corpora creation. The pre-processing phase is essential before launching the Word2Vec (see Section 2.1) models training and embedding KGs. Afterward, the graph-matching process begins automatically.

In the matching phase, the solution automatically calculates the cosine similarity, defined in Section 2.1, between the offensive techniques from D3FEND and the impact and method from VDO. The automated retrieval of artifact entities linked to the offensive techniques is possible thanks to a SPARQL query. 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 artifacts needed to query candidate countermeasures for each CVE. The output from the matching process consists of a list of candidate countermeasures for each CVE.

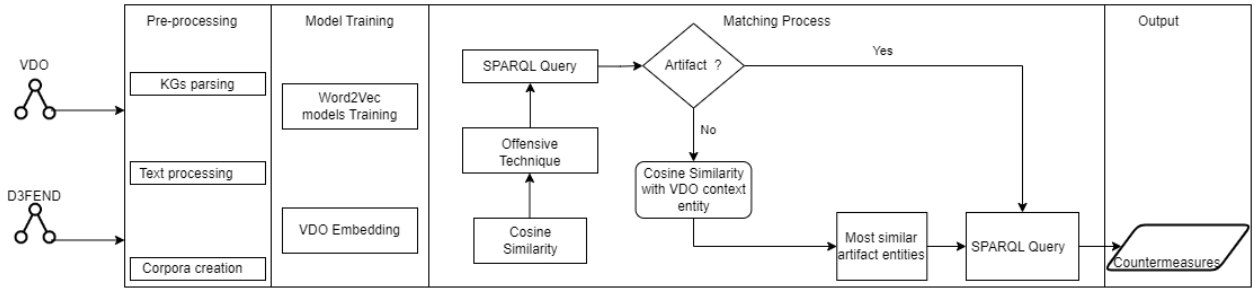


Figure 4.4 Architecture of the proposed solution

4.4 Graph Matching Models Validation

The parameters' values influence the performance of a Word2Vec model. Window size is the size of the context window used to predict surrounding words. A smaller window size focuses on a more specific, localized context, resulting in more accurate word associations. The window size is set to 5. The min_count value serves as a threshold, ignoring words that

appear fewer times than this value during training; it is set to 1. The number of workers indicates the number of CPU cores used for parallel processing. A higher number of workers results in faster training. The number of workers is set to 20. The vector size represents the dimensionality of the word vectors. Higher dimensions capture more semantic nuances and context but may lead to overfitting. The number of iterations over the training corpus refers to the number of epochs. Fewer epochs reduce training time but may result in underfitting, while more epochs enable the model to learn better and converge more fully.

Several Word2Vec model training for the method and impact corpora are performed by modifying the vector size and the number of epochs. To select the best Word2Vec model, the number of VDO entities with correct matches and the similarity score for the matches with the most similar D3FEND entity are considered. Table 4.1 compares some models based on the number of correct matching for impacts and methods in VDO and the average cosine similarity score for accurate prediction.

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

Table 4.1 Word2Vec model prediction evaluation

The green rows in Table 4.1 mark 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. For this model, the average similarity score is 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 low similarity score value 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

The KGs are parsed using SPARQL queries and splitting to extract the required entities for the model's training and matching process. An impact corpus is generated with entities from *ImpactTechnique* and *PrivilegeEscalationTechnique*, both subclasses of *OffensiveTechnique*, as well as *LogicalImpact*. A method corpus is created using entities from the other subclasses of *OffensiveTechnique* and the *ImpactMethod* class. Additionally, an artifact corpus is developed with entities from *Artifact* and *Context*, along with a general corpus containing entities from all classes.

Two Word2Vec models with the impact and method corpora, respectively, are trained. The Word2Vec impact model parameters and the Word2Vec method model parameters are fixed based on the best models represented in Table 4.1 in Section 4.4. A Word2Vec model takes a corpus text and outputs the word vectors. It first constructs a vocabulary from the training text data and then learns the vector representation of words. To get the KGs embeddings with RDF2Vec, the general corpus for the Word2Vec embedder is used.

The VDO graph embedding literals for each CVE is obtained. Then, for each entity classified as an impact from VDO, the cosine similarity is calculated between this entity and each offensive technique entity of the impact corpus using the trained model for the impact corpus. The output is the most similar offensive technique for the impact. The cosine similarity is calculated 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 is the most similar offensive technique for the adversarial exploitation method.

The similarity score varies from 0 to 1. A score equal to 1 for an entity means that this entity is similar to the input entity. However, an entry entity can have several matches. In this case, the graph-matching process chooses the ones whose score is higher than the given similarity threshold from those entities. A SPARQL query is executed on the D3FEND KG to retrieve

the artifact entities associated with the offensive techniques. However, the SPARQL query may not get a specific artifact entity. In this case, the solution calculates 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, the output is the artifact with the highest score. Afterward, a SPARQL query follows to deduce all the countermeasures related to the artifact entities.

4.6 Implementation Results

4.6.1 VDO Population

VDO is an ontology that describes vulnerabilities, the required conditions for exploitation, the consequences, and the product concerned. This approach involves implementing VDO in DL and automatically generating individuals for specific vulnerabilities to construct the KG.

Web scraping is a technique used to extract data from a website. Web scraping is employed to extract CVE descriptions from the https://nvd.nist.gov/vuln website. The information extracted is the description, the Common Product Enumeration (CPE), and the Common Weakness Enumeration (CWE). Then, thanks to the CWE, web scraping is automatically applied to the https://cwe.mitre.org/data/ website to determine the impact of the vulnerability. From the description of the vulnerability, thanks to pre-processing techniques, information about the adversary location for the vulnerability exploitation, the method of exploitation, and the product impacted are extracted.

Thanks to BeautifulSoup library, this solution automatically finds the last individual tag and finds where the new individual tags should be added to the ontology file. Individual tags of this type "<owl:NamedIndividual rdf:about="http://www.semanticweb.org/vdo#CVE-2019-0708>" are created for the vulnerability has been extracted thanks to web scraping and serve to populate the ontology.

4.6.2 Countermeasures Selection Evaluation

The trained models highlighted in green in Table 4.1 are considered for the countermeasures selection procedure[1]. This solution automatically executes a SPARQL query represented by Listing 2 on D3FEND for each selected offensive technique to deduce related artifacts and their defensive techniques. If no artifact is related to an offensive technique, another SPARQL query depicted by Listing 3 is launched to get all the digital artifact entities. Then, the cosine

[1] `https://anonymous.4open.science/r/graph-matching-BE38`

similarity score is calculated to find the top 5 matching artifacts that match the context entity in VDO most. A SPARQL query in Listing 4 obtains the corresponding defensive techniques for each match. The efficiency of the selected countermeasures is evaluated using the precision, recall, and F1 score metrics defined in 2.1.

```
PREFIX : <http://d3fend.mitre.org/ontologies/d3fend.owl#>
Select Distinct ?l where{
  {
      ?sb rdfs:subClassOf ?of.
      ?of rdfs:subClassOf+
       ↪  <http://d3fend.mitre.org/ontologies/d3fend.owl#OffensiveTechnique>
       ↪  .
      ?of <http://www.w3.org/2000/01/rdf-schema#label> '"""+t+"""' .
      ?sb ?p ?b .
      ?b rdfs:subClassOf*
       ↪  <http://d3fend.mitre.org/ontologies/d3fend.owl#Artifact> .
      ?b <http://www.w3.org/2000/01/rdf-schema#label> ?l .
  }
  UNION
  {
      ?of rdfs:subClassOf+
       ↪  <http://d3fend.mitre.org/ontologies/d3fend.owl#OffensiveTechnique>
       ↪  .
      ?of <http://www.w3.org/2000/01/rdf-schema#label> '"""+t+"""' .
      ?of ?p ?b .
      ?b rdfs:subClassOf*
       ↪  <http://d3fend.mitre.org/ontologies/d3fend.owl#Artifact> .
      ?b <http://www.w3.org/2000/01/rdf-schema#label> ?l .
  }
}
```

Listing 2: SPARQL query to get all the artifact entities linked to an offensive technique

The number of False Negative (FN), True Positive (TP), and False Positive (FP) are enumerated for the defensive techniques matched with each method and impact. The precision, recall, and F1 score are calculated. Finally, the macro-value for these metrics is calculated: the average recall, precision, and F1 score value for the method and impact, respectively. Table 4.2 and 4.3 represent the value for these metrics for the methods and impacts, respectively.

The average F1 score for the selected countermeasures for the impacts and the methods is higher than 0.9. Thus, this approach for automatically selecting countermeasures is efficient.

```
PREFIX : <http://d3fend.mitre.org/ontologies/d3fend.owl#>
Select Distinct ?l where{
    ?of rdfs:subClassOf+ ?o .
    ?o <http://www.w3.org/2000/01/rdf-schema#label> '""" +t+"""' .
    ?of <http://www.w3.org/2000/01/rdf-schema#label> ?l .
}
```

Listing 3: SPARQL query to get all the artifact entities sub-classes of "Digital Artifact"

```
PREFIX :<http://d3fend.mitre.org/ontologies/d3fend.owl#>
select ?label ?def ?desc ?p where {
<http://d3fend.mitre.org/ontologies/d3fend.owl#"""+asset+"""> rdfs:subClassOf* ?class .
{
    select ?label ?def ?desc ?p where{
        ?def rdfs:subClassOf+ <http://d3fend.mitre.org/ontologies/d3fend.owl#DefensiveTechnique> .
        ?def ?p ?class .
        ?def rdfs:label ?label .
        ?def <http://d3fend.mitre.org/ontologies/d3fend.owl#definition> ?desc .
        }
    }
}
```

Listing 4: SPARQL query to get the list of countermesures

Table 4.2 Evaluation of the prediction for the methods

Impact MethodMetric	**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 4.3 Evaluation of the prediction for the impacts

Logical ImpactMetric	**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

4.6.3 Illustrative Use Case

This section validates this approach thanks to an illustrative use case represented in Figure 4.5. A WannaCry use case is elected because ransomware is the most trending attack impacting organizations in terms of business process and financially [62, 63]. The use case concerns a Maritime Transportation System (MTS). MTS is a critical sector; in the past, attacks such as the one impacting Maersk [64] have revealed how a ransomware attack against an MTS organization can have a severe financial impact. A terminal port infrastructure

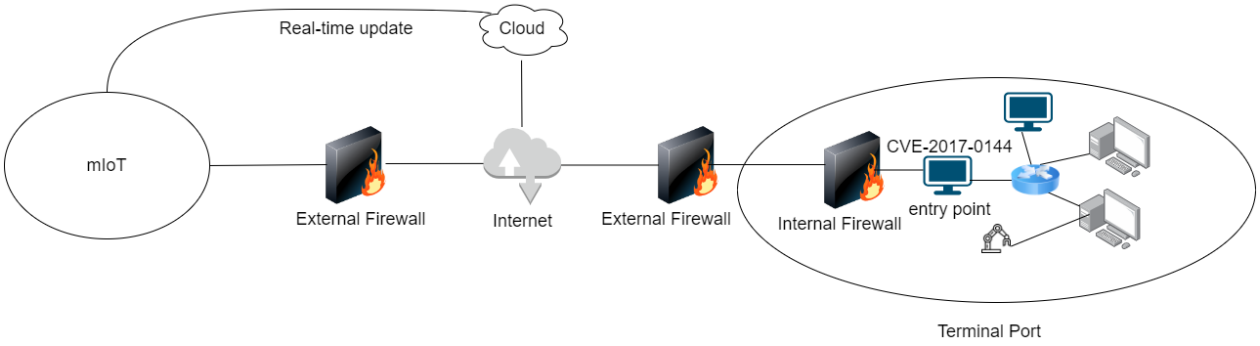


Figure 4.5 A Maritime Transportation System use case

comprises several workstations and a robot hoist that unloads merchandise from ships. The terminal port receives data from the cloud through an internet connection. There are external and internal firewalls between the port network and the internet. The maritime Internet of Things (mIoT) system communicates in real time with the cloud. A user connected to 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 and encrypts files. Ransom requests follow. The terminal port must disconnect from the internet and work in degraded mode. There is no communication between the terminal port and the mIoT system.

This architecture integration generates an AG first for this use case scenario, as shown in Figure 4.6. 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 a 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, CVE-2017-0143; classtype:attempted-admin; sid:1000003; rev:1;). This solution

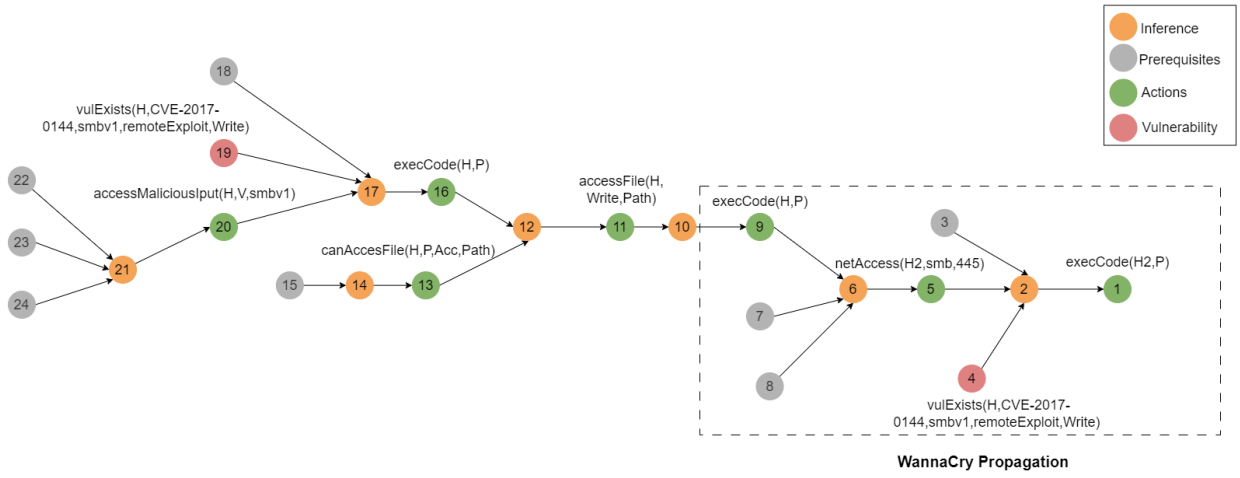


Figure 4.6 An AG generated for the use case scenario

correlates the alert data with the AG generated for the scenario. The alert is associated with Node 19, representing the EternalBlue vulnerability on the entry point. Then, the countermeasures selection process starts to get the associated countermeasures to block the attacker from launching the ransomware. Following is the list of countermeasures proposed by this 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.

4.7 Discussion and Conclusion

The countermeasures selected using this approach are compared with those proposed in the literature for ransomware attacks, particularly in the context of WannaCry use cases [63, 65–67]. In the literature, most countermeasures focus on prevention actions to avoid initial infection. These actions commonly are implementing spam filters to quarantine suspicious emails and attachments [66] and using predictive models to detect malicious behavior [65]. Other frequently suggested countermeasures are patching vulnerabilities [65], excluding kill-switch domains from firewall rules, and blocking SMB ports [67]. In opposition, this approach selects countermeasures from the prevention phase and containment, eradication, and recovery phases. For example, file removal is an eradication action aimed at eliminating ransomware from the system, followed by recovering encrypted files.

The countermeasures discussed in Section 4.6.3 are possible candidates for the IR plan. This work focuses entirely on selecting these countermeasures, which provide a foundation for playbook generation, a subject addressed in a separate contribution [45] presented in Chapter 5. With real-time generation, the AG helps quickly determine which countermeasure actions to apply based on the adversary's progress. Depending on the attacker's position, only specific countermeasures will be instantiated in the AG, leading to the generation of the ADG, which is presented in Chapter 6.

Future work should consider the physical impact of vulnerabilities in graph matching for countermeasure selection.

CHAPTER 5 OPTIMAL AUTOMATED PLAYBOOKS GENERATION

Based on the conference paper [45], presented at DBSec 2024, this chapter presents this approach to generate optimal IR playbooks automatically. As the complexity of cyberattacks escalates, it is difficult for analysts to resolve all received alerts in due time. In order to automate the IR process, many organizations integrate Security, Orchestration, Automation, and Response (SOAR) tools in their system. Nevertheless, incorporating these tools is complex, and organizations often do not receive their expected return on investment on time. This proposed approach aims to generate optimal playbooks automatically. Using the Pareto front [68], this approach enables the selection of the optimal playbooks for a vulnerability automatically. An optimal playbook is a playbook that helps get a good trade-off between conflicting objectives by maximizing its impact on the system and minimizing its loss as well as its complexity.

A playbook ontology is populated with the generated playbook instances. This work demonstrates how integrating the playbook ontology into SOAR tools can address the limitations of these kinds of tools. The experimentation is done for 40 vulnerabilities extracted from the NIST vulnerability database; they are chosen because their description includes information about either the method of exploitation or, their impact or both. This information is essential for the countermeasures selection process necessary for playbook generation. These experiments show that by adopting a multi-objective optimality approach, an average reduction of generated playbooks superior to 75% is achieved.

This chapter is organized as follows. Section 5.1, gives an overview of this approach to generate optimal playbooks for IR automatization. Section 5.2 explains the proposed IR playbook ontology. Section 5.3 presents this optimal IR playbook generation approach. In Section 5.4, the different steps of this approach are represented thanks to an illustrative example and evaluate the approach by comparing the gap between the number of playbooks generated before and after using Pareto and the time required for generating optimal playbooks. Section 5.5 discusses the results and presents a conclusion to this work.

5.1 Overview

The gray components in Figure 5.1 are involved in the optimal IR playbook generation. Figure 5.2 represents the process for generating optimal playbooks. This approach is based on the countermeasures selected from the KG matching process explained in chapter 4. Each

countermeasure of the countermeasures list obtained from the graph-matching process is correlated to a category and an IR stage of the RE&CT framework. Based on these categories and stages, the corresponding actions from RE&CT that should be involved in the playbook generation are obtained.

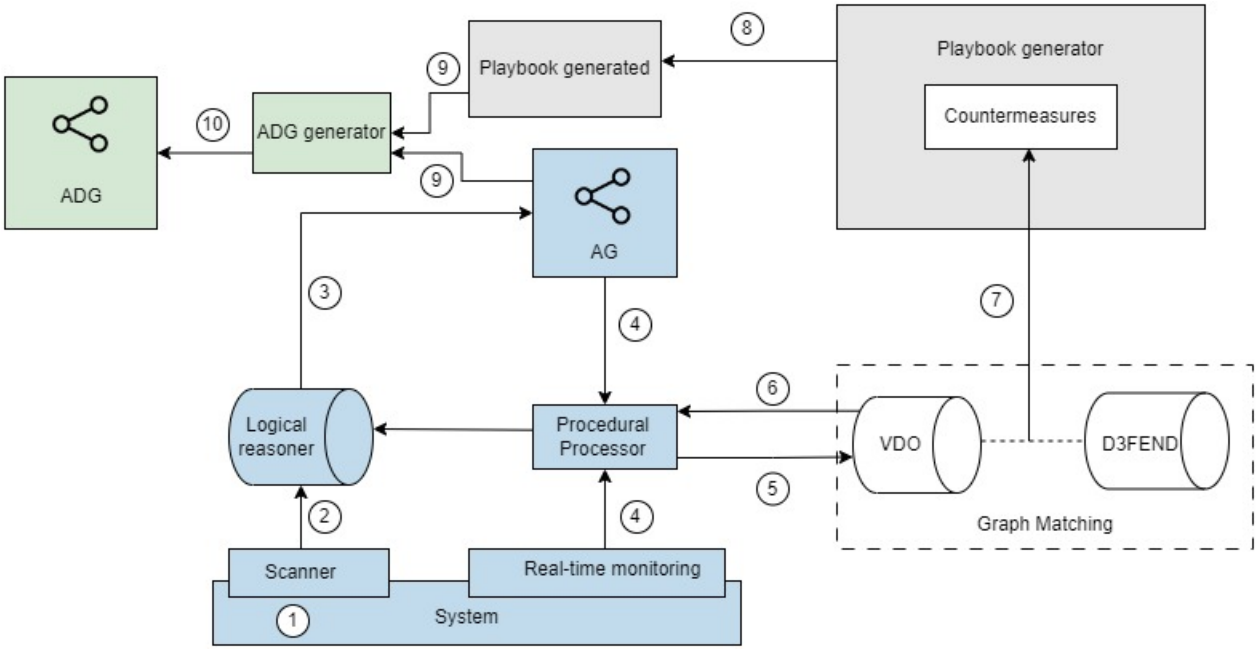


Figure 5.1 The ADG generation process

Thanks to this IR playbook ontology, the action attributes, such as the required tools for its execution, complexity, loss, and impact values, are obtained. The security tools necessary are matched with the system and attack knowledge to shorten this list of actions. From the reduced list of candidate actions, all the possible playbooks are generated by combining all the candidate actions, considering that each playbook contains more than two actions for each stage and the impact value sum of the playbook exceeds a fixed threshold. Pareto optimality, defined in Section 2.1 , is used on the generated playbooks to select an optimal playbook. The playbooks located on the Pareto Front are all optimal. Each of them is as good as the others. One of them can then be chosen to generate ontological individuals to populate this IR playbook ontology.

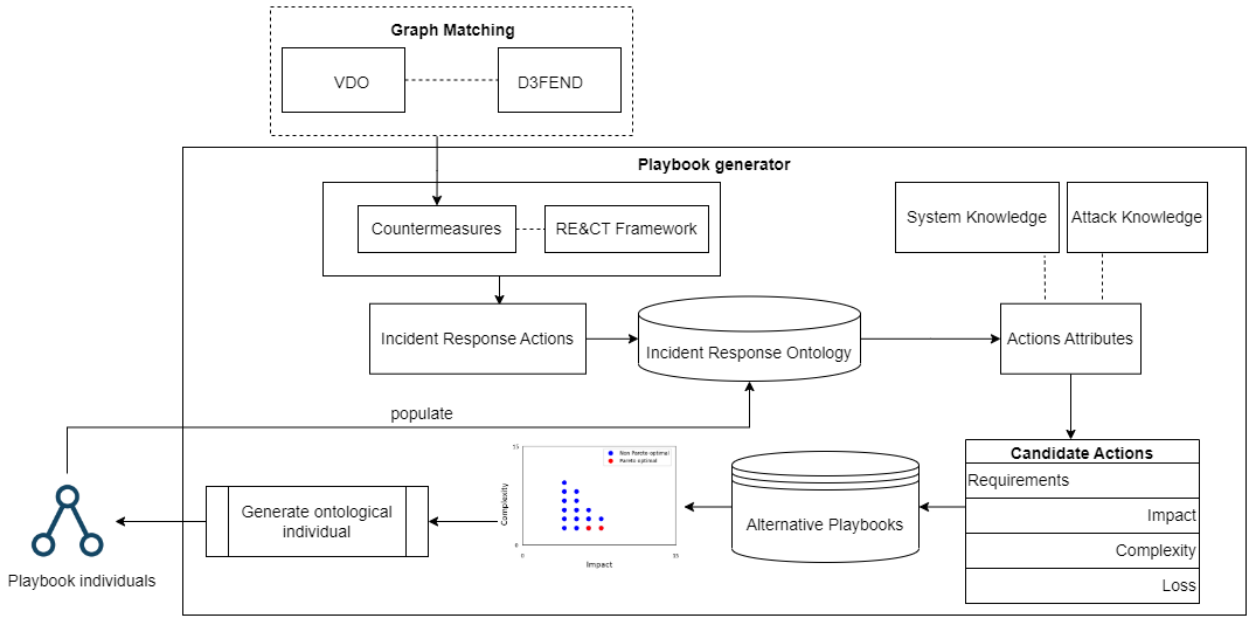


Figure 5.2 The optimal playbook generation process

5.2 IR Ontology

This section describes the IR playbook ontology[1] illustrated by Figure 5.3. The methodology called Simplified Agile Methodology for Ontology Development (SAMOD) [11] is used to develop this ontology. In addition, a step of the Abstractions Translation Ontology Method (ATOM) [12] is added after applying all the SAMOD steps consisting in: **1)** listing questions in natural language, **2)** listing concepts in questions, **3)** do a glossary to define each concept, **4)** traduce the questions into SPARQL query, **5)** add the concepts, the classes and properties defines in the former steps, to the ontology. At the end of the first SAMOD iteration, a step from the ATOM method is added to complete the ontology. This ATOM step consists of adding SWRL rules and descriptive logic to the ontology; this allows inference and obtaining an ontology with more knowledge of the playbooks. These steps are repeated until an ontology responds to the needs of the IR plan. Some needs of the IR plan are the IR stages as well as the required security tools for executing IR actions.

Classes that enable the formalization of IR playbooks while reusing some classes from existing ontologies are defined. However, they are not sufficient for modeling the playbook concept. Then, new classes are also created to associate them with the reused classes, allowing to represent better and formalize the IR playbook domain. The class **Playbook** is a super-class of the class **Playbook Step** that identifies the successive steps corresponding to IR actions;

[1] `https://github.com/phDimplKS/playbook_ontology`

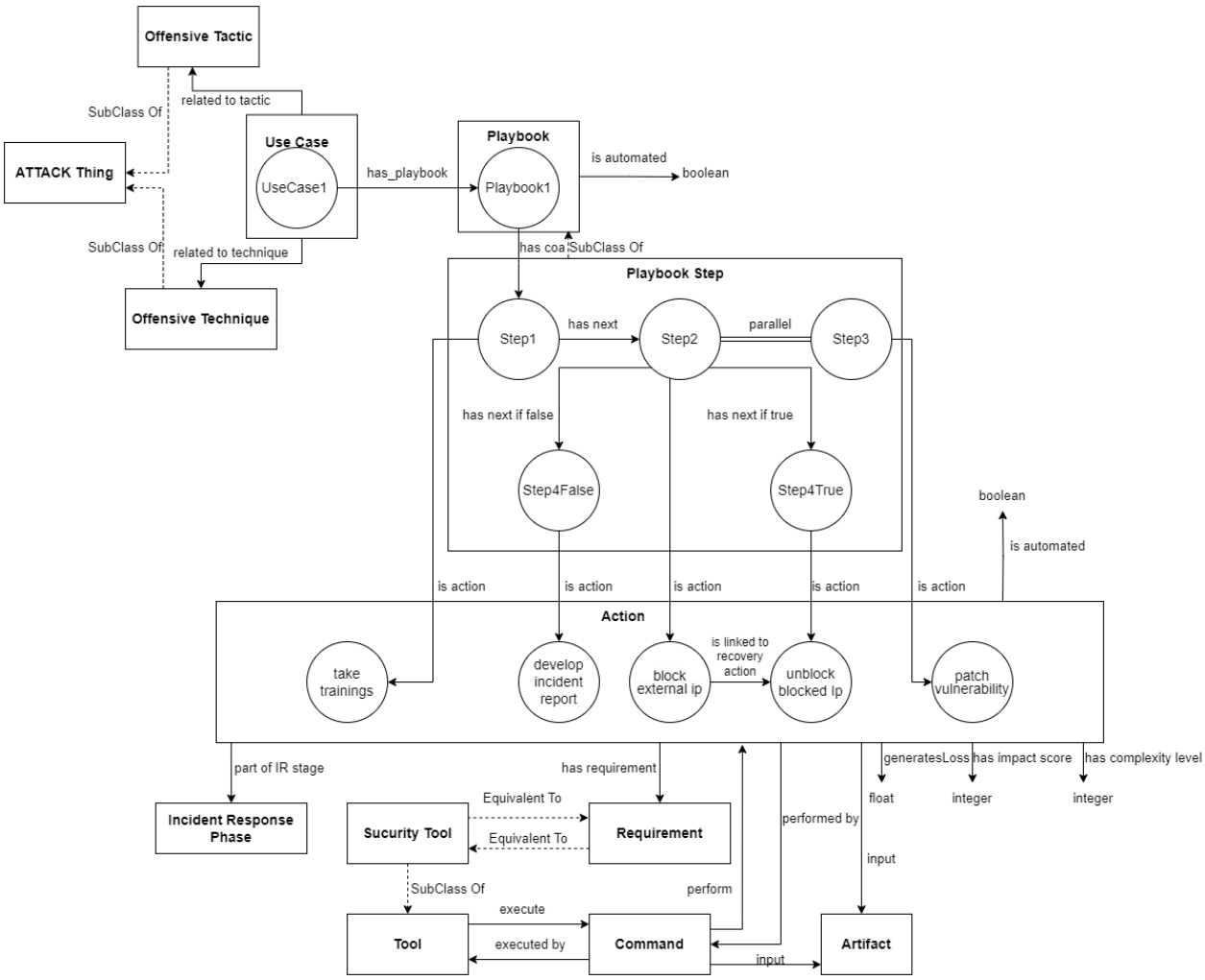


Figure 5.3 The incident response playbook ontology structure. The classes are represented in square form, and the circles represent the instances. The directed arrows represent the relationships between a class and another class, and an undirected arrow represents the relationship between a class and itself. The dashed arrows represent the property taken from RDF schema.

each individual of the playbook step class defines the position of its action in the playbook. The class **Action** from the UCO [40] ontology is taken. The classes **Tool** and **Security Tool**, taken from the existing ontologies UCO and SecOrp [38], represent assets involved in actions execution. The class **Requirement**, created based on the requirements linked to each action from RE&CT, is equivalent with **Security Tool**, which is a subclass of **Tool**. Data represented by the class **Artifact** can also release actions. These actions released allow to respond to attack tactics and techniques used by an attacker as described in the MITRE ATT&CK matrix; these concepts are represented by the class **ATTACK Thing**, which has the subclasses **Offensive Tactic** and **Offensive Technique**. The execution of an action

requires releasing commands. Each **Command** individual is linked to a security tool and an action.

The class **Use Case** concerns a specific use case monitored by an intrusion detection asset. In addition, ontology properties are created to express the relationships between the individuals of the different classes. Each containment action is linked to a recovery action thanks to the property **linked to recovery action**. There is no concept in an ontology that allows different action sequences. It is necessary to create properties that allow an order to be determined. The following properties are then created. The property **has coa** is directed to the first step of a playbook (*coa* signifies *Course of Action*). The property **has next** determines which playbook step should follow in the playbook workflow. Properties **has next if false** and **has next if true** allow representing the order of the actions when a condition exists. The property **parallel** expresses the fact that two actions can be executed at the same time.

Individuals should populate each class of the ontology. This ontology should allow enterprises to create a knowledge base of playbooks. This ontology is populated with individuals generated from two existing security knowledge bases: 1) the MITRE ATT&CK framework constitutes a base for representing attack concepts, and 2) the RE&CT framework is the base for expressing IR. The subclasses **Offensive Technique** and **Offensive Tactic** are instantiated using knowledge from MITRE ATT&CK. This population process is carried out semi-automatically. The class **IR Phase** is instantiated with the phases defined in RE&CT, while the RE&CT actions are used to instantiate the class **Action**. For the **Requirement** class, RE&CT proposes requirements for some actions, so the ontology is populated with them. However, no requirement is proposed for some actions; consulting security blogs, such as MITRE ATT&CK blog, is necessary to identify their requirements and populate the ontology. The classes **Playbook Step** and **Playbook** are populated with individuals created for the optimal playbooks generated automatically. The process of countermeasures selection is necessary for the playbook generation explained in Chapter 4.

The list of defensive techniques corresponds to the candidate countermeasures related to the vulnerability exploited. Each countermeasure is automatically linked with an IR phase and a category of the RE&CT framework. The candidate countermeasures selected serve as input to this optimal playbook generation solution, which will be explained in Section 5.3.

5.3 Optimal Playbooks Generation

An optimal automated playbook generation approach based on the Pareto front concept defined in Section 2.1 is proposed. As represented in Figure 5.2, the playbook generator first

generates a list of IR actions by correlating the candidate countermeasures with the RE&CT framework. This solution then queries the action attributes, such as the requirements for its release. The playbook generator verifies that the system contains at least one required security tool for the action execution. For example, the action *block internal ip address* execution requires an *internal firewall*; if the system knowledge base indicates an internal firewall exists in the enterprise, this solution keeps this action as a candidate for the playbook generation. In addition, it verifies the attacker's position on the attack knowledge base. For a remote attack, which means the attacker is external to the concerned system, it discards the action *block internal IP address*; it is no longer a candidate action for the playbook generation related to this scenario.

The candidate actions list is obtained after the automated pruning of the list of IR actions. The actions are ordered based on an increasing order for the phase and a decreasing order for the impact value. The playbook generator generates different combinations of actions from this list considering the defined conditions. A threshold value is set, requiring the sum of all actions' impact values to exceed it. A playbook should contain at least 2 actions distributed in the Identification, Preparation, Containment, and Eradication phases. This condition is established by requiring an action from the identification phase to detect and analyze an anomaly, followed by at least one action from the containment or eradication phase to mitigate the detected threat. Then, a playbook should be composed of actions of at least two different phases.

Additionally, a maximum number of actions contained in a playbook is fixed. This is done to reduce the execution time required to generate a playbook. After this, based on these experiments presented in Section 5.4.1, an average of more than 20 alternative playbooks for a vulnerability is obtained. Pareto optimality is used in this process to select the optimal option. From the KG of the IR ontology, the 3 quantitative attributes can be obtained to define the objectives for which a good trade-off is sought in selecting the optimal playbooks.

These parameters are the action's impact, complexity, and loss values. The impact value quantifies the impact of an action application on a system, ranging from 1 to 5, where 1 is the value assigned to an action for which the application does not have a significant impact, and 5 is the value assigned to the action with a high impact. The complexity value expresses the level of complexity of the release of an action on a system. The complexity value goes from 1 to 10; an action with a complexity value of 10 is very complex to execute. The loss value is calculated by summing an action's integrity, availability, and confidentiality loss; the value for these three parameters goes from 0 to 1; for an action that does not cause a confidential loss, for example, the confidentiality loss value is assigned a value of 0.

Some actions released can cause a loss in availability, for example, the action *block internal IP address*. For each RE&CT action, a value is assigned to these parameters based on knowledge from the literature review in cybersecurity.

These parameters define three objectives for optimizing this approach based on knowledge from the literature review. These objectives are: **a)** maximize the impact, **b)** minimize the complexity, and **c)** minimize the loss. In order to not compromise an objective at the expense of others, a playbook is selected thanks to the Pareto optimality application. All the playbooks that have no other playbooks dominating them are Pareto optimal. These playbooks dominate all the other playbooks, and they are incomparable.

Figure 5.4 is a 2D representation of Pareto optimal playbooks, considering the objectives of maximizing the impact and minimizing the complexity; a dot represents each playbook. The blue ones represent the dominated playbooks, and the red ones represent the Pareto optimal playbooks on the Pareto Front. Regarding optimization, the two red dots on the Pareto front are equal. Thanks to a query on the KG represented in Listing 6 of Section 5.4.1, the corresponding recovery action for each containment action in the optimal playbooks is added automatically to these playbooks. The actions of the lessons learned phase are also automatically added to the optimal playbooks. One of the optimal playbooks can be selected randomly. In Section 5.4.1, an illustrative example helps to represent an automated generated playbook, with actions of the recovery and lessons learned phases.

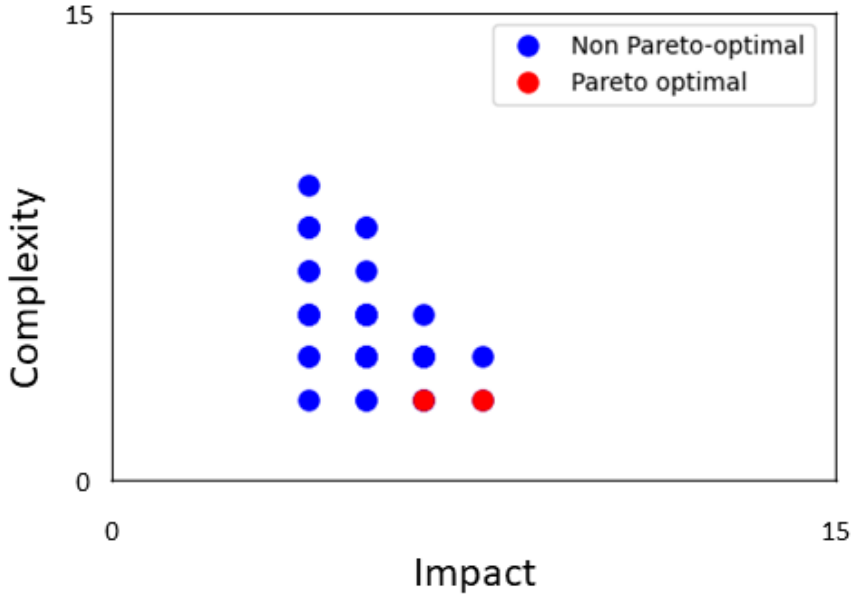


Figure 5.4 Pareto optimality for playbook generation

This approach automatically creates individuals for the chosen playbook destined to populate this IR ontology represented by Figure 5.3. This solution creates an instance for the playbook class. This instance is associated with the instance created for the first playbook step thanks to the property **has coa**. The other playbook step individuals are associated with the actions composing the playbook. This solution associates the different playbook steps thanks to the properties **has next** and **parallel**.

5.4 Results and Evaluation

This section presents the experimental implementation of this approach[2] based on an illustrative example and evaluates this approach for generated playbooks for 40 vulnerabilities.

5.4.1 Illustrative Example

A scenario is chosen to explain the implementation of this approach. The playbook generation process is illustrated based on the graph matching output and the system and attacker position information. The chosen scenario includes the exploitation of CVE-2021-21277. This vulnerability allows remote code execution and can lead to privilege escalation. VDO is semi-automatically instantiated with information about the vulnerability, such as the exploitation method, its impact, and the attacker position required for its exploitation.

VDO and D3FEND are automatically matched to generate the countermeasures list related to the vulnerability. Then, the automated playbook generation can proceed. The inputs consist of the list of selected countermeasures, the RE&CT framework matrix in JSON format, and JSON files containing information about the system and the concerned vulnerability. Figure 5.5 provides an overview of the network environment. The vulnerability existing on a web server is exploitable by a remote attacker. There is a terminal connected to the web server. Various software solutions exist on the terminal, such as an antivirus and a firewall. The same system has a Security Information and Event Management (SIEM) and an Intrusion Detection System (IDS).

Following is the list of countermeasures selected for CVE-2021-21277:

1. Some countermeasures target angular expressions: Software Update, Asset Vulnerability Enumeration, and Service Binary Verification.

2. Several countermeasures aim to prevent and control process execution: Process Code

[2] `https://github.com/phDimplKS/play`

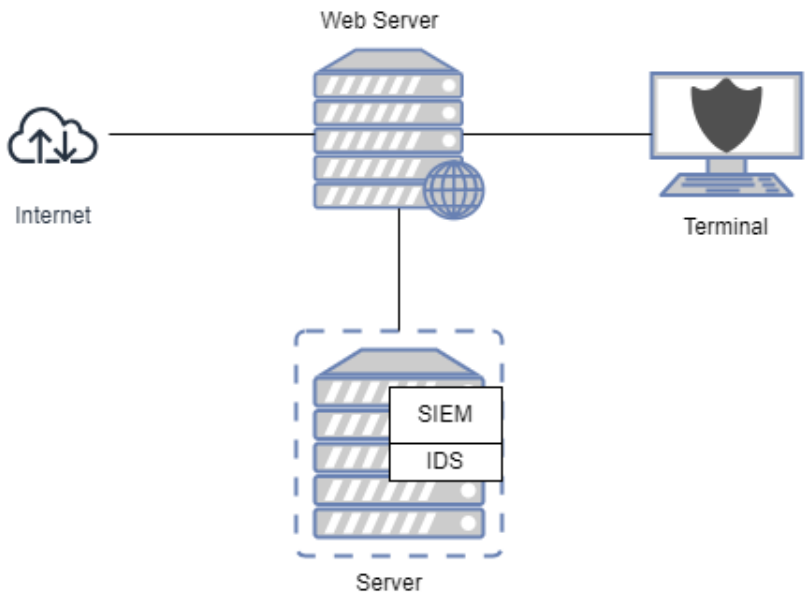


Figure 5.5 Example System

Segment Verification, Memory Boundary Tracking, Process Segment Execution Prevention, and Segment Address Offset Randomization.

3. Other aim to compare stored data with stack frame or shadow call stack: Stack Frame Canary Validation and Shadow Stack Comparisons.

These selected countermeasures are part of different IR stages and categories. Their involvement in the playbook generation will ensure a playbook has actions from several IR stages involving several security tools.

By automatically matching the RE&CT framework with the countermeasures, the playbook generator gets a list of 15 candidate actions depicted in Table 5.1.

Figure 5.6 represents an extract of the RE&CT framework for these 15 actions. The IR ontology is populated with all the RE&CT actions. There is a RE&CT ID for each action. Each action is linked with a phase of an IR stage. The required security tools are linked with each action. Each action has a value of complexity, impact, and loss generated.

The ontology is populated with information concerning how the action is executed, automatically or manually. A SPARQL query on this IR ontology, represented in Listing 5, retrieves the value of the loss, complexity, and impact for each action and the required tools for each action execution. By matching the software products installed on the assets from the example system with the required security tools for each action and considering that the vulnerability

Table 5.1 List of countermeasures for CVE-2021-21277 from graph matching

CVEID	Method or Impact	Countermeasures	Phase	Category
CVE-2021-21277	Code Execution	Software Update	Containment	General
CVE-2021-21277	Code Execution	Asset Vulnerability Enumeration	Identification	General
CVE-2021-21277	Code Execution	Service Binary Evaluation	Identification	General
CVE-2021-21277	Privilege Escalation	Process Code Segment Evaluation	Identification	Process
CVE-2021-21277	Privilege Escalation	Memory Boundary Tracking	Identification	Process
CVE-2021-21277	Privilege Escalation	Segment Address Offset Randomization	Containment	Process
CVE-2021-21277	Privilege Escalation	Process Segment Execution Prevention	Identification	Process
CVE-2021-21277	Privilege Escalation	Stack Frame Canary Validation	Identification	General
CVE-2021-21277	Privilege Escalation	Shadow Stack Comparisons	Identification	General
CVE-2021-21277	Privilege Escalation	Asset Vulnerability Enumeration	Identification	General

is exploited remotely, only 14 can be involved in the automated playbook generation process.

Table 5.2 represents the 14 actions involved in the playbook generation process with their parameter values. Considering the 14 actions, the playbook generator generates all the possible playbooks for CVE-2021-21277 with the constraints mentioned in section 5.3. Two conditions are considered for this experimentation based on the possible repetition of a phase in a playbook. Based on its available resources, an organization may face the constraint of executing a reduced number of actions in its IR plan, which defines the first condition. For the first condition, each phase is considered to be repeatable only once in a playbook.

	Phase					
	Prepa-ration	Identification	Containment	Eradi-cation	Reco-very	Les-sons lear-ned
General		list victims of security alert	Patch vulnerability			
General		list host vulnerabilities				
General		put compromised accounts on monitoring				
Process		list processes executed				
Process		find process by executable format	block process by executable format			
Process		find process by executable path	block process by executable path			
Process		find process by executable content pattern	block process by executable content pattern			
Process		find process by executable metadata	block process by executable metadata			
Process		find process by executable hash	block process by executable hash			

Figure 5.6 List of candidate actions extracted from RE&CT framework for CVE-2021-21277

For this condition, the playbook generator generates 28 playbooks. Considering that other organizations may have more resources available, they can execute more actions in an IR plan as long as these actions have a great impact on the organization's lifecycle; a second condition is defined. For the second condition, a phase can be repeated more than once, but a category can only be repeated once for a phase. In this case, the process generates 214 playbooks.

In both cases, it is necessary to select one playbook. The selected playbook should be an optimal one, meaning that this playbook must allow a good trade-off between the objectives consisting of maximizing the impact and minimizing loss and complexity. An algorithm to establish the Pareto front to select an optimal playbook is implemented. The playbooks that dominate the other playbooks and are not dominated by any playbook constitute the Pareto

```
PREFIX :http://playbook/IncidentResponseOntologyPlaybook#>
        rdfs:<http://www.w3.org/2000/01/rdf-schema#>
        select ?l ?impact ?loss ?complexity ?rl where{
            ?action :react-id '"""+id+"""' .
            ?action rdfs:label ?l .
            ?action :has_impact_score ?impact .
            ?action :generatesLoss ?loss .
            ?action :has_complexity_level ?complexity .
            ?action :has_requirement ?r .
            ?r rdfs:label ?rl .
        }
```

Listing 5: SPARQL query to get the parameters linked to a RE&CT action

front; they are optimal. For the first condition, considering a phase can only be repeated once, the algorithm allows getting 2 optimal playbooks. For the second condition, 6 optimal playbooks are obtained.

Listing 6 represents a SPARQL query that allows getting the recovery actions linked to the action of the containment phase in both cases. In both cases, after adding the actions of the recovery and lessons learned phases in the optimal playbooks, the playbook generator randomly selects 1 of the optimal playbooks.

```
PREFIX :http://playbook/IncidentResponseOntologyPlaybook#>
        rdfs:<http://www.w3.org/2000/01/rdf-schema#>
        select ?l where{
            ?action <http://www.w3.org/2000/01/rdf-schema#label>
            ↪  '"""+action+"""'.
            ?action <http://kevin.abouahmed/IncidentResponseOntologyPlaybook
            #is_linked_to_recovery_action> ?idrec .
            ?idrec <http://www.w3.org/2000/01/rdf-schema#label> ?l .
    }
```

Listing 6: SPARQL query to get the action of the recovery phase linked to an action of the containment phase

Figure 5.7 represents the optimal playbook selected for the first condition. The playbook starts with an action of the identification phase and ends with an action of the lessons learned phase. Figure 5.8 represents the optimal playbook selected for the second condition. In this playbook, two actions of the identification phase can be executed in parallel because they are from different categories; their execution does not need the same requirements. Two actions

Table 5.2 List of actions with their parameters' values for CVE-2021-21277

Action	Impact	Complexity	Loss	Phase	Category
find process by executable hash	5	4	0.3	2	4
find process by executable format	5	3	0.3	2	4
list victims of security alert	4	4	0.2	2	0
list processes executed	4	3	0.2	2	4
find process by executable metadata	4	4	0.3	2	4
put compromised acounts on monitoring	3	7	0.7	2	0
find process by executable path	3	3	0.3	2	4
find process by executable content pattern	3	5	0.3	2	4
patch vulnerability	4	3	0.0	3	0
block process by executable format	4	3	0.0	3	4
block process by executable path	3	4	0.0	3	4
block process by executable hash	3	3	0.0	3	4
block process by executable metadata	2	4	0.0	3	4
block process by executable content pattern	2	6	0.0	3	4

of the containment phase can also be carried out in parallel. However, notice that the action *Patch vulnerability* has higher priority than the action *Block process by executable format* because it has a higher impact value. Notice that there is a logical workflow between the actions in both conditions. For example, after the SOAR executes the action *Find process in*

executable format, it should execute the action *Block process in executable format*. Then, in the recovery phase, it should carry out the action *Unblock blocked process*.

Playbook when each phase can be repeated only once

Step	React ID	Action	Is automated	Tool
Step 1	RA2405	Find process by executable format	True	Powershell
Step 2	RA3404	Block process by executable format	True	Group Policy
Step 3	RA5401	Unblock blocked process	True	Group Policy
Step 4	RA6001	Develop incident report	False	NA
Step 5	RA6002	Conduct lessons learned exercise	False	NA

Figure 5.7 Optimal playbook selected when each phase can be repeated only once

Playbook when a phase can be repeated, but a category only repeats once per phase

Step	React ID	Action	Is automated	Tool
Step 1	RA2405	Find process by executable format	True	Powershell
Step 2	RA2001	List victims of security alerts	True	Suricata
Step 3	RA3001	Patch vulnerability	True	WSUS
Step 4	RA3404	Block process by executable format	True	Group Policy
Step 5	RA5401	Unblock blocked process	True	Group Policy
Step 6	RA6001	Develop incident report	False	NA
Step 7	RA6002	Conduct lessons learned exercise	False	NA

Figure 5.8 Optimal playbook selected when a category is repeated once for a phase

Afterward, the playbook generator automatically generates the ontological instance for the classes playbook and playbook step dedicated to the ontology population. The instances are generated automatically to populate the ontology. Terse RDF Triple Language (Turtle) is a syntax and file format for expressing data in the Resource Description Framework (RDF) data model. In turtle, the instances created for CVE-2021-21277 when a category is repeated once for a phase are shown in Listing 7.

```
IncidentResponseOntologyPlaybook:Playbook__CVE-2021-21277 a
↪  IncidentResponseOntologyPlaybook:Playbook,
        owl:NamedIndividual ;
    rdfs:label "Playbook : Response to CVE-2021-21277" ;
    IncidentResponseOntologyPlaybook:has_coa
    ↪  IncidentResponseOntologyPlaybook:Step1_CVE-2021-21277 ;
    IncidentResponseOntologyPlaybook:has_description "This playbook is intended to answer
    ↪  the vulnerability identified by the IDCVE-2021-21277" .
IncidentResponseOntologyPlaybook:Step1_CVE-2021-21277 a
↪  IncidentResponseOntologyPlaybook:Playbook_Step,
        owl:NamedIndividual ;
    rdfs:label "Step1_CVE-2021-21277",
        "Step2_CVE-2021-21277" ;
    IncidentResponseOntologyPlaybook:is_action IncidentResponseOntologyPlaybook:RA2405 ;
    IncidentResponseOntologyPlaybook:parallel
    ↪  IncidentResponseOntologyPlaybook:Step2_CVE-2021-21277 .
IncidentResponseOntologyPlaybook:Step2_CVE-2021-21277 a
↪  IncidentResponseOntologyPlaybook:Playbook_Step,
        owl:NamedIndividual ;
    rdfs:label "Step2_CVE-2021-21277" ;
    IncidentResponseOntologyPlaybook:has_next
    ↪  IncidentResponseOntologyPlaybook:Step3_CVE-2021-21277 ;
    IncidentResponseOntologyPlaybook:is_action IncidentResponseOntologyPlaybook:RA2001 .
IncidentResponseOntologyPlaybook:Step3_CVE-2021-21277 a
↪  IncidentResponseOntologyPlaybook:Playbook_Step,
        owl:NamedIndividual ;
    rdfs:label "Step3_CVE-2021-21277" ;
    IncidentResponseOntologyPlaybook:is_action IncidentResponseOntologyPlaybook:RA3001 ;
    IncidentResponseOntologyPlaybook:parallel
    ↪  IncidentResponseOntologyPlaybook:Step4_CVE-2021-21277 .
IncidentResponseOntologyPlaybook:Step4_CVE-2021-21277 a
↪  IncidentResponseOntologyPlaybook:Playbook_Step,
        owl:NamedIndividual ;
    rdfs:label "Step4_CVE-2021-21277" ;
    IncidentResponseOntologyPlaybook:has_next
    ↪  IncidentResponseOntologyPlaybook:Step5_CVE-2021-21277 ;
    IncidentResponseOntologyPlaybook:is_action IncidentResponseOntologyPlaybook:RA3404 .
IncidentResponseOntologyPlaybook:Step5_CVE-2021-21277 a
↪  IncidentResponseOntologyPlaybook:Playbook_Step,
        owl:NamedIndividual ;
    rdfs:label "Step5_CVE-2021-21277" ;
    IncidentResponseOntologyPlaybook:has_next
    ↪  IncidentResponseOntologyPlaybook:Step6_CVE-2021-21277 ;
    IncidentResponseOntologyPlaybook:is_action IncidentResponseOntologyPlaybook:RA5401 .
IncidentResponseOntologyPlaybook:Step6_CVE-2021-21277 a
↪  IncidentResponseOntologyPlaybook:Playbook_Step,
        owl:NamedIndividual ;
    rdfs:label "Step6_CVE-2021-21277" ;
    IncidentResponseOntologyPlaybook:has_next
    ↪  IncidentResponseOntologyPlaybook:Step7_CVE-2021-21277 ;
    IncidentResponseOntologyPlaybook:is_action IncidentResponseOntologyPlaybook:RA6001 .
IncidentResponseOntologyPlaybook:Step7_CVE-2021-21277 a
↪  IncidentResponseOntologyPlaybook:Playbook_Step,
        owl:NamedIndividual ;
    rdfs:label "Step7_CVE-2021-21277" ;
    IncidentResponseOntologyPlaybook:is_action IncidentResponseOntologyPlaybook:RA6002 .
```

Listing 7: The individuals added into the IR playbook ontology for CVE-2021-21277 when a category can be repeated once for a phase

5.4.2 Evaluation

This approach is evaluated over 40 CVEs for the 2 different conditions introduced in Section 5.4.1: **First condition** considering that each phase can only be repeated once in a playbook and **Second condition** considering that a phase can be repeated more than once but that a category can be repeated only once for a phase. This evaluation is based on the following metrics:

- The gap between the number of playbooks generated before applying Pareto optimality represented by n and after Pareto optimality represented by m is calculated in percentage using this formula: $(n - m)/n * 100$
- The execution time of this implementation for generating playbooks and selecting the optimal one for each CVE in seconds is calculated.

The playbook generator is run twice to ensure the execution time is correct; the average execution time of the first and second iteration for each CVE is calculated for both conditions mentioned above. The gap calculation for each CVE for the two conditions is processed, then the average for all 40 CVEs is calculated. Table 5.3 compares the average values obtained for the different metrics considering both conditions. In both cases, the Pareto front helps to reduce considerably the number of playbooks generated; the gap is over 75%. Even if the generation time is more elevated for the second condition, this does not impact the model's performance much because the number of playbooks generated before applying Pareto is more elevated for the second condition than the first condition.

Table 5.3 Evaluation of this approach

	No Playbooks before Pareto	**No Playbooks after Pareto**	**First metric: Gap in %**	**Second metric: Time in s**
First condition	26.5	2.1	78.59	7.07
Second condition	8417.5	10.175	82.22	13.01

5.4.3 Other Optimal Playbooks

Playbook for CVE-2017-0144

This section presents the playbook generated for the EternalBlue vulnerability identified for the use-case scenario proposed in Section 4.6.3, considering that a category can be repeated once for a phase.

Launching the countermeasures selection process presented in Chapter 4 leads to selecting countermeasures related to different types of artifacts, such as files, executable files, and network traffic. Below is a list of some selected countermeasures follows:

1. Some countermeasures related to the artifact file are: File Encryption, Local File Permissions, File Removal, and Restore File.
2. Some countermeasures are linked to the artifact executable file: Executable Allowlisting and Executable Denylisting.
3. Some countermeasures linked to the artifact network traffic are: Network Traffic Filtering, DNS Traffic Analysis, and DNS Allowlisting.

The selected countermeasures are mapped to RE&CT to get the list of candidate IR actions for the optimal IR playbook generation. Table 5.4 represents the mapping between the countermeasures linked to the artifact executable file and the phases and categories of RE&CT framework. The countermeasures cover several stages of the IR plan: identification, containment, and recovery. Based on the security tool in the infrastructure, such as an internal firewall, a SIEM, WireShark, an external firewall, and the required location of the adversary for the vulnerability exploitation, the playbook generator starts generating an optimal IR playbook for the EternalBlue vulnerability.

Figure 5.9 represents the optimal playbook generated for EternalBlue, considering that the infrastructure is formed of these security tools: WSUS, an internal firewall, a SIEM, WireShark, an external firewall, Group Policy, an antivirus, Powershell, Suricata, and Firewall Software. Actions from the same phase but from different categories can be released in parallel because their execution does not require the same security tools. However, the two actions from the Lessons learned should be carried out successively as it is necessary to have the incident report in order to conduct the lessons learned exercise.

Table 5.4 The countermeasures related to the artifact executable file for CVE-2017-0144

CVEID	Method or Impact	Context	Artifact	Countermeasure	Phase	Category
CVE-2017-0144	Manipulation	Host Operating System	Executable File	Dynamic Analysis	Identification	File
CVE-2017-0144	Manipulation	Host Operating System	Executable File	Emulated File Analysis	Identification	File
CVE-2017-0144	Manipulation	Host Operating System	Executable File	Executable Allowlisting	Containment	File
CVE-2017-0144	Manipulation	Host Operating System	Executable File	Executable Denylisting	Containment	File
CVE-2017-0144	Manipulation	Host Operating System	Executable File	File Encryption	Containment	File
CVE-2017-0144	Manipulation	Host Operating System	Executable File	Local File Permissions	Identification	File
CVE-2017-0144	Manipulation	Host Operating System	Executable File	Decoy File	Identification	File
CVE-2017-0144	Manipulation	Host Operating System	Executable File	File Analysis	Identification	File
CVE-2017-0144	Manipulation	Host Operating System	Executable File	File Removal	Eradication	File
CVE-2017-0144	Manipulation	Host Operating System	Executable File	File Integrity Monitoring	Identification	File
CVE-2017-0144	Manipulation	Host Operating System	Executable File	Restore File	Recovery	File

Playbook for CVE-2019-0708

Here, the concept of default playbook is introduced, and the default playbook is generated for the use case scenario proposed in Section 3.6.1.

A default playbook is generated when the information concerning the available security tools in the infrastructure and the vulnerability exploitation requirements are not enough to generate an optimal playbook. This approach then generates a best-effort playbook, which can be released on any infrastructure to block the adversary from causing more damage in the system, no matter the requirements for vulnerability exploitation.

Figure 5.10 represents the default playbook generated when the playbook generator cannot

Playbook when a phase can be repeated, but a category only repeats once per phase

Step	React ID	Action	Is automated	Tool
Step 1	RA2309	Find file by format	True	Suricata
Step 2	RA2107	List host communicated by port	True	Nmap
Step 3	RA3109	Block user external communication	True	External Firewall
Step 4	RA3301	Quarantine file by format	True	Antivirus
Step 5	RA4301	Remove file	True	Powershell
Step 6	RA5105	Unblock blocked user	True	External Firewall
Step 7	RA5301	Restore quarantined file	True	Powershell
Step 8	RA6001	Develop incident report	False	NA
Step 9	RA6002	Conduct lessons learned exercise	False	NA

Figure 5.9 Optimal playbook generated for the exploitation of EternalBlue in the use-case scenario introduced in Section 4.6.3

generate an optimal playbook from the available information. The actions proposed aimed to patch the vulnerability on the machine. The instances created in Turtle for the default playbook look are shown in Listing 8.

Default playbook generated for CVE-2019-0708

Step	React ID	Action	Is automated	Tool
Step 1	RA2002	List host vulnerabilities	True	Nessus
Step 2	RA3001	Patch vulnerability	True	WSUS
Step 3	RA6001	Develop incident report	False	NA
Step 4	RA6002	Conduct lessons learned exercise	False	NA

Figure 5.10 Default playbook generated for BlueKeep

```
IncidentResponseOntologyPlaybook:Playbook_CVE-2019-0708 a
↪  IncidentResponseOntologyPlaybook:Playbook,
        owl:NamedIndividual ;
    rdfs:label "Playbook : Response to CVE-2019-0708" ;
    IncidentResponseOntologyPlaybook:has_coa
    ↪  IncidentResponseOntologyPlaybook:Step1_CVE-2019-0708 ;
    IncidentResponseOntologyPlaybook:has_description "This playbook is intended to answer
    ↪  the vulnerability identified by the ID CVE-2019-0708" .
IncidentResponseOntologyPlaybook:Step1_CVE-2019-0708 a
↪  IncidentResponseOntologyPlaybook:Playbook_Step,
        owl:NamedIndividual ;
    rdfs:label "Step1_CVE-2019-0708" ;
    IncidentResponseOntologyPlaybook:has_next
    ↪  IncidentResponseOntologyPlaybook:Step2_CVE-2019-0708 ;
    IncidentResponseOntologyPlaybook:is_action IncidentResponseOntologyPlaybook:RA2002 .
IncidentResponseOntologyPlaybook:Step2_CVE-2019-0708 a
↪  IncidentResponseOntologyPlaybook:Playbook_Step,
        owl:NamedIndividual ;
    rdfs:label "Step2_CVE-2019-0708" ;
    IncidentResponseOntologyPlaybook:has_next
    ↪  IncidentResponseOntologyPlaybook:Step3_CVE-2019-0708 ;
    IncidentResponseOntologyPlaybook:is_action IncidentResponseOntologyPlaybook:RA3001 .
IncidentResponseOntologyPlaybook:Step3_CVE-2019-0708 a
↪  IncidentResponseOntologyPlaybook:Playbook_Step,
        owl:NamedIndividual ;
    rdfs:label "Step3_CVE-2019-0708" ;
    IncidentResponseOntologyPlaybook:has_next
    ↪  IncidentResponseOntologyPlaybook:Step4_CVE-2019-0708 ;
    IncidentResponseOntologyPlaybook:is_action IncidentResponseOntologyPlaybook:RA6001 .
IncidentResponseOntologyPlaybook:Step4_CVE-2019-0708 a
↪  IncidentResponseOntologyPlaybook:Playbook_Step,
        owl:NamedIndividual ;
    rdfs:label "Step4_CVE-2019-0708" ;
    IncidentResponseOntologyPlaybook:is_action IncidentResponseOntologyPlaybook:RA6002 .
```

Listing 8: The individuals added into the IR playbook ontology for CVE-2019-0708 correspond to the default playbook generated when the requirements are not met to generate an optimal playbook

5.5 Discussion and Conclusion

This approach is based on the results of countermeasures selection based on KG matching presented in Chapter 4. The selected countermeasures are matched with the IR actions of the RE&CT framework. Based on the system and attack characteristics, this proposed solution prunes the actions by matching the countermeasures with RE&CT. This solution generates all the possible combinations of the reduced list of actions, taking into account some defined conditions, such as the maximum number of actions in the playbook, the minimum number of IR phases that should be involved, and the total value of the impact of a playbook. Considering the optimality objectives of minimizing loss, complexity, and impact, this solution finds the Pareto optimal playbooks. The playbook generator selects one randomly, and the playbook and playbook step instances are generated based on the workflow step from the CACAO framework.

This approach contributes to resolving the integration problem in the SOAR using an ontology. The use of an ontology makes this proposed solution scalable and reusable. It is easier to populate a knowledge base communicating with a SOAR than to manage multiple playbook formats. This playbook generation approach is optimal. Each action has a specific value regarding loss, impact, and complexity; the Pareto front ensures a perfect trade-off regarding the optimality objectives when selecting a playbook. These parameters are assigned wisely and logically based on the literature review knowledge because they impact the playbook selection. Thanks to all the knowledge bases involved in this approach, the actions included in a playbook compose a logical workflow. As long as the playbook generator is run for a specific attack scenario and a specific system whose composition does not change, the selected playbook is optimal and can respond efficiently. In Chapter 6, this work focuses on how the generated playbook actions can be instantiated on an AG to get an ADG.

Integrating the proposed IR playbook ontology within a SOAR can be considered a future direction for the research.

CHAPTER 6 ATTACK-DEFENSE GRAPH GENERATION

This chapter, based on the conference paper [69] presented at TrustCom 2024, presents this approach to automatically generate an Attack-Defense graph in real-time by instantiating mitigation actions on AG. An expert can analyze each possible action of an attacker and their impact thanks to an AG to decide which mitigation actions should be applied to block the attack fulfillment. This chapter proposes generating real-time ADGs by instantiating IR actions on a LAG. The system's real-time monitoring enables malicious action detection, leading to alert generation. The generated alerts are correlated with the AG to determine the attacker's position within the system. This solution can identify optimal points to implement IR actions, minimizing the attack's impact. These IR actions are derived from an automatically generated optimal playbook tailored to the attack. This work proposes associating IR actions to the fact nodes of the AG to determine which actions should be instantiated on the graph. Hence, this work proposes generating predicates for the mitigation actions and aligning them with the AG predicates. This approach is validated for three use-case scenarios focused on critical industrial infrastructures. The countermeasures implemented on the AGs for the scenarios that can achieve the attack goal are examined. The approach is assessed based on the security relevance of the instantiated countermeasures on the AGs for various attack paths. The approach's time performance across various scenarios within the use cases is also evaluated.

This section presents this proposed methodology for real-time ADG generation. First, an overview of this process is presented in Section 6.1. Afterward, the generation of countermeasure predicates from IR actions in the proposed approach is explained in Section 6.2. Section 6.3 explains how countermeasure predicates are matched with AG predicates. Then, the proposed algorithm for countermeasure predicates instantiation on AGs, leading to an ADG as output, is explained in Section 6.4. Section 6.5 focuses on analyzing the complexity of the ADG generation process. The implementation of the approach is presented in Section 6.6. Section 6.7 presents the experimental setup of the ADG generation approach for 2 industrial infrastructures. Section 6.8 describes 3 use-case scenarios. Section 6.9 presents the experimental results for the 3 scenarios. Section 6.10 evaluates the security relevance of the countermeasure nodes instantiated. Section 6.11 presents the time performance evaluation of the approach. A discussion and conclusion for this chapter is provided in Section 6.12.

6.1 Overview

Figure 6.1 gives an overview of this approach for ADG generation. The output of an initial system scan serves as input to a logical reasoner for AG generation. In parallel, real-time system monitoring occurs with a Security Information and Event Management (SIEM) tool. This ADG generator correlates each generated alert information with the AG information. If an alert maps a node in the AG, the ADG generator queries an IR playbook ontology to get the IR playbook linked to the exploited vulnerability. Therefore, the ADG generator elects which actions in the playbook should be instantiated on the AG and on which nodes of the AG.

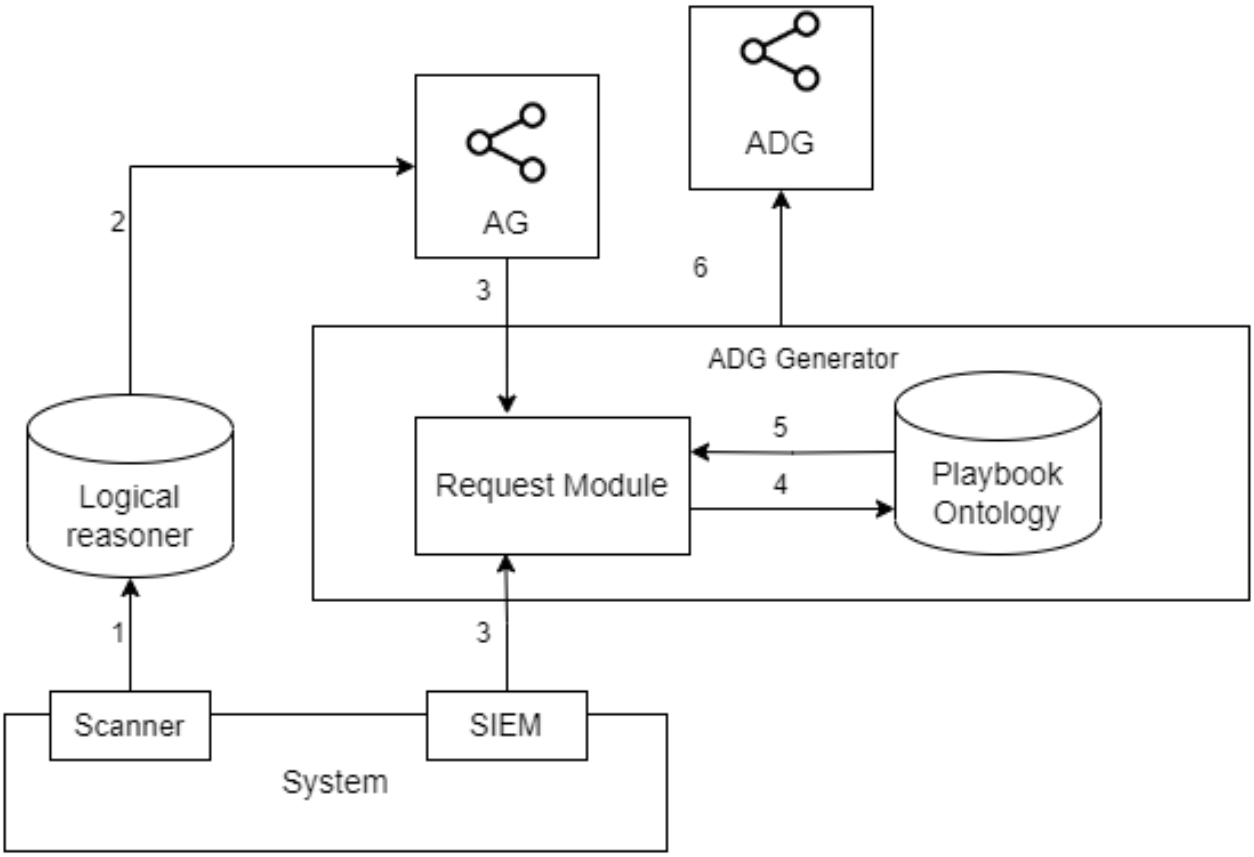


Figure 6.1 The ADG generation process

6.2 Countermeasure Predicates Generation

This section presents the methodology for generating countermeasure predicates from IR actions. A predicate is defined and illustrated in Section 2.1. To generate the countermea-

sure predicates, a POS tagging model is trained using some RE&CT framework[1] IR actions involved in the playbook generation process. POS tagging, a Natural Language Processing (NLP) technique, consists of labeling each word in a sentence with its corresponding part of speech, such as a noun, a verb, or an adjective. Table 6.1 introduces the main POS categories tags with their explanation.

Table 6.1 The core POS categories tags

Open class words	**Closed class words**	**Other**
ADJ: adjective	ADP: adposition	PUNCT: punctuation
ADV: adverb	AUX : auxiliary	SYM: symbol
INTJ: interjection	CCONJ: coordinating conjunction	X: other
NOUN: noun	DET: determiner	
PROPN: proper noun	NUM: numeral	
VERB: verb	PART: particle	
	PRON: pronoun	
	SCONJ: subordinating conjunction	

Based on the trained POS tagging model, this solution generates the logical predicates for all the IR actions queried from the IR playbook KG. The predicates start with a lowercase word tagged as VERB or ADP (see Table 6.1). The other words subsequent in the predicates start with uppercase. All actions starting with the expression *Get ability* of the Preparation stage of the IR playbook are discarded because those actions are not executable on a system. Nevertheless, the Identification, Containment, Eradication, Recovery, and Lessons Learned actions are part of the countermeasure predicates generation process. This solution creates a countermeasure predicates table with each IR action RE&CT ID, label, comment, and generated predicate.

6.3 Attack and Countermeasure Predicates Matching

An analogy approach is first proposed to link each AG predicate with one or more countermeasure predicates. The predicates of countermeasures are matched with the AG predicates using several conditions. For example, *patchVulnerability* is correlated with *vulExists* based on the prefix *vul* in both predicates. The predicates *networkServiceInfo* and *hasAccount* are matched to countermeasure predicates with some words in the AG predicates' parameters. The predicate *accessFile* is mapped with all countermeasures predicates containing the word

[1]https://atc-project.github.io/atc-react/

Process. An excerpt of the analogy is demonstrated in Figure 6.2.

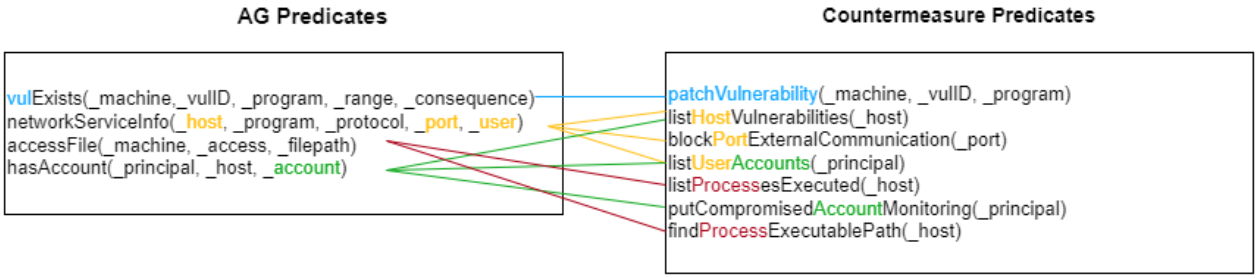


Figure 6.2 The analogy approach to match AG predicates with countermeasure predicates

The cosine similarity between each AG predicate and the analogous countermeasure predicates is then calculated, based on the AG predicate description and the IR action description from RE&CT, to identify the most similar countermeasure predicates matched with an AG predicate. The AG predicate description is taken from the works on the logical reasoner [70, 71], as well as from the logical reasoner documentation on GitHub[2].

Word2Vec is an NLP technique that helps obtain vector representations of words. A Word2Vec model trained on all the descriptions converts the descriptive text for AG and countermeasure predicates into a vector. Then, the cosine similarity is calculated between an AG description vector and its analogous countermeasure description vector.

The cosine similarity is a metric that measures 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; their cosine similarity can be measured using Formula 6.1.

$$\cos(AB) = \frac{A.B}{||A||||B||} \tag{6.1}$$

6.4 Countermeasures Instantiation on AG

This section presents the proposed countermeasures instantiation on the AG process based on anti-correlation, defined in Section 2.1. Depending on the severity of an alert, it can impact several nodes of the AG. A countermeasure for an attacker's action aims to contradict the action precondition that maps the alert. However, when all the preconditions of the attacker's action map the alert, the countermeasures should aim to contravene the action itself to

[2]https://github.com/risksense/mulval.git

prevent the adversary from advancing toward the attack goal. If the attacker is already on the attack's final goal, it is too late to react, so no mitigation action is introduced.

Algorithm 2 illustrates the logic behind this process. An AG is generated from an infrastructure network information file. The solution generates a flow matrix, a mathematical representation describing the flow of entities between different nodes within a network or system, for the AG. Each matrix cell represents the flow from one node to another. The flow value takes either a 0 or a 1. A cell with a 0 represents that no intrusion has been detected, and a 1 represents a vulnerability, the adversary's location, or intrusion detection. The flow value from a node representing a vulnerability existence and an adversary location is always at 1 when the AG is generated at first. The flow value from other nodes goes to 1 when an alert is detected concerning them.

Algorithm 2: Algorithm of Countermeasures Instantiation on Attack Graph

$G = (V; E)$;
$\Gamma^{+}(j) \leftarrow$ the set of successors of a node j;
$\Gamma^{-}(j) \leftarrow$ the set of predecessors of a node j;
$flowmatrix \leftarrow \begin{vmatrix} f(1,1) & .. & f(1,j) \\ .. & .. & .. \\ f(j,1) & .. & f(j,j) \end{vmatrix}$;
if *new_alert* **then**
  for $\forall j \in V$ *with* $\Gamma^{-}(j) \neq \emptyset$ **do**
    if $\sum_{k \in \Gamma^{-}(j)} f_t(k,j) = |\Gamma^{-}(j)|$ **then**
      Apply countermeasures on $\forall l \in \Gamma^{+}(j)$ as $\Gamma^{+}(l) \neq \emptyset$ and node type is not AND;
      //Identify countermeasures from countermeasure predicates table Update $flow_matrix$;
      //$f_t(j,l) = 0$ and $f_t(k,j) = 0$ if k does not identify vulnerability existence
    if $\sum_{k \in \Gamma^{-}(j)} f_t(k,j) < |\Gamma^{-}(j)|$ **then**
      Apply countermeasures on $\forall k \in \Gamma^{-}(j)$ as $f_t(k,j) = 1$ and node type is not AND;
      //Identify countermeasures from countermeasure predicates table Update $flow_matrix$;
      //$f_t(k,j)$=0 if k does not identify vulnerability existence

j is a node of the graph. The set of predecessors is illustrated as $\Gamma^{-}(j)$. Each predecessor of j from this set is represented by k. $\Gamma^{+}(j)$ represents the set of successors of j in which each successor is marked as l.

When a new alert is detected, the algorithm traverses the AG to verify if the detected alert concerns a node j with at least one predecessor. Suppose the sum of flow value going to j from

all nodes belonging to the predecessors set $\Gamma^{-}(j)$ equals the cardinality of the predecessors of j. In that case, countermeasures should be applied to each successor l from the successors set $\Gamma^{+}(j)$ of j as l has at least one successor. The node type of l should not be AND (see Section 2.1. Then, the flow matrix should be updated, putting all the flow values $f_t(j, l)$ and $f_t(k, j)$ to 0.

Supposing that the sum of flow value going to j from all nodes belonging to the predecessors set $\Gamma^{-}(j)$ of j is less than the cardinality of the predecessors of j. Then, the countermeasures should be applied simultaneously to each predecessor k of j as $f_t(k, j) = 1$, and the node type should not be AND. The algorithm updates the flow matrix, putting all the flow values $f_t(k, j)$ to 0.

When instantiating the countermeasures on the AG, the algorithm considers the possible negative impact an IR action execution could have on the system. Some countermeasures may affect others, which are referred to as conflicting countermeasures. For example, a recovery action should not be applied until all containment and eradication actions are applied, except if the vulnerabilities that open a breach to the attack have been patched.

Table 6.2 provides an extract from the predicates table, created by advantaging the analogy between AG predicates and countermeasure predicates. This analogy is obtained due to part-of-speech tagging on IR actions and the cosine similarity calculation between AG predicates and their corresponding IR action predicates.

Table 6.2 An extract of the predicates table generated by correlating AG predicates with countermeasure predicates

Countermeasure Predicates	**AG Predicates**
findProcessExecutableFormat(_host)	netAccess(_machine, _protocol, _port)
blockProcessExecutableFormat(_host)	netAccess(_machine, _protocol, _port)
listHostsCommunicatedPort(_port)	networkServiceInfo(_host, _program, _protocol, _port, _user)
blockUserExternalCommunication(_principal)	networkServiceInfo(_host, _program, _protocol, _port, _user)
blockProcessExecutableFormat(_host)	execCode(_host, _user)
findProcessExecutableFormat(_host)	execCode(_host, _user)
listHostsCommunicatedPort(_port)	hacl(_src, _dst, _prot, _port)
listHostsCommunicatedPort(_port)	hasAccount(_principal, _host, _account)

Figure 6.3 shows an extract of an AG generated for a multi-stage attack, indicating that the attack involves multiple sub-goals. The sub-goal $execCode(h, u)$ is an action that can be

inferred by the logical reasoner using the following interaction rule (Rule 1):

```
execCode(_host, _user) :-
    networkServiceInfo(_host, _program, _protocol, _port, _user),
    vulExists(_host, _vulID, _program, _range, _consequence),
    netAccess(H, Protocol, Port).
```

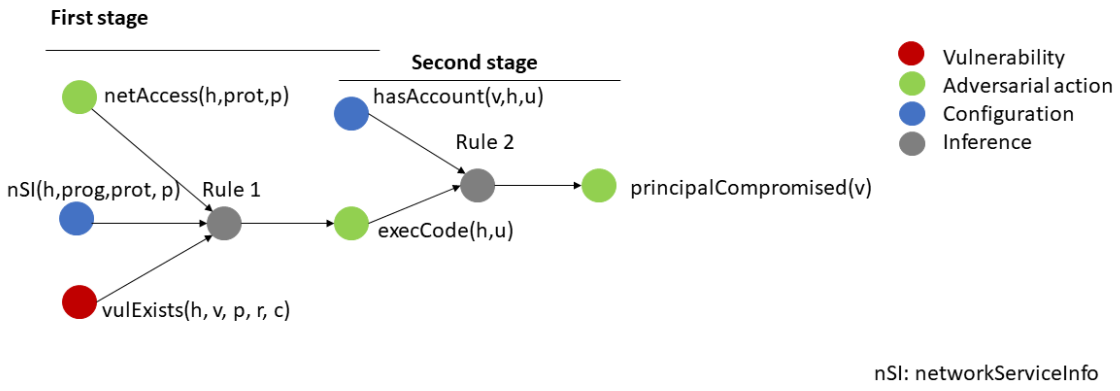


Figure 6.3 An extract of a multi-stage Attack Graph

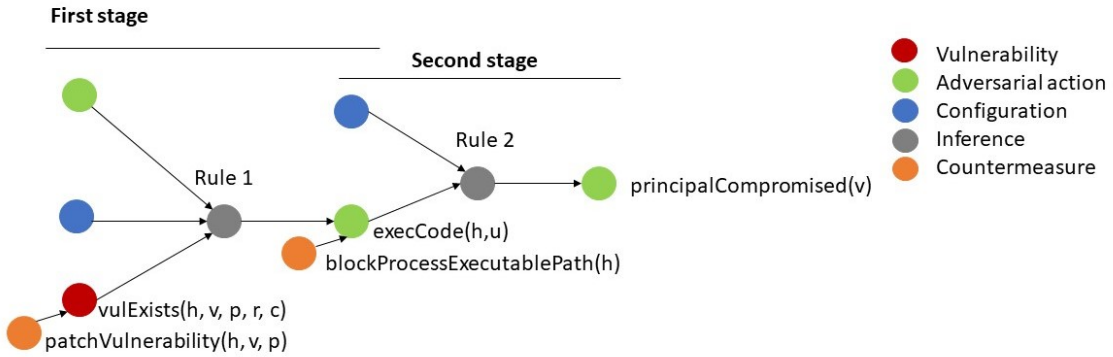


Figure 6.4 Countermeasures applied on some nodes

As long as there are no alerts matched with the predicates *networkServiceInfo(h,prog,prot,p)* and *netAccess(h,prot,p)*, the solution only instantiate a countermeasure predicate on the node interrelated to the predicate *vulExists(h,v,p)*. Figure 6.4 shows the countermeasure predicate instantiation *patchVulnerability(h,v,p)* on the AG predicate *vulExists(h,v,p)*. This logical rule illustrates the anti-correlation linked with mapping the countermeasure predicate to the AG predicate as follows:

$$vulExists(h, v, p) \wedge patchVulnerability(h, v, p) \rightarrow not(execCode(h, u))$$

This means patching the existing vulnerability on host h prevents code execution on h.

When a triggered alert matches the predicates *networkServiceInfo(h,prog,prot,p)* and *netAccess(h,prot,p)*, all preconditions for the *execCode(h,u)* action are met. Consequently, this solution prevents the adversary from achieving the second-stage goal of compromising a victim v. To address this, a countermeasure predicate *blockProcessExecutablePath(h)* must be applied to the predicate *execCode(h,u)*. The correspondent logical rule is as follows: *execCode(h,u)* $\wedge$ *blockProcessExecutablePath(h)* $\rightarrow$ *not(principalCompromised(v))*.

6.5 Complexity Analysis

Some countermeasures may affect others, which are referred to as conflicting countermeasures.

An AG has N nodes. The algorithm successfully treats the alerts A; however, an alert can be mapped to several nodes. While a new alert is generated, the algorithm associates the alert with the corresponding nodes in the AG by traversing all the nodes of the AG having at least one predecessor V. The complexity of the loop on all the nodes with at least one predecessor is $\mathcal{O}(V)$ in the worst case.

Then, the algorithm checks the mapped countermeasures C against the node successor S or its predecessors P. When several preconditions are achieved, their inference is represented by an AND node; the post-condition of the AND node is then their successor. Therefore, the number of successors S is always 1 as each AND node has one post-condition. The complexity for finding countermeasures for the successor is $\mathcal{O}(1 * C)$.

When the flow value is inferior to the number of predecessors, at least one of the predecessors is not concerned by the alert. In this case, the number of nodes on which countermeasures should be applied is $P - 1$ in the worst case. The complexity for searching countermeasures for the predecessors is the $\mathcal{O}((P - 1) * C)$.

The total number of countermeasure checks is $\mathcal{O}(1 + P - 1) * C)$ which simplifies to $\mathcal{O}(P * C)$. Applying the countermeasures on their corresponding node has a $\mathcal{O}(1)$ complexity for each application.

Therefore, every time a new alert is generated, the algorithm traverses all the nodes that have a predecessor until the alert pairing is found, which has a complexity of $O(V)$. When a match is found in the worst case, it takes a complexity of $\mathcal{O}(P * C)$ to identify its correlated countermeasures and to instantiate them.

Therefore, the time complexity for the countermeasures instantiating on the AG is $\mathcal{O}(V * P * C)$, knowing that V and P are inferior to the number of nodes N on the AG.

Ou et al. [71] demonstrate that the time complexity for a LAG generation in the worst case for a fully connected network is $\mathcal{O}(n^3)$, where n is the number of hosts. Therefore, the algorithm to instantiate countermeasures on a LAG does not cause a worse time complexity since the complexity is still cubic in the worst case.

The IR actions instantiated on the AG come from the IR playbooks generated for the AG vulnerabilities. Even if a generated alert is not associated with a vulnerability, the solution consistently applies vulnerability patching actions in the predicates table to all vulnerability predicates, thereby preventing conflicts with potential recovery actions. The approach obtains the playbooks generated for all the AG vulnerabilities when an alert is generated. Then, the tool's time complexity depends on the number of vulnerabilities V on the AG. The AG generator traverses all the AG vulnerabilities to find the playbooks, so the complexity of querying the playbooks is annotated as $O(V)$.

In parallel, the playbook generation process depends on the amount of equipment E in the system. The solution first evaluates all system equipment and then obtains a set of equipment to identify the actions involved in the IR playbook generation process. The complexity of this operation is $O(E)$.

The solution compares the system equipment E with security tools S required for the IR actions execution to get the series of actions that should be involved in the playbook generation process. This complexity operation is then $O(E * S)$, considering each iteration takes $O(1)$ time.

As the previous loop runs E times, the solution appends S elements to the list of actions at the worst, resulting in a list of $E * S$ elements. The complexity of this actions-trimming process, which verifies if the actions can stop the attacker based on his/her position, is $O(E * S)$.

The solution generates all possible combinations of this set of actions, organizing them according to the IR stage to which each belongs.

There is a loop that iterates from $r = 2$ to $r = min(E * S, 7)$. The number of iterations is at most 6 (for r=2,3,. . . ,7) if $E * S \geq 7$. The combination of actions generated contains at least 2 actions and at most 7 actions.

When the number of actions $E*S \geq 7$, the complexity is dominated by the maximum number of combinations, $C(E * S, 7)$, and the cost per combination $O(7)$. The time complexity is $O((E * S)^7)$r.

When n<7, the complexity is the sum of $C(E * S, r) * r$ for $r = 2$ to n. Then the time Complexity is $O((E * S)^{E*S})$ in the worst case (for small n).

The time complexity of IR playbook generation is non-polynomial regarding the amount of

equipment in the system, underlining the significant computational demand. Section 6.11 evaluates the real-time IR actions instantiation on an AG regarding time performance and discusses the computational limitation solution.

6.6 The ADG generator Implementation

This section explains the proposed solution's technical implementation, presents the involved technologies, and demonstrates the use case scenario used to validate this approach. Figure 6.5 depicts the solution's technical implementation, the technologies involved, and how they are connected.

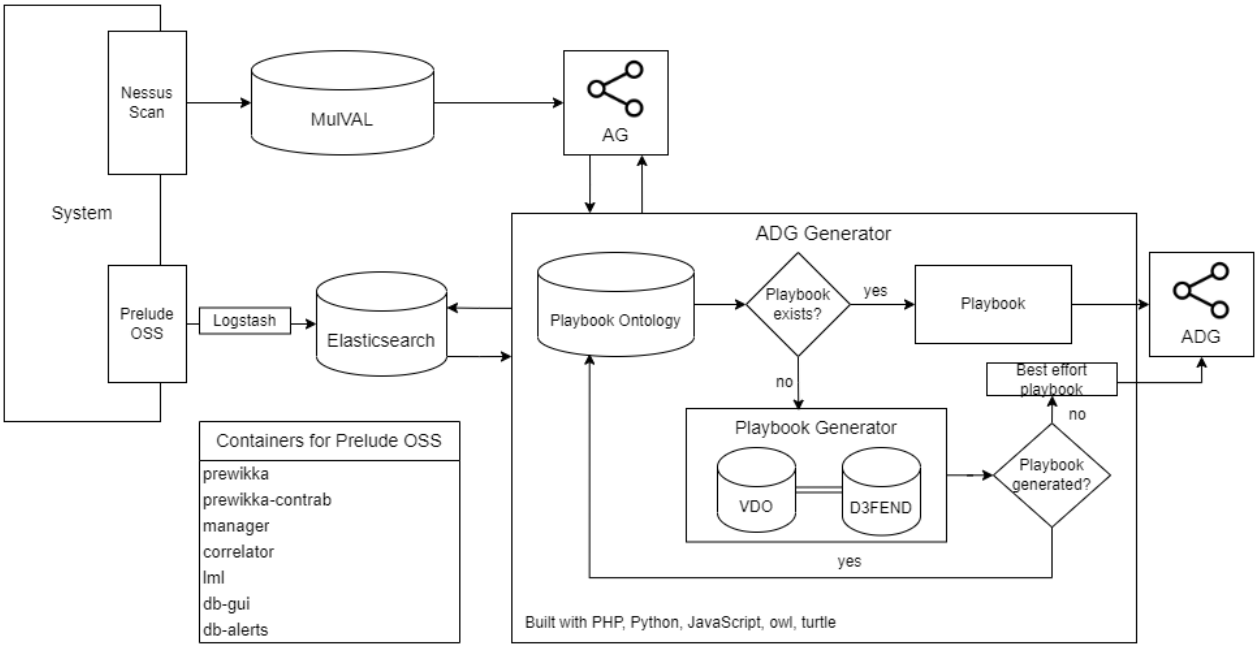


Figure 6.5 The ADG generator architecture

Docker [3] is used to simplify future installations of this solution. Docker is a platform that automates application deployment, scaling, and management by packaging them into normalized units called containers. Containers are created for the essential components of the proposed solution.

An Apache container is created where MulVAL, a LAG generator, is installed. Several modules are created using PHP, JavaScript, and Python, allowing the user to interact with the AG generator. A user provides a Nessus[4] (a network scanner able to discover vulnerabilities)

[3]https://www.docker.com/
[4]https://fr.tenable.com/products/nessus/nessus-essentials

scan output or an input file containing the discovered vulnerabilities, the network configuration information, the adversary location, and the attack goal to a web interface to initiate the LAG generation. The AG visualizer interacts with the logical reasoner to facilitate AG generation, which is displayed on the web and stored in the backend.

A dockerized ADG generator is developed using Python, PHP, and JavaScript, and it is organized into multiple modules. A module is tasked to correlate AG information with alert data by continuously querying an Elasticsearch index, where alerts from the SIEM, Prelude OSS[5], are stored.

The SIEM is implemented using docker; a container for each module and agent of Prelude OSS for the databases linked to Prelude OSS is built. The Intrusion Detection System (IDS), Suricata[6], is also installed as an agent on Prelude OSS.

A container for Logstash[7] is implemented to ingest the generated alert into an implemented container for Elasticsearch [8]. When a new alert is generated, the ADG generator correlates the alert to the AG information.

The ADG generator checks which node of the AG is affected by the alert and queries the playbook ontology to locate a corresponding playbook. If no suitable playbook is found for exploitation, the generator includes a module for playbook generation. If the playbook generation process fails, a default playbook is chosen. The generator then instantiates the interrelated IR actions of the playbook on the AG by looking for the countermeasure predicates that can mitigate AG predicates in the predicates table.

A module is built to create the predicates table, which can find the analogous countermeasure predicates for each AG predicate. The countermeasure predicates are created through a POS tagging model trained on some IR actions. A model based on word embedding with the descriptions of the AG predicates, and the countermeasure predicates are then trained. The trained model is then used to find the most similar countermeasure predicate for each AG predicate. The process of creating the predicate table is launched once.

[5]https://www.prelude-siem.org/
[6]https://suricata.io/
[7]https://www.elastic.co/fr/logstash
[8]https://www.elastic.co/fr/elasticsearch

6.7 Experimental Setup

6.7.1 Architecture 1

Figure 6.6 depicts a virtual industrial architecture deployed on a cyber-range, where a crane deposits elements onto a conveyor belt. This infrastructure comprises:

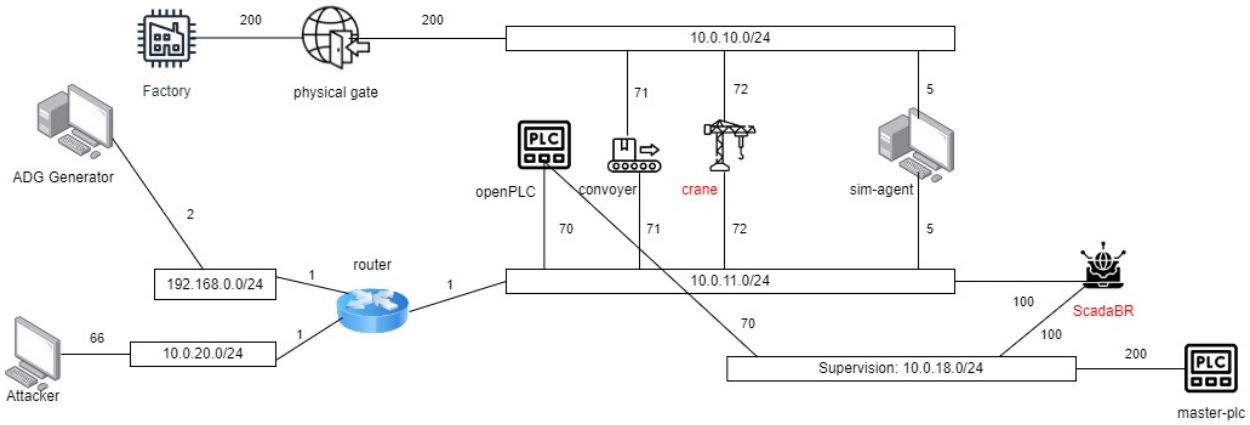


Figure 6.6 A virtual industrial architecture composed of a Computer running FactoryIO, a SimAgent VM that pulls sensor values from FactoryIO and pushes docker slaves' output values, a SCADA that starts and stops the entire process, a Master PLC that forwards the start/stop order to other PLCs, a PLC that controls the slave devices, and a crane that puts elements on a conveyor belt.

- 1 SCADA (ScadaBR): monitors and starts/stops the entire process.
- 1 Master PLC (OpenPLC): forwards the start/stop order to other PLCs.
- 1 PLC (OpenPLC): controls the crane device.
- 2 devices (Modbus slave docker): the crane and the conveyor: emulates the Modbus devices.
- 1 SimAgent VM: pulls sensors values from FactoryIO, and pushes docker slaves outputs values.
- 1 Computer running FactoryIO.

The ADG generator[9] is installed on a workstation located on the LAN network 192.168.0.0 by running the command *sudo make* on its directory. The workstation is configured to monitor

[9]https://github.com/Kekere/prelude-elk

the industrial network. The workstation is set up to collect Syslog logs from other devices by including the following lines in /etc/Syslog.conf $*.*@@localhost:514$, $*.*@localhost:514$ and uncommenting these lines in /etc/rsyslog.conf:

```
module(load="imudp")
input(type="imudp" port="514")

module(load="imtcp")
input(type="imtcp" port="514")
```

The router is set up to send all rsyslog logs and traffic from the interfaces linked to networks 10.0.11.0 and 10.0.20.0 to the ADG generator host machine. This machine receives the syslog log through port 514. Monitoring these networks is possible using the Prelude OSS SIEM integrated into the proposed solution.

Suricata is installed with Prelude enabled on the workstation using the following commands *sudo ./configure –enable-prelude –with-libprelude-prefix=/usr CC="gcc -std=gnu99"*, *sudo make* and $make\ install - full$. Before executing these commands, the following lines in the configure file are commented:

```
# Prelude doesn't work with -Werror
STORECFLAGS="${CFLAGS}"
CFLAGS="${CFLAGS} -Wno-error=unused-result"
```

The file /usr/local/etc/suricata/suricata.yaml is edited to enable prelude alerting. The Suricata agent is linked to Prelude by running the following command:

> *"sudo prelude-admin register Suricata "idea:w admin:r" 0.0.0.0:5553 –uid 0 –gid 0."*

Suricata is run on the interface connected to the local network 192.168.0.0 by executing this command, assuming enp0s3 is the interface linked to the 192.168.0.0 network:

> *"sudo LD_LIBRARY_PATH=/usr/local/lib /usr/local/bin/suricata -c /usr/local/etc/suricata/suricata.yaml -i ens224"*

.

A Vyos router regulates the traffic between the local networks in the industrial infrastructure. The Vyos router is set up to send the network traffic from the local networks 10.0.20.0, where the attacker is located, and 10.0.11.0, where the slave devices, the SIM agent, the openplc, and the SCADA are located, to network 192.168.0.0. The router can transfer all the traffic from 10.0.20.0 and 10.0.20.0 to the network 192.168.0.0.

The ADG generator receives an input file in format .P, one of the formats accepted by the MulVAL logical reasoner. Further interaction rules are created for the use-case scenario based on the same normalized predicates in MulVAL, ensuring that these rules are correctly built. One of the rules illustrates the brute force attack leading to the compromise of a victim. The requirements leading to this fact are represented by the predicates constituting the body of the rule. This rule follows:

```
principalCompromised(Victim) :
    inCompetent(Victim),
    netAccess(_host, _protocol, _port),
    hasAccount(Victim, _host, _user).
```

The second rule applies to the RCE. It requires adversary authentication with the user's compromised credentials and a specific vulnerability on a host H. This rule follows:

```
execCode(H, Perm) :
    vulExists(H, _, Software, remoteEx-
    ploit, privEscalation),
    hasAccount(Victim, H, Perm),
    principalCompromised(Victim)).
```

6.7.2 Architecture 2

This solution is installed on a workstation named ADG generator in Figure 6.7 on the LAN network 192.168.0.0/24. This workstation is connected to the networks 10.0.13.0/24 and 10.0.18.0/24 as well in order for it to receive the traffic from these networks. Suricata is installed, enabling Prelude on the workstation with these commands: *sudo ./configure –enable-prelude –with-libprelude-prefix=/usr CC="gcc -std=gnu99"*, *sudo make* and *make install − full*. Before running these commands, on the following lines are commented in the configure file:

```
# Prelude doesn't work with -Werror
```

```
STORECFLAGS="${CFLAGS}"
CFLAGS="${CFLAGS} -Wno-error=unused-result"
```

The file /usr/local/etc/suricata/suricata.yaml is edited to enable prelude alerting. The Suricata agent is connected to Prelude by executing this command: *sudo prelude-admin register Suricata "idea:w admin:r" 0.0.0.0:5553 –uid 0 –gid 0.* Suricata is run for the interface connected to the local network 192.168.0.0 by executing the following command supposing that ens224 is the interface connected to the network 192.168.0.0: *sudo LD_LIBRARY_PATH= /usr/local /lib /usr/local/bin/suricata -c /usr/local/etc/suricata/suricata. yaml -i ens193 -i ens256.*

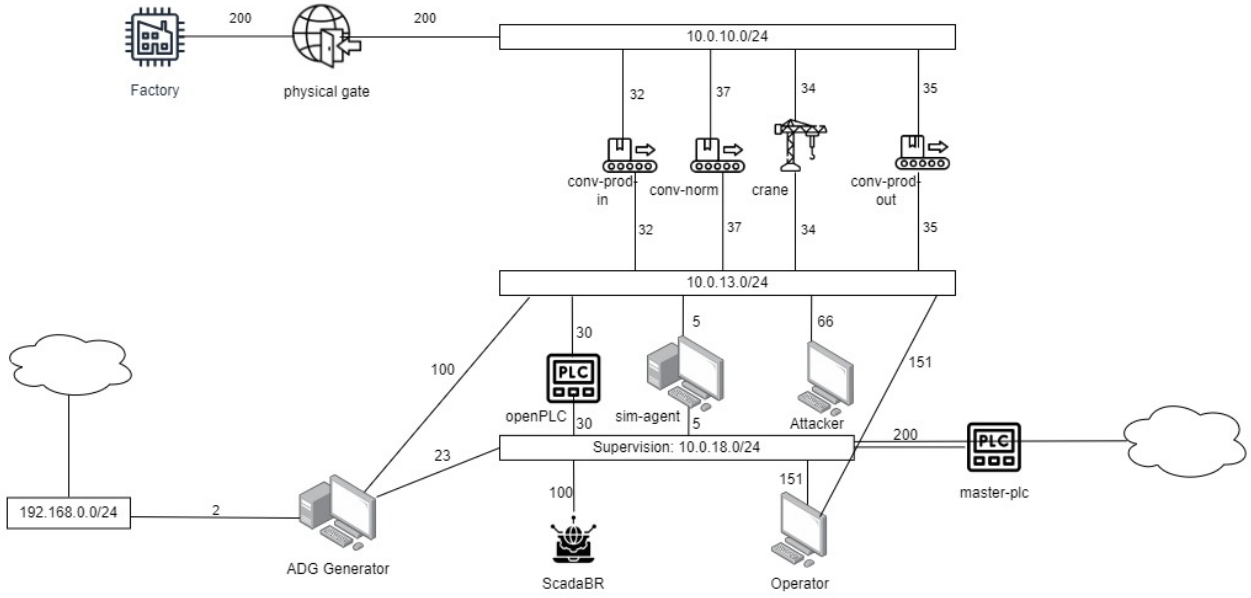


Figure 6.7 A virtual industrial architecture with a computer running FactoryIO, a SimAgent VM for sensor and output management, an operator workstation, SCADA for process control, a master PLC coordinating other PLCs, a slave PLC managing devices, and a crane handling elements on conveyors.

6.8 Use case Scenario

6.8.1 Scenario 1

The vulnerability identified as **CVE-2021-26828** exists on the SCADA. The vulnerability exploitation consists of Remote Code Execution (RCE). This vulnerability facilitates remote authenticated users to upload and execute arbitrary JSP files through the view_edit.shtm

endpoint. The vulnerability exploit impacts the integrity, confidentiality, and availability of the system's information. Once exploited, the adversary can gain privileges, leading him/her to read and modify sensitive information that affects all industrial processes.

Nevertheless, to exploit the vulnerability, the attacker must authenticate; the attacker should steal the credentials first. Suppose the adversary launches a brute force attack by executing a command remotely to obtain a user's credentials who defined a weak password.

6.8.2 Scenario 2

The SCADA system is vulnerable to Remote Code Execution (RCE), identified by the CVE identifier **CVE-2021-26828**, which leads to privilege escalation. As for Scenario 6.8.1, the adversary performs a brute force attack by launching a command remotely to gain a user's credentials who has set a weak password.

The crane and the OpenPLC are vulnerable to a Man-in-the-Middle (MITM) attack. The vulnerability identifier is **CVE-2023-48795**. After obtaining the administrator privileges on the SCADA, the attacker can connect as root through SSH. Then, the adversary can launch the MITM attack as he/she is now on the same local network as the OpenPLC and the crane.

This solution can only capture traffic passing through the router. Then, it cannot intercept the MITM attack. The goal is to block the adversary before launching the MITM attack. Section 6.9 presents the AG generated, and the countermeasures instantiated on the AG for each attack path when a generated alert matches an AG node. The relevance of the predicates of the instantiated IR actions to the AG is evaluated in terms of security in Section 6.10.

6.8.3 Scenario 3

This attack scenario involves 2 local networks: 10.0.13.0/24 and 10.0.18.0/24. The adversary located on 10.0.13.0.0/24 scans the network from this network and sees the opened ports on the machines. An operator workstation is vulnerable to Bluekeep (CVE-2019-0708). It enables the attacker to connect remotely to the operator with the IP address 10.0.13.151. From the operator machine, the adversary scans the network 10.0.18.0/24 and discovers that the SCADA is vulnerable to RCE (CVE-2021-26828). However, the adversary needs a user's credentials to exploit this vulnerability.

He/she launches an MITM attack between the PLC and the conv-prod-in to cause a Denial-of-Service (DoS). Then, he/she forces an administrator to connect to the operator machine and then to the SCADA to steal the administrator's credentials with a keylogger. Once the attacker gains the credentials, he/she exploits the RCE vulnerability on SCADA. From

the SCADA, the adversary launches an MITM attack between the PLC and the conv-norm, leading to stopping the process and killing the SCADA process.

6.9 Experimental Results

6.9.1 Countermeasures Selection

The ADG generation process requires the automated selection of countermeasures for the vulnerabilities identified in the AG. This section presents some countermeasures selected for the vulnerabilities whose CVE IDs are CVE-2021-26828 and CVE-2023-48795.

CVE-2021-26828

Several countermeasures are selected for CVE-2021-26828 thanks to the graph-matching approach presented in Chapter 4. The selected countermeasures concern different types of artifacts. Following are some of the selected countermeasures for CVE-2021-26828.

1. Some countermeasures are associated with a process, such as Process Code Segment Verification, Memory Boundary Tracking, and Process Segment Execution Prevention.
2. Other countermeasures are related to the network traffic, such as Network Traffic Community Deviation and Remote Terminal Session Detection.
3. Another category is linked to an executable file, such as Dynamic Analysis, File Encryption, File Removal, and File Analysis.

These countermeasures are involved in generating an optimal IR playbook illustrated in Section 6.9.2.

CVE-2023-48795

The graph-matching process enables the selection of several countermeasures for the vulnerability identified as CVE-2023-48795. A list of some selected countermeasures follows above:

1. Some countermeasures are related to authentication: Authentication Event Thresholding, Resource Access Pattern Analysis, and Session Duration Analysis.
2. Others are linked to the executable file artifact: File Analysis, File Removal, and File Integrity Monitoring.

3. Some countermeasures are associated with the network traffic artifact: Network Traffic Filtering, Remote Terminal Session Detection, and DNS Traffic Analysis.

6.9.2 Optimal Playbooks

Based on countermeasures selected for CVE-2021-26828 and CVE-2023-48795 in Section 6.9.1, the playbook generator generates optimal playbooks for these vulnerabilities. This Section presents the optimal IR playbooks generated for CVE-2021-26828 and CVE-2023-48795. Considering the security tools (firewall software, SIEM, Suricata, etc.) available in the infrastructures presented in Section 6.7, the playbook generator generates the optimal IR playbooks.

IR Playbook generated for CVE-2021-26828

Figure 6.8 illustrates the optimal IR playbook involved in the ADG generation.

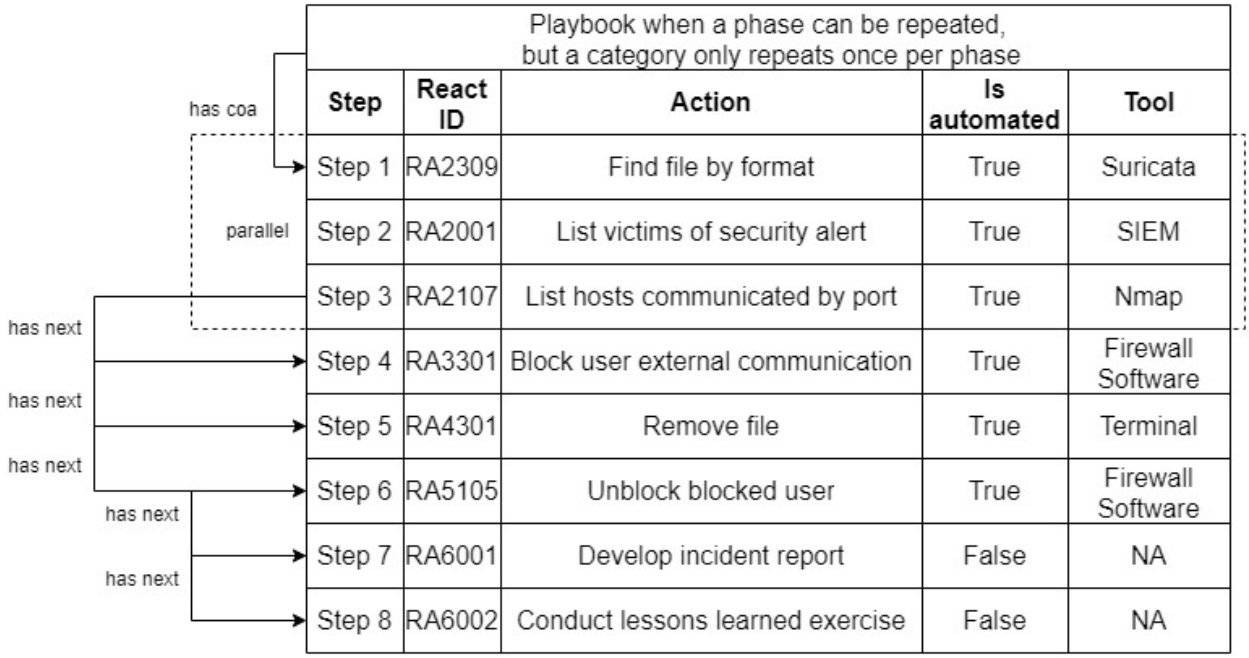
Playbook when a phase can be repeated, but a category only repeats once per phase

Step	React ID	Action	Is automated	Tool
Step 1	RA2309	Find process by executable format	True	Terminal
Step 2	RA2001	List victims of security alert	True	SIEM
Step 3	RA2309	Find file by format	True	Suricata
Step 4	RA2107	List hosts communicated by port	True	Nmap
Step 5	RA3109	Block user external communication	True	Fail2ban
Step 6	RA3404	Block process by executable format	True	Firewall Software
Step 7	RA5105	Unblock blocked user	True	Fail2ban
Step 8	RA5401	Unblock blocked process	True	Firewall Software
Step 9	RA6001	Develop incident report	False	NA
Step 9	RA6002	Conduct lessons learned exercices	False	N/A

Figure 6.8 Optimal IR playbook for CVE-2021-26828

IR playbook generated for CVE-2023-48795

Figure 6.9 illustrates the optimal IR playbook involved in the ADG generation.



Playbook when a phase can be repeated, but a category only repeats once per phase

Step	React ID	Action	Is automated	Tool
Step 1	RA2309	Find file by format	True	Suricata
Step 2	RA2001	List victims of security alert	True	SIEM
Step 3	RA2107	List hosts communicated by port	True	Nmap
Step 4	RA3301	Block user external communication	True	Firewall Software
Step 5	RA4301	Remove file	True	Terminal
Step 6	RA5105	Unblock blocked user	True	Firewall Software
Step 7	RA6001	Develop incident report	False	NA
Step 8	RA6002	Conduct lessons learned exercise	False	NA

Figure 6.9 Optimal IR playbook for CVE-2023-48795

6.9.3 Scenario 1

AG Generation

MulVAL identifies multiple attack paths and produces the LAG. Figure 6.10 illustrates the LAG generated for the use case scenario. Green nodes represent primitive network details, including running services, open ports, and identified vulnerabilities on the system. Pink nodes represent inferences derived from preconditions that lead to an adversarial action. Blue nodes illustrate the adversary's action. The logical reasoner detects several attack paths. Three attack paths are explained in the following paragraphs, mentioning only the nodes involved in these attack paths.

Attack Path 1 Node 6 depicts the adversary's position on network 10.0.20.0. Node 5 illustrates that network 10.0.11.0 is accessible to 10.0.20.0 through the TCP protocol and port 8080. Node 4 describes the inference made from nodes 5 and 6's fulfillment, leading to an adversary's action illustrated by node 3. Node 3, the goal of this attack path, represents

that the adversary can send packets to port 8080 on the SCADA through the TCP protocol.

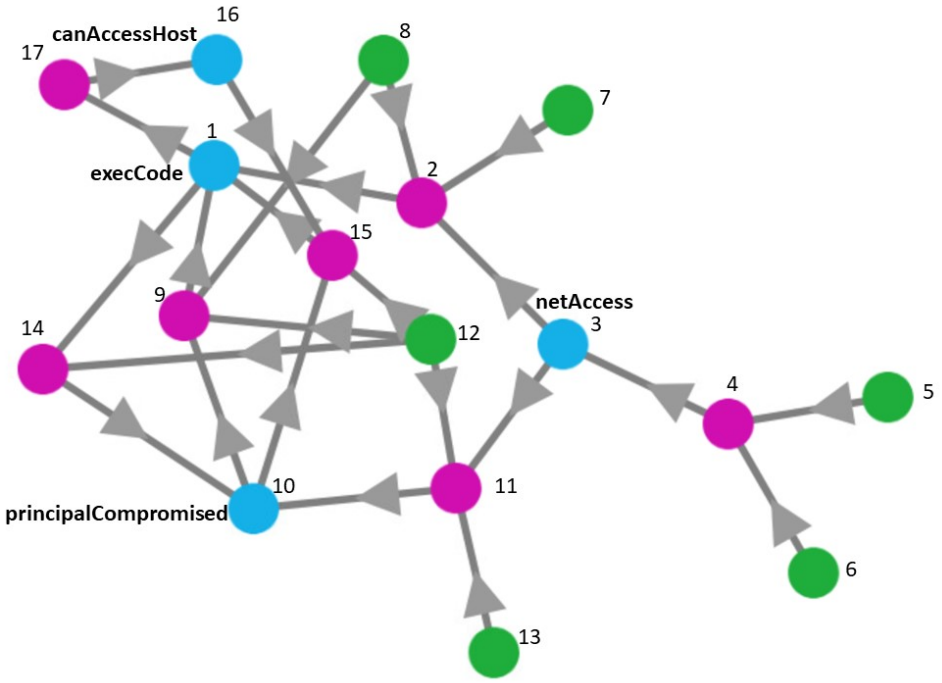


Figure 6.10 The AG generated for the first use-case scenario

Attack Path 2 Hence, the adversary has access to the SCADA system port 8080 with the IP address 10.0.11.100 (Node 3); she/he can leverage this access to initiate a brute force attack, depicted by node 10, via the ScadaBR login page trying several credential combinations until the correct one is found. Node 11 draws the inference made when node 12, which indicates a user logged into an account on ScadaBR, and node 13, which exemplifies that the user is incompetent, are both met, resulting in the brute force attack.

Attack Path 3 Node 13 suggests that a user, Bob, is incompetent due to the weakness of his credentials. Node 12 implies that Bob has an account connected to ScadaBR. Node 11 illustrates the inference that leads to node 10 when nodes 3, 12, and 13 are satisfied, indicating that the user has been compromised, as demonstrated by node 10. Once the adversary gains these credentials, remote code execution (RCE) on 10.0.11.100, depicted by node 1, becomes possible due to the vulnerability represented by node 8, with port 8080 opened for the ScadaBR service over the TCP protocol (node 7). Node 9 reflects the inference drawn from the fulfillment of nodes 7, 8, and 10, culminating in the attack path goal of RCE.

Since the logical reasoner examines all possibilities, the AG also draws alternative paths the adversary could take to achieve the goal of node 1, assuming the attacker already has the credentials for any user. Additionally, node 14 depicts an inference of password sniffing for the user Bob following the code execution.

The AG is automatically updated whenever the user gives the tool a new input file. The tool also initiates the AG update when a new alert is generated. The algorithm traverses the AG to identify the node affected by the generated alert, enabling it to instantiate countermeasure predicates on the AG, which leads to the ADG generation described in Section 6.9.3.

ADG Generation

Table 6.3 illustrates the mapped generated playbook IR action predicates linked to the vulnerability identified by ID CVE-2021-26828 alongside the AG predicates. The first column identifies each match; its name begins with the letter C for countermeasure.

Table 6.3 The 11 matched AG predicates with the IR actions from the CVE-2021-26828 playbook

Name	CVE	Countermeasures	Predicates
C1	CVE-2021-26828	findProcessExecutableFormat(_host)	netAccess(_machine, _protocol, _port)
C2	CVE-2021-26828	findProcessExecutableFormat(_host)	execCode(_host, _user)
C3	CVE-2021-26828	listHostsCommunicatedPort(_port)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C4	CVE-2021-26828	listHostsCommunicatedPort(_port)	hacl(_src, _dst, _prot, _port)
C5	CVE-2021-26828	listHostsCommunicatedPort(_port)	hasAccount(_principal, _host, _account)
C6	CVE-2021-26828	listHostsCommunicatedPort(_port)	netAccess(_machine, _protocol, _port)
C7	CVE-2021-26828	listHostsCommunicatedPort(_port)	execCode(_host, _user)
C8	CVE-2021-26828	blockUserExternalCommunication(_principal)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C9	CVE-2021-26828	blockProcessExecutableFormat(_host)	netAccess(_machine, _protocol, _port)
C10	CVE-2021-26828	blockProcessExecutableFormat(_host)	execCode(_host, _user)
C11	CVE-2021-26828	blockUserExternalCommunication(_principal)	execCode(_host, _user)

When a generated alert corresponds to an AG node, the algorithm checks for the countermeasure predicates associated with the AG node's predicate in Table 6.3 to instantiate the

countermeasure predicates on the AG. As previously mentioned, the algorithm considers two conditions when instantiating countermeasures on the AG:

- **Condition 1**: The flow value is equal to the number of predecessors of a node.
- **Condition 2**: The flow value is inferior to the node number.

Two types of adversarial actions are reproduced. For condition 1, an alert with high severity is provoked, starting the brute force attack with this command:

```
hydra -L userfile.txt -P passwordfile.txt -u -f 10.0.11.100 -s 8080 http
-post-form "/login.php:username=^USER^&password= ^PASS^:F=<form name='login'"
```

The generated alert is associated with nodes 3, 5, and 7 in Figure 6.11, whose predicates are *netAccess('10.0.11.100',tcp,8080)*, *hacl('10.0.20.66','10.0.11.100',tcp,8080)*, and *networkServiceInfo('10.0.11.100',scadabr,tcp,8080,user)*, respectively. The flow value for these nodes updates to 1. The flow from node 8, which illustrates the vulnerability on ScadaBR, starts with a value of 1 when the AG is generated. The total flow value reaching node 2 is 3, corresponding to the number of its predecessor nodes. In this scenario, the countermeasure predicates should be instantiated on the successor of node 2, node 1. Based on Table 6.3, the predicates matched with *execCode(_host,_user)* are *blockProcessExecutableFormat(_host)*,*findProcessExecutableFormat(_host)*, *listHostsCommunicatedPort(_port)* and *blockUserExternalCommunication(_principal)*.

Figure 6.11 represents the countermeasures instantiation for condition 1. The 4 countermeasures above predicates are instantiated on node 1, depicting code execution. Node 3 has another successor, node 11, but the flow value from the other predecessors of 11 is equal to 0; the algorithm instantiates 3 countermeasures predicates *blockProcessExecutableFormat(_host)*,*findProcessExecutableFormat(_host)* and *listHostsCommunicatedPort(_port)* on node 3.

For condition 2, an alert with a low severity is triggered by sending packets to the target machine. The generated alert is mapped to nodes 5 and 7, and their outgoing flow values change to 1. The flow value of node 6 is 1 when the AG is generated. Then, the total flow value entering node 4 equals its number of predecessors, 2. Then, as shown in Figure 6.11 illustrates, the countermeasure nodes are being instantiated on the successor of node 4, node 5. The flow value from node 5 equals 1, as well as the flow value from node 7 that was matched with the alert. Node 8 represents a vulnerability existing in the system, and its flow value is 1. Then, the entering flow value to node 2 equals its number of predecessors, 3. Countermeasures are then instantiated on node 1, the successor of node 2.

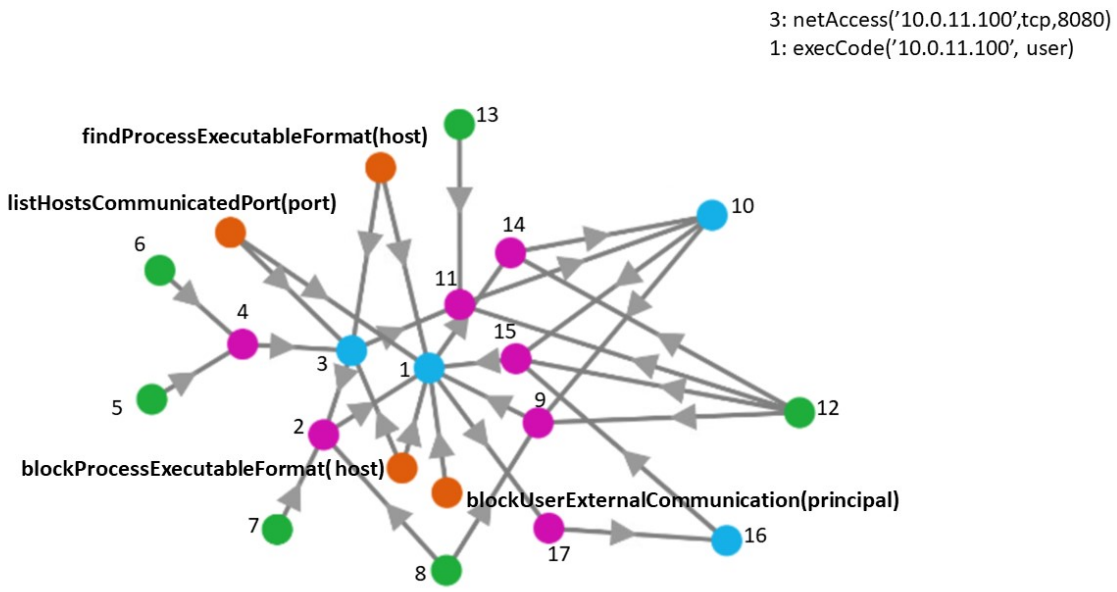


Figure 6.11 The ADG generated for for the packets sent and the brute force attack launched. The orange nodes represent the countermeasures in response to the brute force attack instantiated on the AG generated previously in Figure 6.10

6.9.4 Scenario 2

AG Generation

Figure 6.12 represents the LAG generated for Scenario 6.8.2. It is an extended AG of the graph generated for Scenario 6.8.1 represented in Figure 6.10. Green nodes indicate primitive network information, such as running services, open ports, and discovered vulnerabilities. Pink nodes represent inferences from preconditions that lead to adversarial actions, while blue nodes depict the adversary's actions. The attack goal consists of exploiting the vulnerability whose existence is represented by node 24. Exploiting this vulnerability leads to an MITM attack represented by node 1. The MulVAL generator finds two additional attack paths. The adversary can send packets to the crane with IP address 10.0.11.72 from the Scada with IP address 10.0.11.100 once he/she gains administrator privileges after launching the RCE. Node 3 represents this adversarial action. Node 1 represents the MITM attack as the adversary is located in the LAN network 10.0.11.0.

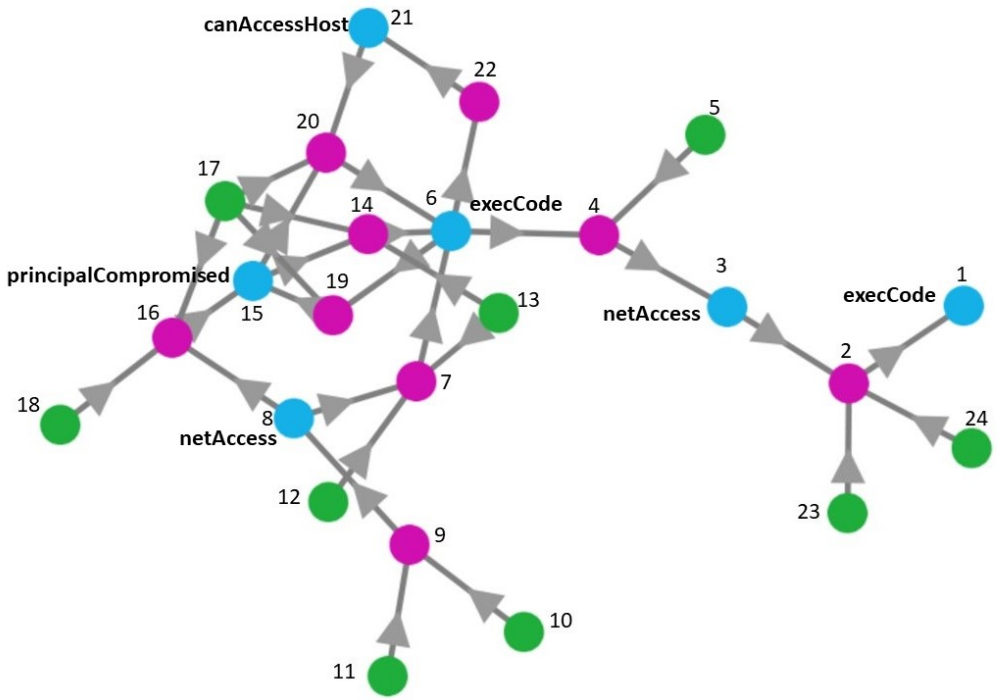


Figure 6.12 The AG generated for the second use-case scenario

ADG Generation

This solution depends on information about the security tools in the system to generate an IR playbook. As mentioned in Section 6.5, this data contributes to retrieving the IR actions involved in the IR playbook generation. Otherwise, it generates a default playbook. This section presents the ADGs generated when the playbooks are generated based on security tools installed on the architecture and when the solution generates the default playbook for CVE-2021-26828 and CVE-2023-48795.

Table 6.4 shows the calculation done by the algorithm for deducing where the countermeasures node should be instantiated on the ADGs generated for the three attack paths presented in this section based on the nodes correlated with the generated attack for the attacker's actions.

Playbooks generated based on security tools Table 6.5 depicts 18 matched countermeasure predicates with AG predicates based on two playbooks generated for CVE-2021-26828 and CVE-2023-48795 using existing industrial security tools.

The proposed scenario presents the ADG for three attack paths, explaining how attacks are launched and detected via the integrated SIEM. The solution is demonstrated by matching

Table 6.4 Flow value calculation for countermeasures instantiation on the AG for Scenario 2

Attack Path	**Action**	**Nodes matched with alert**	**Calculation**	**Countermeasure**
1	Send packets from adversary to SCADA	Nodes 10 and 12	$f(10,9)$=1 and $f(11,9)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(j)} f_t(k,9) = \lvert\Gamma^-(9)\rvert$ $f(12,7)$=1, $f(13,7)$ = 1 and $f(8,7)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(j)} f_t(k,7) = \lvert\Gamma^-(7)\rvert$	Successors: Nodes 8 and 6
2	Brute force attack	Nodes 10, 8 and 12	$f(11,7)$=1 and $f(11,9)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(9)} f_t(k,9) = \lvert\Gamma^-(9)\rvert$ $f(12,7)$=1, $f(13,7)$ = 1 and $f(8,7)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(7)} f_t(k,7) = \lvert\Gamma^-(7)\rvert$	Successors: Nodes 8 and 6
3	The adversary connects through SSH to the SCADA	Node 5	$f(5,4)$=1 and $f(6,4)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(4)} f_t(k,4) < \lvert\Gamma^-(4)\rvert$	Predecessor: Node 5

alert information with the AG to identify affected nodes and searching for and applying countermeasure predicates to the matching AG nodes.

Figure 6.13 shows the ADG for the first and second attack paths. Orange nodes represent the countermeasures in response to the brute force attack on the AG.

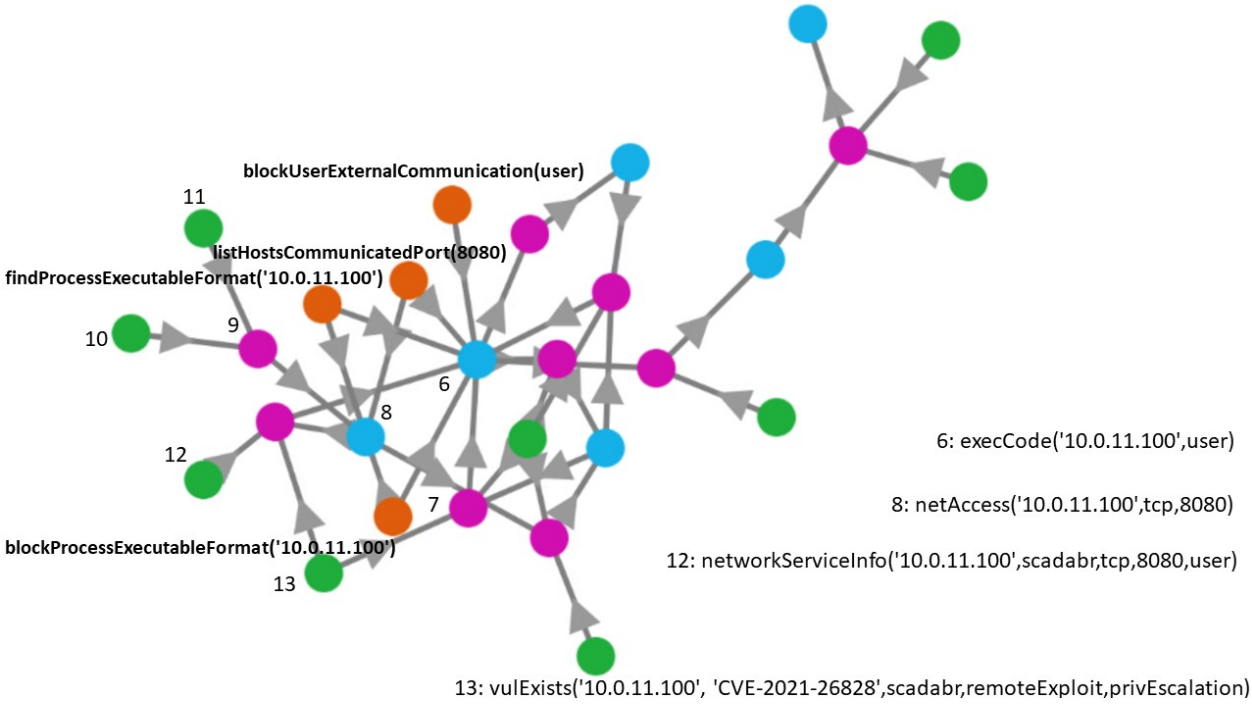


Figure 6.13 The ADG generated for the packets sent and the brute force attack launched.

Table 6.5 The 18 matched AG predicates with the IR actions predicates

Name	CVE	Countermeasures	Predicates
C1	CVE-2021-26828	findProcessExecutableFormat(_host)	netAccess(_machine, _protocol, _port)
C2	CVE-2021-26828	findProcessExecutableFormat(_host)	execCode(_host, _user)
C3	CVE-2021-26828	listHostsCommunicatedPort(_port)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C4	CVE-2021-26828	listHostsCommunicatedPort(_port)	hacl(_src, _dst, _prot, _port)
C5	CVE-2021-26828	listHostsCommunicatedPort(_port)	hasAccount(_principal, _host, _account)
C6	CVE-2021-26828	listHostsCommunicatedPort(_port)	netAccess(_machine, _protocol, _port)
C7	CVE-2021-26828	listHostsCommunicatedPort(_port)	execCode(_host, _user)
C8	CVE-2021-26828	blockUserExternalCommunication(_principal)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C9	CVE-2021-26828	blockProcessExecutableFormat(_host)	netAccess(_machine, _protocol, _port)
C10	CVE-2021-26828	blockProcessExecutableFormat(_host)	execCode(_host, _user)
C11	CVE-2021-26828	blockUserExternalCommunication(_principal)	execCode(_host, _user)
C12	CVE-2023-48795	listHostsCommunicatedPort(_port)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C13	CVE-2023-48795	listHostsCommunicatedPort(_port)	hacl(_src, _dst, _prot, _port)
C13	CVE-2023-48795	listHostsCommunicatedPort(_port)	hasAccount(_principal, _host, _account)
C14	CVE-2023-48795	listHostsCommunicatedPort(_port)	netAccess(_machine, _protocol, _port)
C15	CVE-2023-48795	listHostsCommunicatedPort(_port)	execCode(_host, _user)
C16	CVE-2023-48795	blockUserExternalCommunication(_principal)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C17	CVE-2023-48795	blockUserExternalCommunication(_principal)	execCode(_host, _user)
C18	CVE-2023-48795	unblockBlockedUser(_principal)	networkServiceInfo(_host, _program, _protocol, _port, _user)

Attack Path 1 Packets from the adversary machine located on network 10.0.20.0 are sent to the SCADA, whose IP address is 10.0.11.100. The attacker's location is represented by node 11; the outgoing flow value from this node is at 1 when the AG is generated. The solution correlates the alert generated by sending packets with node 11, resulting in a flow value change

to 1. The total flow value entering node 9, which represents the fulfillment of nodes 10 and 11, is the same as its number of predecessors, 3. Therefore, countermeasures should be applied to its successor, node 8, whose predicate is *netAccess('10.0.11.100',tcp,8080)*. The algorithm searches for the countermeasures associated with this predicate in Table 6.5 and applies them to the AG.

An alert can be correlated to more than 1 node. The tool also correlates the generated alert with node 12, which has the predicate *networkServiceInfo('10.0.11.100',scadabr,tcp,8080, user)*. Node 7 represents the inference from the realization of node 12, node 8, and node 13. Node 13, whose predicate is *vulExists('10.0.11.100','CVE-2021-26828',scadabr, remoteExploit,privEscalation)* has an outgoing flow value of 1, representing a vulnerability. Therefore, node 7 has an incoming flow value of 3, which equals its number of predecessors. Consequently, node 7 has an incoming flow value of 3, corresponding to the number of its predecessors. The tool finally looks for the countermeasure predicates related to the predicate of node 6 in Table 6.5 and instantiates their nodes on the AG.

Attack Path 2 A brute force attack is instantiated on the ScadaBR web interface by launching the following command from the attacker's machine:

```
hydra -L userfile.txt -P passwordfile.txt
-u -f 10.0.11.100 -s 8080 http-post-form
"/login.php:username=^USER^&password=
^PASS^:F=<form name='login'"
```

The ADG generator correlates the generated alert with nodes 10, 8, and 12. The flow value from node 11 starts with 1 when the AG is generated. Then, the total flow value entering node 9 equals the number of its predecessors; countermeasures should then be instantiated on its successor, node 8. The successor of nodes 8 and 12 is node 7, representing its predecessor's fulfillment (nodes 8,12 and 13). Node 13 represents a vulnerability, CVE-2021-48795; its outgoing flow value is 1 when the AG is generated. Furthermore, the flow value coming to node 7 is 3, as well as its number of predecessors. The countermeasure nodes should be instantiated on its successor, node 6, whose predicate is *execCode('10.0.11.100',user)*.

Attack Path 3 The RCE attack leads to privileges escalation. Once the adversary gains administrator privileges, he/she logs in to the SCADA as root through SSH. The SIEM generates an alert. The tool correlates this alert with node 5, whose predicate is *hacl('10.0.11.100', '10.0.11.72',tcp,22)*. The tool looks for the countermeasure predicate related to this AG predicate and instantiates a countermeasure node for this predicate on node 5.

Figure 6.14 represents the ADG generated for the ssh connection to 10.0.11.100.

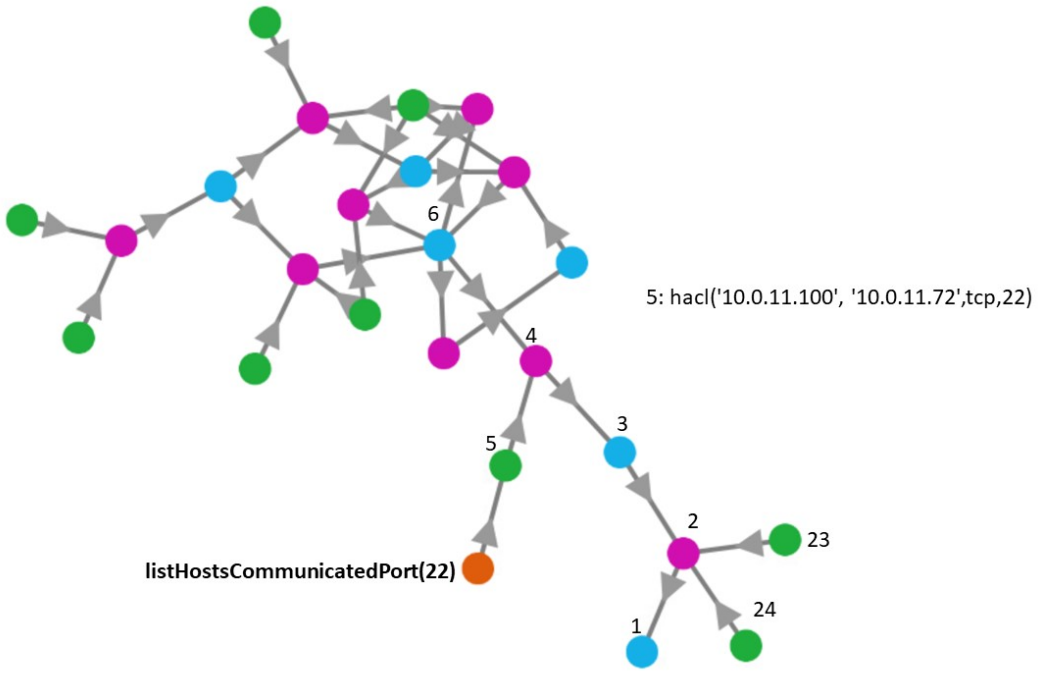


Figure 6.14 The ADG generated when the adversary connects through SSH to the SCADA using root credentials.

Default playbooks When the tool can not generate an optimal playbook because the requirements for playbook generation are not fulfilled, the solution generates a default playbook for the AG vulnerabilities. This section presents a situation where the default playbook is generated for CVE-2021-26828 and CVE-2023-48795. Table 6.6 represents 8 matched AG predicates with the IR actions of these playbooks. The default playbooks' actions consist of listing the vulnerabilities on the hosts and patching them.

Figure 6.15 illustrates the ADG generated for the 2 first attack paths when the adversary sends default packets to 10.0.11.100 and when the brute force attack occurs.

Attack Path 1 Packets from the adversary machine are sent to 10.0.11.100. The SIEM generates an alert. The tool correlates this alert with the AG nodes, identifying that nodes 10 and 12 match the alert. The outgoing flow value from these nodes is then 1. The outgoing flow value from node 11 also has 1 as flow value. The incoming flow to node 9 equals its number of predecessors, resulting in a flow value of 1 from it. Consequently, node 8's flow value is also 1. Node 13, which represents the CVE-2021-26828 vulnerability on the SCADA, has 1

Table 6.6 8 matched AG predicates with the IR actions predicates

Name	CVE	Countermeasures	Predicates
C1	CVE-2021-26828	patchVulnerability(_machine, _vulID _program)	vulExists(_machine, _vulID, _program, _range, _consequence)
C2	CVE-2021-26828	listHostVulnerabilities(_host)	execCode(_host, _user)
C3	CVE-2021-26828	listHostVulnerabilities(_host)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C4	CVE-2021-26828	listHostVulnerabilities(_host)	hasAccount(_principal, _host, _account)
C5	CVE-2023-48795	patchVulnerability(_machine, _vulID _program)	vulExists(_machine, _vulID, _program, _range, _consequence)
C6	CVE-2023-48795	listHostVulnerabilities(_host)	execCode(_host, _user)
C7	CVE-2023-48795	listHostVulnerabilities(_host)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C8	CVE-2023-48795	listHostVulnerabilities(_host)	hasAccount(_principal, _host, _account)

as flow value, leading to a total flow of 3 entering node 7. Countermeasures are needed for its successor, node 6. The tool instantiates patching actions for nodes depicting vulnerabilities when proposed in the predicates table. Node 13's predicate is *vulExists('10.0.11.100','CVE-2021-26828',scadabr,remoteExploit,privEscalation)*, and node 24 has the same predicate. The solution then searches for a match in Table 6.6 to apply it to the AG.

Attack Path 2 The alert generated for the brute force attack is correlated with nodes 10, 12, and 8, updating their flow values to 1. The flow value from node 13 remains at 1. The total flow value going to node 7 equals its number of predecessors. Then, the tool instantiates the countermeasure on node 7's successor, node 6. The tool also implements countermeasures that involve patching vulnerabilities on the nodes that indicate the existence of those countermeasures.

Attack Path 3 Figure 6.16 illustrates the ADG generated when the adversary connects through ssh on the SCADA. Once the adversary obtains privileges on the system, they con-

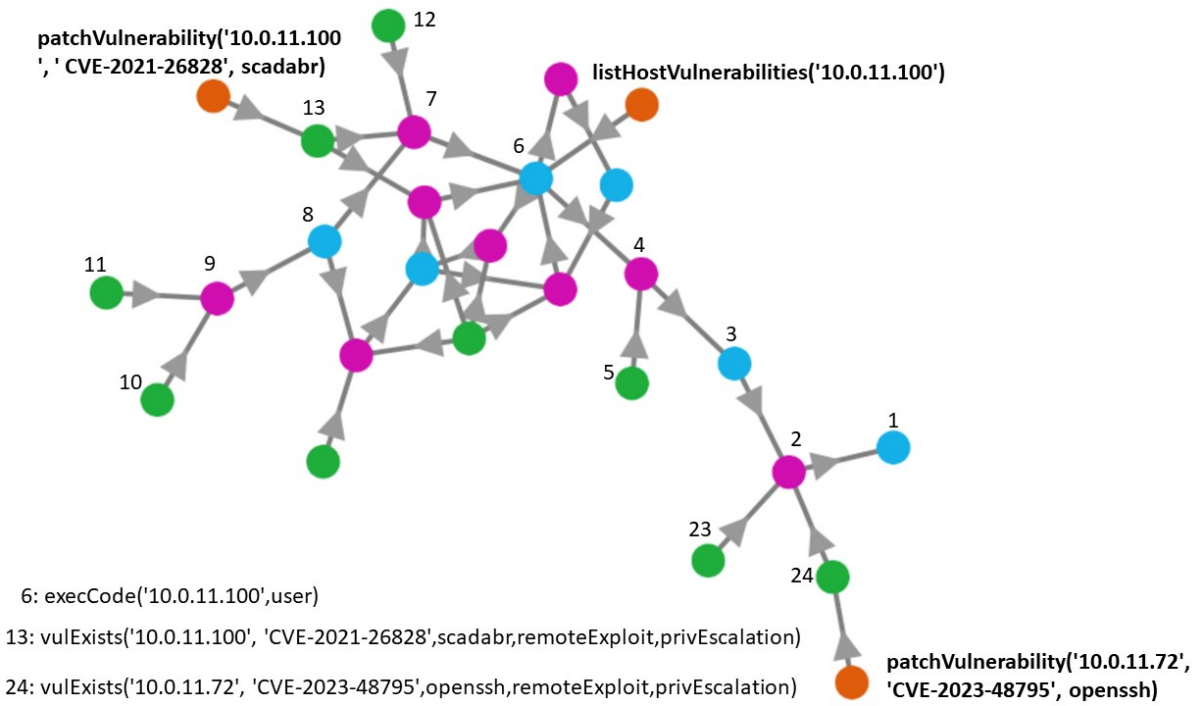


Figure 6.15 The ADG generated by instantiating IR actions of the default playbooks for the packets sent and the brute force attack launched.

nect to the SCADA as root via SSH, triggering an alert. The tool correlates this alert with node 5, whose predicate is *hacl('10.0.11.100','10.0.11.72',tcp,22)*. However, no countermeasure predicate matches this predicate in Table 6.6. The solution only instantiates nodes for the countermeasure predicates related to patching vulnerability on nodes 13 and 23.

6.9.5 Scenario 3

AG Generation

Figure 6.17 represents the LAG generated for Scenario 6.8.3. The attack's final goal consists of launching an MITM between a PLC and a conveyor. This attack goal is represented by node 1. The MulVAL reasoner finds attack paths that can lead to this attack. First, the adversary leads a scan of network 10.0.13.0/24; node 18 represents this action. Exploiting the BlueKeep vulnerability represented by node 23 leads to the code execution represented by node 16. The adversary can then identify the vulnerabilities on the machine in the local network 10.0.18.0/24. To exploit an RCE vulnerability identified on the SCADA 10.0.18.100, the adversary launched an MITM attack between a PLC and a conveyor, leading to DoS represented by node 51. An administrator is then forced to connect to the SCADA (node 49),

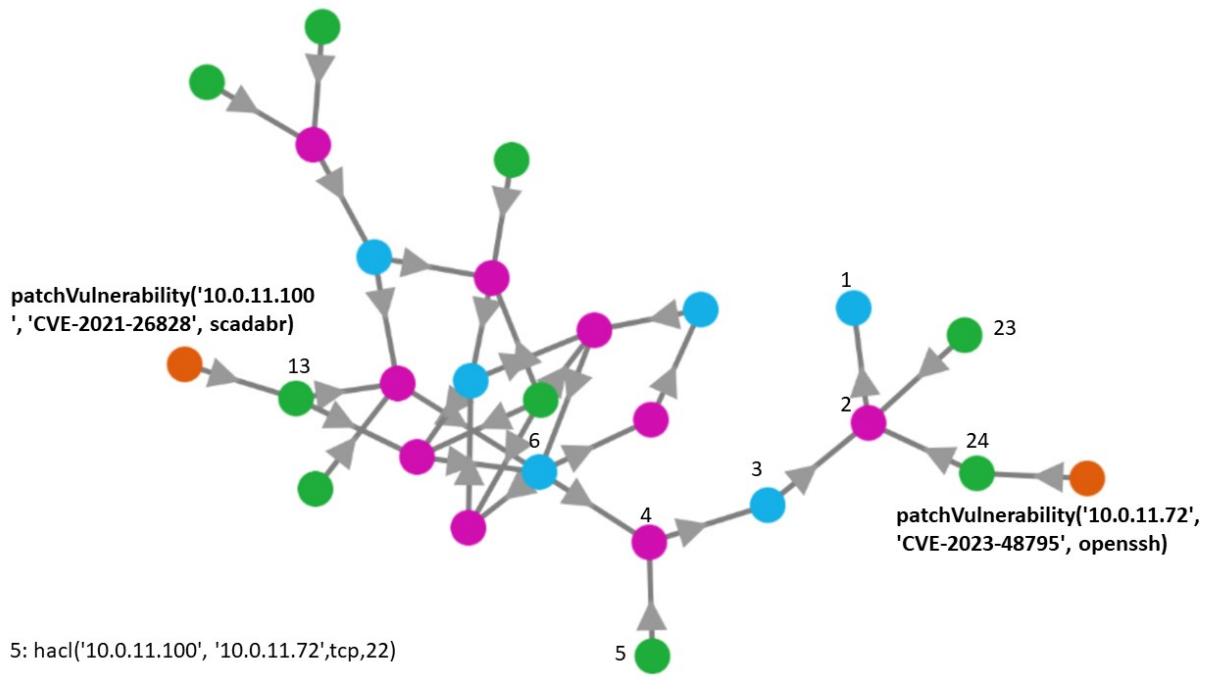


Figure 6.16 The ADG generated when the adversary connects through SSH to the SCADA using root credentials.

leading the adversary to harvest the credentials required to launch the RCE. The exploitation of the RCE vulnerability on the SCADA is represented by node 28. The RCE leads the adversary to launch the final MITM attack.

ADG Generation

Table 6.7 illustrates 25 countermeasures predicates matched with the AG predicates. These countermeasure predicates are based on the playbooks generated for the vulnerabilities identified as CVE-2019-0708, CVE-2021-26828, and CVE-2023-48795.

This section introduces the ADG, a strategic tool that maps out 7 distinct attack paths. It explores the instantiation of countermeasure nodes on the AG, providing insights into the attacker's strategy when launching successful attacks or exploiting vulnerabilities through discovery scans.

Table 6.8 shows the calculation done by the algorithm for deducing where the countermeasures node should be instantiated on the ADGs generated for the seven attack paths presented in this section based on the nodes correlated with the generated attack for the attacker's actions.

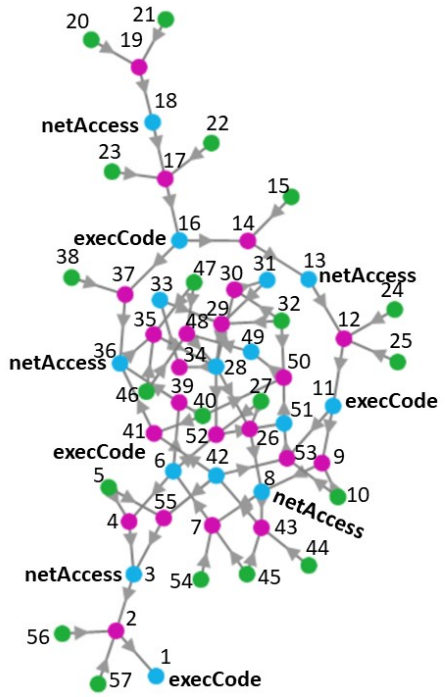


Figure 6.17 The AG generated for the third use-case scenario

Attack Path 1 The first attack path concerns the attacker scanning RDP port 3389 on the operator workstation. When the scan of RDD port 3389 is detected, the nodes matched with the alert are nodes 20 and 22 with predicates respective *hacl('10.0.13.0','10.0.13.151',tcp, 3389)* and *networkServiceInfo('10.0.13.151',rdp,tcp,3389,user)* represented in Figure 6.18. The Suricata rule that enables the detection follows:

```
alert tcp any any -> any 3389 (msg:"Possible SYN scan detected
on port 3389"; flags:S; threshold:type both, track by_src,
count 5, seconds 5; classtype:network-scan; priority:2; sid:
1000028; rev:1;)
```

The flow value from both nodes to their successor's respective nodes, 19 and 17, is 1. The node representing the adversary position, node 21, has a flow value 1. Then, the total flow value of 19 equals 2, the same as the number of predecessors. The flow value going to and from its successor node 18 is 1. Then, the ADG generator instantiates countermeasure nodes on node 18 as it finds matches for the predicate of node 18 in Table 6.7. Node 23, representing the Bluekeep vulnerability, already has its flow value at 1. Then, the total flow value going to node 17 is the same as its number of predecessors, meaning 3. Then, the ADG generator instantiates countermeasure nodes on node 16, the successor of node 17, as it finds matches

Table 6.7 The 25 matched AG predicates with the IR actions predicates

Name	CVE	Countermeasures	Predicates
C1	CVE-2019-0708	listHostVulnerabilities(_host)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C2	CVE-2019-0708	listHostVulnerabilities(_host)	hasAccount(_principal, _host, _account)
C3	CVE-2019-0708	listHostVulnerabilities(_host)	netAccess(_machine, _protocol, _port)
C4	CVE-2019-0708	listHostVulnerabilities(_host)	execCode(_host, _user)
C5	CVE-2019-0708	patchVulnerability(_machine, _vulID _program)	vulExists(_machine, _vulID, _program, _range, _consequence)
C6	CVE-2021-26828	findProcessExecutableFormat(_host)	netAccess(_machine, _protocol, _port)
C7	CVE-2021-26828	findProcessExecutableFormat(_host)	execCode(_host, _user)
C8	CVE-2021-26828	listHostsCommunicatedPort(_port)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C9	CVE-2021-26828	listHostsCommunicatedPort(_port)	hacl(_src, _dst, _prot, _port)
C10	CVE-2021-26828	listHostsCommunicatedPort(_port)	hasAccount(_principal, _host, _account)
C11	CVE-2021-26828	listHostsCommunicatedPort(_port)	netAccess(_machine, _protocol, _port)
C12	CVE-2021-26828	listHostsCommunicatedPort(_port)	execCode(_host, _user)
C13	CVE-2021-26828	blockUserExternalCommunication(_principal)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C14	CVE-2021-26828	blockProcessExecutableFormat(_host)	netAccess(_machine, _protocol, _port)
C15	CVE-2021-26828	blockProcessExecutableFormat(_host)	execCode(_host, _user)
C16	CVE-2021-26828	blockUserExternalCommunication(_principal)	execCode(_host, _user)
C17	CVE-2023-48795	listHostsCommunicatedPort(_port)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C18	CVE-2023-48795	listHostsCommunicatedPort(_port)	hacl(_src, _dst, _prot, _port)
C19	CVE-2023-48795	listHostsCommunicatedPort(_port)	hasAccount(_principal, _host, _account)
C20	CVE-2023-48795	listHostsCommunicatedPort(_port)	netAccess(_machine, _protocol, _port)
C21	CVE-2023-48795	listHostsCommunicatedPort(_port)	execCode(_host, _user)
C22	CVE-2023-48795	blockUserExternalCommunication(_principal)	networkServiceInfo(_host, _program, _protocol, _port, _user)
C23	CVE-2023-48795	blockUserExternalCommunication(_principal)	execCode(_host, _user)
C24	CVE-2023-48795	unblockBlockedUser(_principal)	networkServiceInfo(_host, _program, _protocol, _port, _user)

Table 6.8 Flow value calculation for countermeasures instantiation on the AG for Scenario 3

Attack Path	**Action**	**Nodes matched with alert**	**Calculation**	**Countermeasure**
1	Exploitation of Bluekeep vulnerability	Nodes 20, 22 and 18	$f(20,19)$=1 and $f(21,19)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(j)} f_t(k,19) = \lvert\Gamma^-(19)\rvert$ $f(22,17)$=1, $f(23,17)$ = 1 and $f(18,17)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(j)} f_t(k,17) = \lvert\Gamma^-(17)\rvert$	Successors: Nodes 18 and 16
2	The attacker scans RDP port 3389	Nodes 20 and 22	$f(20,19)$=1 and $f(21,19)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(19)} f_t(k,19) = \lvert\Gamma^-(19)\rvert$ $f(22,17)$=1, $f(23,17)$ = 1 and $f(18,17)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(17)} f_t(k,17) = \lvert\Gamma^-(17)\rvert$	Successors: Nodes 18 and 16
3	The adversary scans and discovers the SCADA vulnerability	Nodes 38, 40 and 46	$f(38,37)$=1 and $f(16,37)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(37)} f_t(k,37) < \lvert\Gamma^-(37)\rvert$ $f(40,39)$=1 and $f(6,39)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(39)} f_t(k,39) < \lvert\Gamma^-(39)\rvert$ $f(46,48)$=1, $f(47,48)$=0 and $f(49,48)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(48)} f_t(k,48) < \lvert\Gamma^-(48)\rvert$	Predecessors : 38, 40 and 46
4	Scanning of port 22 on openplc	Nodes 5, 10, 27, 44, 58	$f(10,9)$=1 and $f(11,9)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(9)} f_t(k,9) < \lvert\Gamma^-(9)\rvert$ $f(27,26)$=1 and $f(28,26)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(26)} f_t(k,26) < \lvert\Gamma^-(26)\rvert$ $f(5,4)$=1 and $f(6,4)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(4)} f_t(k,4) < \lvert\Gamma^-(4)\rvert$ $f(54,43)$=1, $f(8,43)$=0 and $f(45,43)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(43)} f_t(k,43) < \lvert\Gamma^-(43)\rvert$ $f(44,7)$=1, $f(8,7)$=0 and $f(45,7)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(7)} f_t(k,7) < \lvert\Gamma^-(7)\rvert$	Predecessors: 5, 10, 27, 44 and 54
5	MITM attack between openplc and conv-prod-in	Nodes 5, 10, 27, 44, 54 and 8	$f(10,9)$=1 and $f(11,9)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(9)} f_t(k,9) < \lvert\Gamma^-(9)\rvert$ $f(27,26)$=1 and $f(28,26)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(26)} f_t(k,26) < \lvert\Gamma^-(26)\rvert$ $f(54,43)$=1, $f(8,43)$=1 and $f(45,43)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(43)} f_t(k,43) = \lvert\Gamma^-(43)\rvert$ $f(44,7)$=1, $f(8,7)$=1 and $f(45,7)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(7)} f_t(k,7) = \lvert\Gamma^-(7)\rvert$ $f(5,4)$=1 and $f(6,4)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(4)} f_t(k,4) = \lvert\Gamma^-(4)\rvert$ $f(5,55)$=1 and $f(42,55)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(55)} f_t(k,55) = \lvert\Gamma^-(55)\rvert$	Predecessors: Nodes 10 and 27 Successors: Nodes 6, 42 and 3
6	Successful RCE on the SCADA	Nodes 38, 40, 46, and 36	$f(38,37)$=1 and $f(16,37)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(37)} f_t(k,37) < \lvert\Gamma^-(37)\rvert$ $f(40,39)$=1 and $f(6,39)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(39)} f_t(k,39) < \lvert\Gamma^-(39)\rvert$ $f(46,48)$=1, $f(47,48)$=0 and $f(49,48)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(48)} f_t(k,48) < \lvert\Gamma^-(48)\rvert$ $f(36,35)$=1, $f(46,35)$=1 and $f(47,35)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(35)} f_t(k,35) = \lvert\Gamma^-(35)\rvert$	Predecessors: 38, 40 and 46 Successor: 28
7	MITM attack between open-plc and convoyer conv-norm	Nodes 56, 5 and 3	$f(5,4)$=1 and $f(6,4)$=0 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(4)} f_t(k,4) < \lvert\Gamma^-(4)\rvert$ $f(56,2)$=1, $f(3,2)$=1 and $f(57,2)$=1 $\Gamma\Rightarrow \sum_{k\in\Gamma^-(2)} f_t(k,2) < \lvert\Gamma^-(2)\rvert$	Predecessor: Node 5

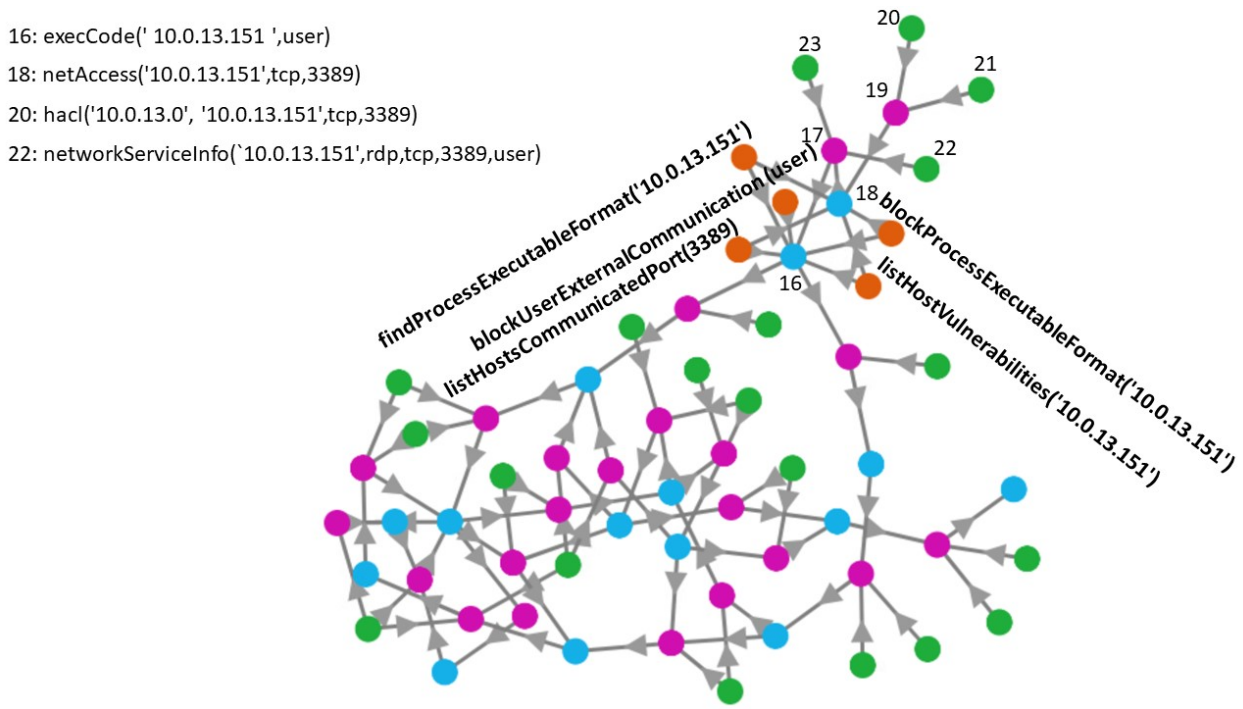


Figure 6.18 ADG generated when the Bluekeep vulnerability is exploited

for the predicate of node 16 in Table 6.7.

Attack Path 2 Figure 6.18 also represents the ADG generated for the exploitation of BlueKeep vulnerability, whose existence is represented by node 23. When the exploitation of the Bluekeep vulnerability is detected, the ADG generator matches nodes 20, 22, and 18 with the alert generated; the countermeasure nodes are still instantiated on nodes 18 and 16.

Attack Path 3 Once the attacker connects to the operator workstation, he/she scans the 10.0.18.0/24 and discovers that the SCADA with IP address 10.0.18.100 is vulnerable to CVE-2021-0708. The generated alert for the scanning of port 8080 on the SCADA is matched with nodes 38, 40 and 46 whose predicates are respectively *hacl('10.0.13.151','10.0.18.100',tcp, 8080)*, *hacl('10.0.13.30','10.0.18.100',tcp,8080)* and *networkServiceInfo('10.0.18.100',scadabr, tcp,8080,lea)*. The flow value entering the successors of these nodes is 1; it is less than the number of predecessors. Then, the ADG generators instantiate countermeasure nodes on nodes 38, 40, and 46 as shown in Figure 6.19 and on node 23, representing the Bluekeep vulnerability because it is the only vulnerability for which the patch of vulnerability is included in the generated playbook.

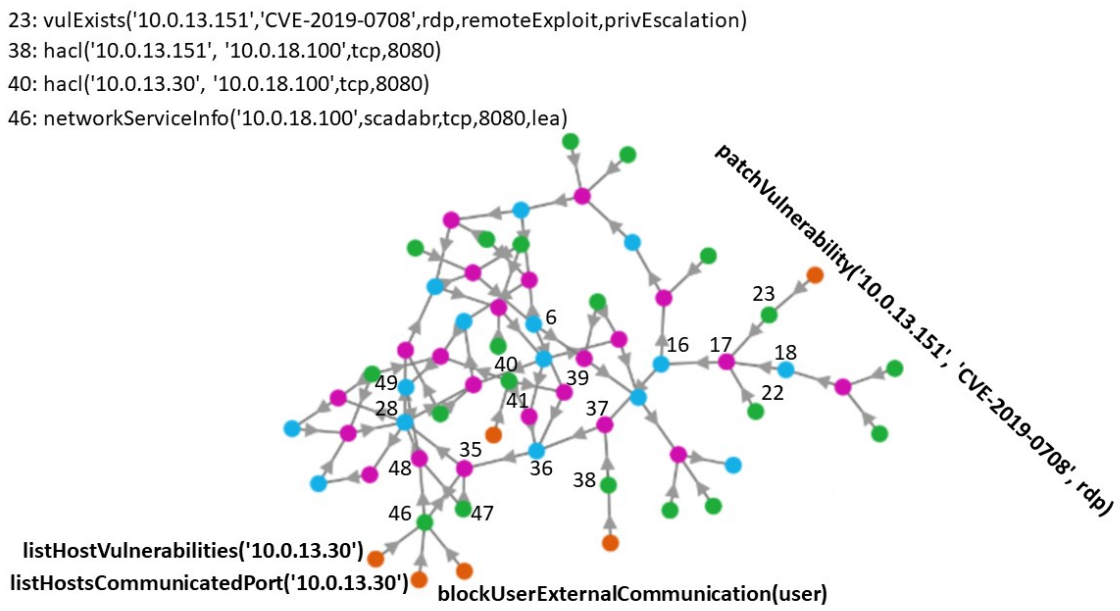


Figure 6.19 The ADG generated for the scanning of port 8080 on the SCADA

Attack Path 4 The exploitation of vulnerability CVE-201-26828 on the SCADA requires user credentials. As the adversary does not have this credential, he/she installs a keylogger on the SCADA. The network scan of network 10.0.13.0/24 lets the adversary know that the openplc and the conv-prod-in are vulnerable to an MITM attack. By launching the MITM attack, a DoS will be caused in the process; then, an administrator will connect to the SCADA, allowing the adversary to steal his/her credentials.

The alert generated for the Scan of port 22 on the OpenPLC is matched with nodes 5, 10, 27, 44, and 54 whose predicates are respectively *hacl('10.0.13.30', '10.0.13.37',tcp,22*, *hacl('10.0.13.32','10.0.13.30',tcp,22)*, *hacl('10.0.18.100', '10.0.13.30',tcp,22)*, *networkServiceInfo('10.0.13.30',openssh,tcp,22,user)* and *networkServiceInfo('10.0.13.30',openssh,tcp,22, lea)*. The flow value from those nodes to their successors is then 1. As shown in Figure 6.20, the flow value from these nodes to their successors is less than the number of successors of the successors; the ADG generator instantiates countermeasure nodes on this node and node 23 representing the Bluekeep vulnerability because it is the only vulnerability for which the patch of vulnerability is included in the generated playbook.

Attack Path 5 The alert generated for launching the MITM attack is matched with nodes 5, 10, 27, 44, 54, and 8. The predicate of node 8 is *netAccess('10.0.13.30',tcp,22)*. The flow

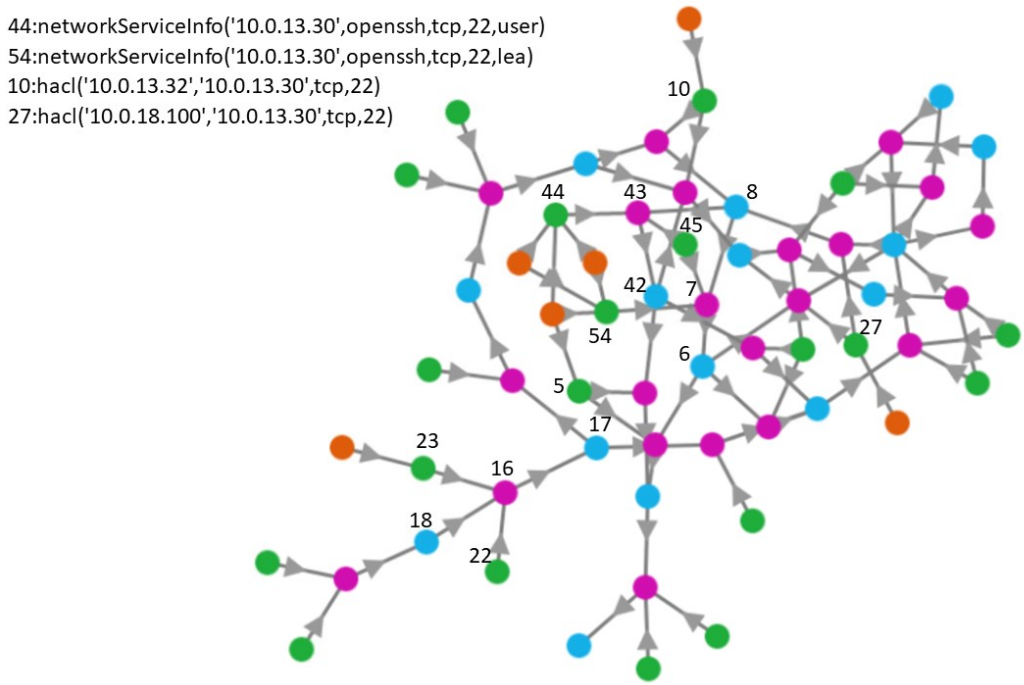


Figure 6.20 The ADG generated for the scanning of port 22 on the openplc

value from all these nodes changes to 1. Node 45 represents the vulnerability with ID CVE-2021-26828 on the OpenPLC, and its flow value is 1. Nodes 8, 45, and 44 have node 7 as a successor. Nodes 8, 45, and 54 have node 43 as a successor. Then, the flow value entering nodes 7 and 43 is the same as their number of predecessors. Therefore, the ADG generator instantiates countermeasure nodes on their successor, nodes 6 and 42. Then, the total flow value entering nodes 4 and 55 is the same as their number of predecessors, so countermeasures nodes are instantiated on their common successor node 3. Figure 6.21 represents the ADG obtained for this attack.

Attack Path 6 Once the adversary obtains the user credentials, he/she launches the RCE on the SCADA. The alert generated for this attack is correlated with nodes 38, 40, 46, and 36. The predicate of node 36 is *netAccess('10.0.18.100',w tcp,8080)*. The flow value from these nodes changes to 1. Node 47 represents a vulnerability whose ID is CVE-2021-26828 on the SCADA. Then, the flow value from node 47 to node 35, representing the inference made by nodes 47 and 36, is 1. The total flow value entering node 35 is 2, the same as its number of predecessors. The countermeasure nodes should be instantiated on its successor, node 28. The total flow value entering node 48, representing the inference of nodes 46 and 49, is 1 because the flow value outgoing from node 49 is still 0. Then, countermeasures nodes

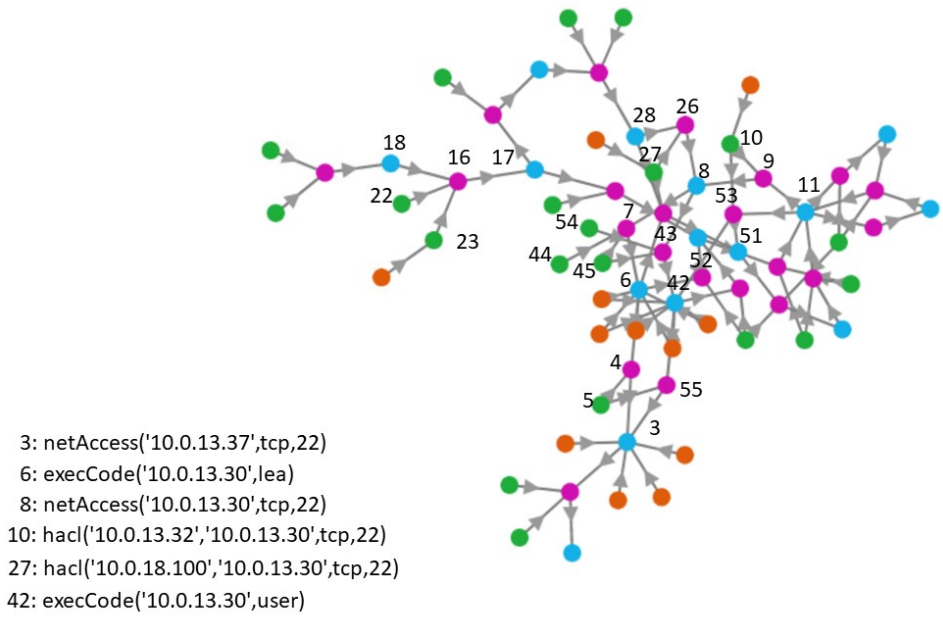


Figure 6.21 The ADG generated when the MITM attack between openplc and conv-prod-in is successful

are instantiated on node 46. It is the same for nodes 36 and 40, which are predecessors of inference nodes with other predecessors whose outgoing flow value is 0. Figure 6.22 represents the ADG generated for the successful exploitation of CVE-2021-26828.

Attack Path 7 The attacker is connected to the SCADA as root. Then, the adversary can launch an MITM attack between the OpenPLC and the conveyor conv-norm, paralyzing the industrial process. The alert generated for this attack matches nodes 56, 5 and 3 whose predicates are respectively *networkServiceInfo('10.0.13.37', openssh,tcp, 22, lea)*, *hacl('10.0.13.30','10.0.13.37', tcp, 22)* and *netAccess('10.0.13.37', tcp, 22)*. The flow value from these nodes changes to 1. Countermeasure nodes are instantiated on node 5 as the flow value entering its successors is less than its number of predecessors. Node 57 represents the existence of the vulnerability identified by the ID CVE-2023-48795, so the flow value from it is 1. Therefore, the total flow value entering node 2, which represents the inference by nodes 3, 56, and 57, is the same as its number of predecessors. However, its successor node 1 does not have any successor as it is the final goal of the AG; countermeasure nodes are not instantiated on it as it is already too late to react. At this stage, patching the vulnerabilities to put the system in an optimal state is essential. However, the countermeasure node cor-

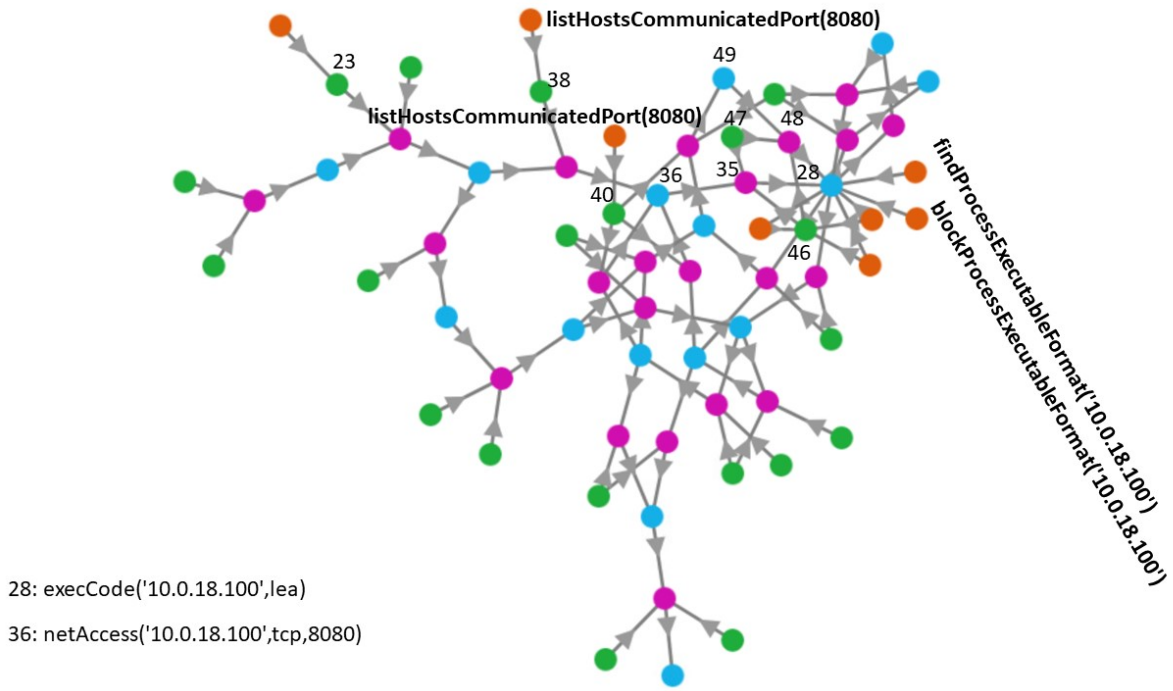


Figure 6.22 The ADG generated when CVE-2021-26828 is exploited successfully on the SCADA

responding to that is only instantiated on the blueKeep vulnerability because the generated playbook does not propose this action for the other vulnerabilities. Figure 6.23 represents the ADG obtained for this MITM attack.

6.10 Relevance Evaluation

6.10.1 Scenario 1

This section assesses the significance of the security of the countermeasure predicates on the AG nodes where they are applied.

Countermeasures set 1 The description of the AG predicate *netAccess(_machine, _protocol, _port)* follows: **a user can send packets to a port on a machine through a protocol**. The purpose of this process is to access the target machine. The countermeasures predicated matched to it are *findProcessExecutableFormat(_host)*, *listHostsCommunicatedPort(_port)* and *blockProcessExecutableFormat(_host)*. The port is an AG predicate artifact that helps list all the hosts communicating through it to investigate whether blocking this port can negatively impact the industrial process. Once it is discovered that the process

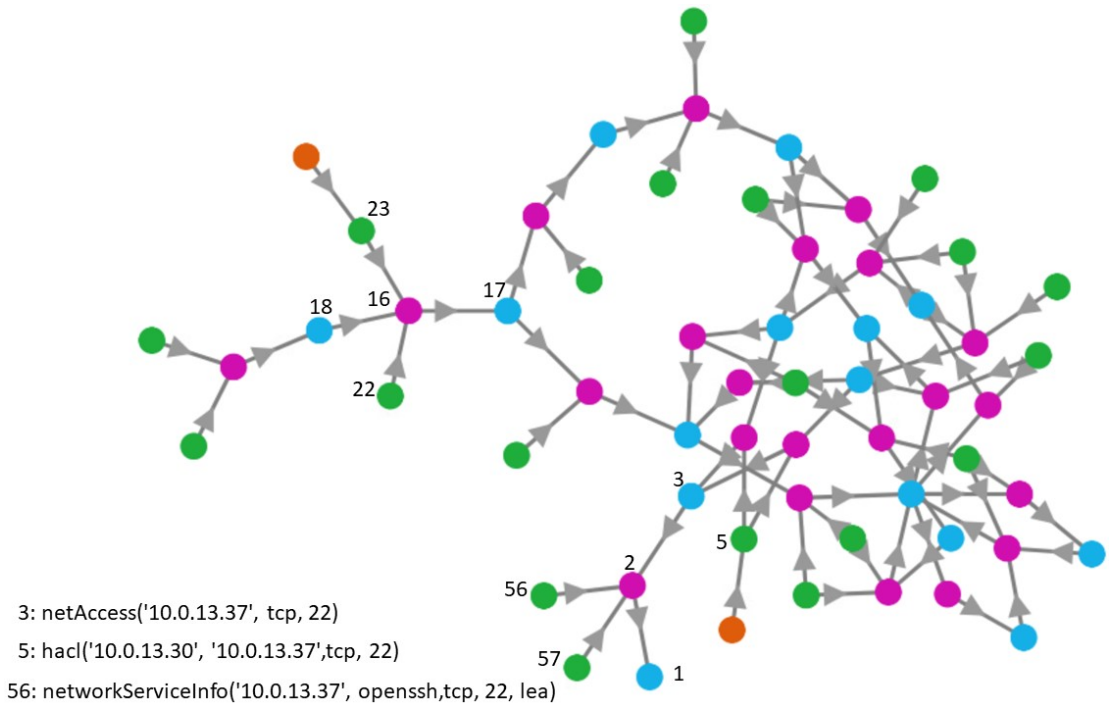


Figure 6.23 The ADG generated after the adversary reach the final goal of the attack scenario

comes from an adversarial user, a containment action will block this process.

Countermeasures set 2 The description for the AG predicate *execCode(_host, _user)* follows: *the attacker can execute arbitrary code on the machine under a user privilege.* The countermeasures predicates instantiated on it are *findProcessExecutableFormat(_host)*, *listHostsCommunicatedPort(_port)*, *blockUserExternalCommunication(_principal)* and *blockProcessExecutableFormat(_host)*. Code execution is a process initiated by an attacker to gain privileges on a system or to cause a denial of service. The port information is obtained from the alert, enabling the listing of all the hosts communicating through that port. The process format can be found in the logs. Consequently, containment actions can be implemented to block the process and prevent external communication for the user whose credentials were used to execute the attack.

6.10.2 Scenario 2

When the generated playbooks are based on security tools

Countermeasures set 1 The orange node in the ADG graph in Figure 6.13 represents the countermeasure nodes. The description of the AG predicate *netAccess('10.0.11.100',tcp,8080)*

is as follows: **a user can send packets to port 8080 on a SCADA machine with IP address 10.0.11.100 through the TCP protocol**. This process aims to access the target machine. The countermeasures predicates of the node instantiated on it include *findProcessExecutableFormat('10.0.11.100')*, *listHostsCommunicatedPort(8080)*, and *blockProcessExecutableFormat('10.0.11.100')*. The port number is an artifact derived from the AG predicate that can be used to identify all hosts communicating through it, helping to evaluate whether blocking this port would negatively impact the industrial process. If it is determined that the process originates from an adversarial user, a containment action will be taken to block it.

Countermeasures set 2 The description for the AG predicate *execCode('10.0.11 .100',user)* is as follows: *the attacker can execute arbitrary code on the machine with IP address 10.0.11.*
100 under a user privilege. The countermeasures predicates of the nodes instantiated on it are *findProcessExecutableFormat('10.0.11. 100')*, *listHostsCommunicatedPort(8080)*, *blockUser ExternalCommunication (user)* and *blockProcessExecutableFormat('10.0.11.100')*. Code execution is a process initiated by an attacker to obtain privileges on a system or to induce a denial of service. The port information is obtained from the alert. Then, listing all the hosts communicating through this port is possible. Subsequently, containment actions can be initiated to block the process and prevent external communication for the user whose credentials were used to execute the attack.

Countermeasures set 3 The description of the predicate *hacl('10.0.11.100', '10.0.11.72', tcp,22)* is as follows: "the network allows '10.0.11.100' to access '10.0.11.72' through protocol TCP and port 22". The instantiated countermeasure node whose predicate on it is *listHostsCommunicatedPort(22)* in Figure 6.14. Even though the received alert pertains to the attacker from the 10.0.20.0 network accessing the SCADA machine via SSH, the tool maps it to the predicates related to a tentative connection from host 10.0.11.100 (the SCADA) to host 10.0.11.72 (the crane). Once the adversary has access to the SCADA machine, he/she can access the crane or other devices on the network 10.0.11.0 and launch MITM attacks. The alert has information about the port impacted. In fact, from 10.0.11.100, it is possible to list all the hosts communicating through port 22 with it, thanks to the alert information. However, since the router does not capture the traffic within the network, no additional incident response actions have been proposed, representing this experiment's limitation.

When the default playbook is generated

Countermeasures set 1 In Figure 6.15, the countermeasure node of predicate *listHost Vulnerabilities('10.0.11.100')* is instantiated on the AG node with predicate *execCode('10.0 .11.100',user)*. This countermeasure involves identifying the host's vulnerabilities, which can be patched once discovered. The countermeasure nodes for predicates *patchVulnerability('10.0.11.100', 'CVE-2021-26828', scadabr)* and *patchVulnerability('10.0.11.72', 'CVE-2023-48795', openssl)* are instantiated on nodes 13, whose predicate is *vulExists('10.0.11.100', 'CVE-2021 -26828',scadabr,remoteExploit,privEscalation)*, and 24, whose predicate is *vulExists('10.0.11.72','CVE-2023-48795',openssh,remoteExploit,privEscalation)*, respectively. The list of vulnerabilities is observed to be suggested only for device 10.0.11.100, as it is the host associated with the alert. The solution consistently recommends mitigation actions for the device where the adversary is detected. At the same time, it proposes patching the vulnerabilities if this countermeasure predicate is in the predicates table. However, vulnerability patching is not the most urgent action to do in real time when an attack occurs. The primary objective of the IR plan is to prevent the adversary from causing further damage to the system and to minimize the attack's impact.

Countermeasures set 2 In Figure 6.16, countermeasure nodes are instantiated solely on nodes that indicate the existence of vulnerabilities; they consist of patching vulnerabilities. Patching vulnerabilities is the only countermeasure proposed as the default playbook does not contain any IR action matching the AG predicate *hacl('10.0.11.100','10.0.11.72',tcp,22)*. This is because the default playbook is suggested when there is insufficient information about the security tools available in the system. It represents a trade-off between taking no action and making the best effort with the information available.

6.10.3 Scenario 3

Countermeasures Set 1 The orange nodes in the ADG represented in Figure 6.18 and Figure 6.21 represent the countermeasure nodes. For example, the description of the AG predicate *netAccess('10.0.13.151',tcp,3389)* is as follows: **a user can send packets to port 3389 on a the operator machine with IP address 10.0.13.151 through the tcp protocol**. This process aims to access the target machine. The countermeasures predicated instantiated on it are *findProcessExecutableFormat('10.0.13.151')*, *blockProcessExecutableFormat('10.0.13.151')*, *listHostsCommunicatedPort(3389)* and *listHostVulnerabilities('10.0.13.151')*. The port number is an artifact gained from the AG predicate that can be used to list all the hosts communicating through it to figure out if blocking this port would

have a negative impact on the industrial process. Once discovering that the process comes from an adversarial user, a containment action will block this process. Listing the vulnerabilities is important to discover which other vulnerabilities the adversary could exploit on this host.

Countermeasures Set 2 The description for the AG predicate *execCode('10.0.13 .151',user)* is as follows: **the attacker can execute arbitrary code on the machine with IP address 10.0.13.151 under a user privilege**. The countermeasure predicates instantiated on it are *findProcessExecutableFormat('10.0.13. 151')*, *listHostsCommunicatedPort(8080)*, *blockUserExternalCommunication (user)*, *listHostVulnerabilities('10.0.13.151')* and *blockProcessExecutableFormat('10.0.11.100')*. The execution of code is a process launched by an adversary to gain privilege on a system or to cause denial of service. The port information is received from the alert. Then, it is possible to list all the hosts communicating through this port using this information. The process format can be found in the logs. Then, the containment actions can be launched to block the process and external communication for the user whose credentials are used to launch the attack. The user name is an artifact of the AG predicate *execCode('10.0.13.151',user)* that is useful to block all external communication for this user. So the user logged on to the vulnerable machine cannot connect to machines on the network 10.0.18.0/24, preventing the attacker from remotely installing the keylogger on the SCADA.

Countermeasures Set 3 The predicate corresponding to the vulnerability patching *patch Vulnerability('10.0.13.151, 'CVE-2019-0708', rdp)* is not instantiated on Figure 6.18 because the adversary already exploits it; the most urgent action consists in blocking the adversary from going further in the network. However, it is proposed to the operator for the other attack paths. Vulnerability patching would be proposed for all the other vulnerabilities when the attack does not concern them if this action is included in the generated playbook.

Countermeasures Set 4 In Figure 6.19, Figure 6.22 and Figure 6.20, countermeasures are proposed for the nodes with predicate as *networkServiceInfo(_host, _program, _protocol, _port, _user)*. These countermeasures' predicates for the scanning of port 22 for the OpenPLC are *listHostVulnerabilities('10.0.13.30')*, *listHostsCommunicatedPort('10.0.13.30)* and *blockUserExternalCommunication(user)*. The detection of the scanning allows for the prediction that the attacker is trying to discover the host's vulnerability. It is judicious for the expert to launch a vulnerability and port scan to detect which vulnerabilities the adversary can potentially exploit. It can also be judicious to block external communication from

an incompetent user with other networks.

Countermeasures Set 5 In Figure 6.22, Figure 6.19 and Figure 6.20, a countermeasure node is instantiated on AG nodes whose predicates are as *hacl(_src, _dst, _prot, _port)*. The proposed countermeasure consists of listing the hosts communicated through this port. Then, the expert will know whether it is a critical port before deciding if it is blocking it.

The countermeasures nodes instantiated on the AG represent countermeasures that can be executed automatically without paralyzing the industrial process. The optimal generation of playbooks ensures that the selected playbook makes a good tradeoff between 3 optimization objectives: minimizing loss, complexity, and impact. That is why, for some vulnerabilities, the generated playbook does not include vulnerability patching.

Summary

The ADG generator queries the generated playbooks for all the vulnerabilities of an AG and involves them in correlating the countermeasure predicates with the AG predicates. This approach considers architectures where communication between networks is possible. Then, all the security tools are considered for all the architecture components. Therefore, an IR action predicate from a playbook of any vulnerability of the AG is instantiable on the AG. It is only for the vulnerability patching predicate that the matching with nodes in the AG requires the vulnerability ID. Future work could include patching the vulnerability in the playbook generation by default if an organization considers this IR action mandatory in an IR plan.

The analysis of the relevance of the selected countermeasures for the scenarios demonstrates that no matter the constitution of the infrastructure and the constraints of an organization, the proposed approach is capable of proposing effective IR actions for the automation of the IR playbook that can help in blocking the attacker or recovering the system. The evaluation also helps validate that the generated playbooks are correct and relevant, as they can contribute to recovering the system or blocking the adversary.

Instantiating the countermeasure predicates on the correlative AG predicates is significant for security. This process can favor automating the IR plan. Section 6.12 explores the potential integration of this work into the automation of the IR playbook within a SOAR framework.

6.11 Time Performance Evaluation

This section evaluates the approach's time performance considering several constraints. The first step is to assess whether the IR playbook is generated in real-time upon detection of an adversarial action. In both cases, it is considered that the information necessary for optimal playbook generation is accessible for an ADG with one vulnerability, followed by an ADG with two vulnerabilities. The default playbook is also considered to be proposed for an ADG with one vulnerability, followed by an ADG with two vulnerabilities. The scenario presented in 6.8.2 is considered for the experiments, with the final attack goal being MITM, made possible by exploiting a vulnerability with CVE ID CVE-2023-48795 on the crane device. The scenario in 6.8.1 is considered for one vulnerability. Table 6.9 presents the time, measured in seconds, required to instantiate the countermeasures on the AG in real-time, taking the condition above into account.

Table 6.9 Time performance for instantiating IR actions on real-time on an ADG

	1 vulnerability	2 vulnerabilities	Default playbook for 1 vulnerability	Default playbook for 2 vulnerabilities
Playbook generated before ADG generation	0.159	0.176	0.147	0.162
Playbook generated in real time	58.25	236.44	24.74	67.89

The time required to instantiate the countermeasures increases in a non-polynomial way when the playbook is generated in real time. The time complexity is addressed in Section 6.5. This non-polynomial time increment is because the playbook generator generates combinations of all the candidate IR actions. Nevertheless, the playbook generator identifies the optimal playbook by regulating its impact on the system, its execution complexity, and the potential losses regarding integrity, availability, and confidentiality.

6.12 Discussion and Conclusion

A real-time ADG generation method is proposed, involving the instantiation of IR action predicates onto AG predicates. Three industrial case scenarios are used to validate this

approach. This solution leverages the interrelationship between AG predicates and countermeasure predicates to determine the appropriate countermeasure to apply when the system receives an alert about the adversary activity.

An alert can match multiple AG nodes, setting their flow values to 1. This solution prioritizes countermeasure predicates instantiation based on the attacker's progress within the AG. Typically, when the total flow value reaching a node equals the number of its predecessors, urgent mitigation actions must prevent the attacker from advancing to the node's successors, and countermeasure predicates are instantiated to the successors. However, suppose the total flow value reaching a node is lower than the number of its predecessors. In that case, the mitigation action should focus on lowering the flow from the predecessors to 0, with countermeasure predicates instantiated to the predecessors.

The countermeasure predicates take the AG node predicates parameters. These parameters represent the system artifact information. This process suppresses IR actions from the generated playbook if the necessary artifact information is unavailable. It also ensures that the IR actions are matched with the correct artifacts needed for automated execution by a SOAR system. The SOAR operator must only configure the system to distinguish which parameters from the IR action predicates related to artifacts should be sent to the module responsible for automated execution. Additionally, the IR playbook ontology [36], which communicates with the SOAR via a REST API, can specify the security tool instance needed to carry out each IR action. Thus, the SOAR can leverage its integrations to execute actions on the associated security tools directly.

The proposed real-time ADG generation approach is highly adaptable, as it is based on countermeasures instantiation on an AG. The instantiated countermeasures are derived from an optimal IR playbook that can be generated a priori or in real-time, depending on the organization's needs. The time complexity of the playbook generation process, while non-polynomial, can be managed effectively. Real-time playbook generation for ADG generation is a viable option for organizations with space and memory constraints. For those with sufficient resources, it is recommended to launch the playbook generation when the first AG is generated, demonstrating the adaptability of the approach.

The proposed algorithm is designed to create a countermeasure node for vulnerability patching on vulnerability nodes, even if the alert does not directly match the vulnerability, as long as the vulnerability is included in the playbook. Consequently, the ADG generator consistently searches for generated playbooks addressing all vulnerabilities within an AG. Furthermore, the default playbook's IR actions include listing and patching vulnerabilities on hosts. The approach's time complexity ,regarding the number of vulnerabilities, is $O(V)$ so

no matter the number of vulnerabilities on the AG, the ADG generation process is relatively fast.

In this approach, only countermeasures relevant to the system are instantiated on the AG. However, countermeasures not directly related to the system, such as those in the Preparation and Lessons Learned stages, are left to the expert's discretion to determine when and which actions should be implemented.

The approach is evaluated for different use case scenarios, ensuring it is adaptable to different infrastructures and attacks.

CHAPTER 7 CONCLUSION

As cyberattacks against cyber-physical systems and information systems increase, it is essential to provide solutions to prevent cyberattacks and also to respond to their occurrence faster. In this thesis, four main contributions are proposed to solve this issue. The first contribution consists of generating and enriching LAGs automatically using a vulnerability ontology to boost the prevention of cyberattacks. The second contribution is automating countermeasure selection to match the vulnerability ontology with D3FEND. The third contribution is the automated generation of optimal playbooks based on the countermeasures selected from the second contribution. Finally, the fourth contribution consists of instantiating countermeasures on AG leading to ADG generation.

The conclusion chapter organization follows. In Section 7.1, The contributions of this work are summarized. Section 7.2 overviews the limitations of this work. Section 7.3 outlines directions for future research.

7.1 Summary of Works

Chapter 3 demonstrates how ontology contributes to updating LAGs based on real-time generated alerts. LAGs help represent the paths an adversary can follow to cause damage to a system by expressing a pre and post-condition relationship between nodes. The achievement of pre-conditions is represented by the AND node, showing the pre-conditions inference, and leads to the post-condition node, representing adversarial actions. The real-time monitoring of the system helps to detect threats and adversarial actions. Their detection leads to the correlation of the alert information with the LAG to detect which vulnerability is exploited or susceptible to being exploited. The solution queries the vulnerability ontology, VDO, to deduce the other potential attack paths the adversary can take to reach the attack goal. Finding a defensive technique that can block the adversary is necessary.

Chapter 4 shows how KGs contribute to automated selecting countermeasures against cyberattacks. Matching VDO with countermeasures KG, D3FEND, helps obtain countermeasures that can contribute to building an IR plan. The cosine similarity between the VDO and D3FEND entities helps determine which defensive technique can counteract a vulnerability impact or an adversary exploitation method. Once countermeasures are selected, they must be organized logically within the IR plan. Several constraints should be taken into account when building an IR plan.

Chapter 5 demonstrates how the countermeasures selected serve as a basis for the automated generation of optimal IR playbooks. The correlation of the selected countermeasures with an IR framework, RE&CT, contributes to getting a list of IR actions that can be involved in the IR playbook generation. The information about the available security tools and the required attacker position for a vulnerability exploitation helps prune the list of IR actions. Combining the pruned IR actions and some defined constraints, such as the maximum and the minimum number of actions a playbook can contain, enables the automated generation of an IR playbook. The Pareto optimality algorithm helps select the optimal playbook by doing a tradeoff between maximizing the impact of the playbook and minimizing its loss and complexity. This approach ensures that all the playbook actions that are released automatically are executable by an accessible security tool. The approach also clearly indicates the actions that need an expert intervention.

Chapter 6 demonstrates how the IR actions of the optimal playbook are automatically instantiated on the AG, leading to an ADG when an alert matches a node of the AG. The flow value of a node impacted by an alert changes from 0 to 1. An alert can concern several nodes; some nodes have an output flow value of 1 since the first generation of the AG. This approach can prioritize where to apply countermeasures based on the total flow value going to an AND node representing the inference of all its predecessor nodes. Thanks to anti-correlation, this approach can correlate the AG nodes with the correct countermeasures, leading to the instantiation of countermeasures nodes on the AG nodes in real time.

7.2 Limitations

This section identifies some of the limitations of this work.

The proposed solution cannot automatically generate new interaction rules based on the input file provided when the logical reasoner finds no attack path. It can only generate new ones by ontology inference in the AG enrichment phase. Therefore, a user should create the new rules manually when necessary.

Although effective, the automated graph matching process is time-consuming. Initiating this process in advance is advisable to avoid any delay in launching the IR plan.

The playbook generation's time complexity is non-polynomial. Launching the playbook generation in real time while instantiating countermeasures on the AG can require more than 216 seconds, depending on the system's size and complexity and the number of vulnerabilities discovered in it.

The graph matching and ADG generation processes are only validated for vulnerabilities with

logical impact, such as privilege escalation, DoS, and information disclosure on CPS. They are not validated for vulnerabilities with physical impact, like physical resource consumption or property damage. Therefore, these contributions cannot be fully ensured to adapt to physical-impact vulnerabilities.

7.3 Future Research

Future work regarding LAG generation involves integrating a Cyber Knowledge Graph (CKG) into the LAG generation. The CKG is generated from a cyber threat Intelligence (CTI) report. Integrating them into the AG generation can provide important information for decision-making based on attack trends in an activity sector or geographical zone. This will ensure that organizations are a step ahead in preventing attacks occurring in their sector of activity or their region.

The CKG can also launch the automated generation of interaction rules, enriching the logical reasoner's knowledge base.

This CKG integration can also simplify the graph-matching process. Structured Threat Information Expression (STIX) is an open-source language and serialization format used to exchange CTI. There is a relationship between a STIX attack pattern object and a STIX vulnerability object. By creating a relationship between the attack pattern object and any class of D3FEND, there will be no need to do the graph and word embedding for matching VDO and D3FEND. The defense techniques from D3FEND will be obtained by querying the STIX JSON object, reducing the time consumed for countermeasure selection.

Future work regarding the IR playbook ontology involves linking the IR actions with commands that can be executed automatically, which will speed up the IR automation process.

Thanks to the artifact information provided by the ADG predicates, the ADG and the IR playbook ontology can be integrated within a SOAR to facilitate the IR automation process.

7.4 Conclusion

This thesis proposes a solution to automatically select IR actions that can be integrated into automating an organization's IR plan based on corporate security tools. A proposed optimal IR playbook generation approach contributes to generating a playbook from an IR framework and the information regarding the security tools of the infrastructure and the attacker's position. The attacker position is obtained from an AG. An approach to automatically generate an AG based on the system scanning information, such as the vulnerabilities

discovered and the network configuration, is proposed. Real-time system monitoring is recommended to deal with the increasing number of logs due to the rise of cyberattacks. The system monitoring contributes to the AG enrichment based on vulnerability ontology. This vulnerability ontology is matched with D3FEND to select the candidate defensive techniques involved in the optimal IR playbook generation process. From the IR actions composing the IR playbook, countermeasure predicates are generated. Therefore, the correlation of the AG predicate with these countermeasure predicates is introduced to choose which countermeasure predicates can counterattack a specific malicious action, leading to the automatic instantiation of countermeasures on the AG when a threat is detected. Therefore, an ADG is generated in real time from the initial AG. The ADG is also updated in real-real based on the new upcoming alerts.

REFERENCES

[1] J. Chigada and R. Madzinga, "Cyberattacks and threats during covid-19: A systematic literature review," *South African Journal of Information Management*, vol. 23, no. 1, pp. 1–11, 2021.

[2] S. A. Khanday, H. Fatima, and N. Rakesh, "Intrusion detection systems for trending cyberattacks," in *Advancing Computational Intelligence Techniques for Security Systems Design*. CRC Press, 2022, pp. 125–142.

[3] M. A. I. Mallick and R. Nath, "Navigating the cyber security landscape: A comprehensive review of cyber-attacks, emerging trends, and recent developments," *World Scientific News*, vol. 190, no. 1, pp. 1–69, 2024.

[4] S. Jajodia, S. Noel, and B. O'berry, "Topological analysis of network attack vulnerability," in *Managing cyber threats*. Springer, 2005, pp. 247–266.

[5] F.-X. Aguessy, L. Gaspard, O. Bettan, and V. Conan, "Remediating Logical Attack Paths Using Information System Simulated Topologies," in *C&ESAR 2014*, ser. C&ESAR 2014, Rennes, France, Nov. 2014, p. 187. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01144971

[6] F.-x. Aguessy, "Dynamic Risk Assessment and Response Computation using Bayesian Attack Models To cite this version : HAL Id : tel-01408035 Évaluation Dynamique de Risque et Calcul de Réponses Basés sur des Modèles d ' Attaques Bayésiens," 2017.

[7] CVE. (1999-2020) Common vulnerabilities and exposures. [Online]. Available: https://cve.mitre.org/

[8] D. Schlette, P. Empl, M. Caselli, T. Schreck, and G. Pernul, "Do you play it by the books? a study on incident response playbooks and influencing factors," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023, pp. 60–60.

[9] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe, "A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0," *University of Manchester*, 2004.

[10] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid,

A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann, "Knowledge graphs," *ACM Comput. Surv.*, vol. 54, no. 4, jul 2021.

[11] S. Peroni, "Samod: an agile methodology for the development of ontologies," in *Proceedings of the 13th OWL: Experiences and Directions Workshop and 5th OWL reasoner evaluation workshop (OWLED-ORE 2016)*, 2016, pp. 1–14.

[12] A. Babar, F. Imam, T. R. Dean, and J. Fernandez, "An approach to represent and transform application-specific constraints for an intrusion detection system," in *Proceedings of the 30th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '20. USA: IBM Corp., 2020, p. 53–62.

[13] Y. Emek, S. Kutten, M. Shalom, and S. Zaks, "Hierarchical b-matching," in *SOFSEM 2021: Theory and Practice of Computer Science*, T. Bureš, R. Dondi, J. Gamper, G. Guerrini, T. Jurdziński, C. Pahl, F. Sikora, and P. W. Wong, Eds. Cham: Springer International Publishing, 2021, pp. 189–202.

[14] L. Ma and Y. Zhang, "Using word2vec to process big text data," in *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 2015, pp. 2895–2897.

[15] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International journal of machine learning and cybernetics*, vol. 1, pp. 43–52, 2010.

[16] P. Ristoski and H. Paulheim, "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*. Springer, 2016, pp. 498–514.

[17] F.-X. Aguessy, O. Bettan, G. Blanc, V. Conan, and H. Debar, "Hybrid risk assessment model based on bayesian networks," in *Advances in Information and Computer Security: 11th International Workshop on Security, IWSEC 2016, Tokyo, Japan, September 12-14, 2016, Proceedings 11*. Springer, 2016, pp. 21–40.

[18] N. Ghosh and S. Ghosh, "A planner-based approach to generate and analyze minimal attack graph," *Applied Intelligence*, vol. 36, no. 2, pp. 369–390, 2012.

[19] H. Shirazi, B. Bezawada, I. Ray, and C. Anderson, "Adversarial sampling attacks against phishing detection," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2019, pp. 83–101.

[20] S. Roschke, F. Cheng, and C. Meinel, "Using vulnerability information and attack graphs for Intrusion Detection," *2010 6th International Conference on Information Assurance and Security, IAS 2010*, pp. 68–73, 2010.

[21] M. Inokuchi, Y. Ohta, S. Kinoshita, T. Yagyu, O. Stan, R. Bitton, Y. Elovici, and A. Shabtai, "Design procedure of knowledge base for practical attack graph generation," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, ser. Asia CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 594–601. [Online]. Available: https://doi.org/10.1145/3321705.3329853

[22] C. Hankin, P. Malacaria *et al.*, "Attack dynamics: An automatic attack graph generation framework based on system topology, capec, cwe, and cve databases," *Computers & Security*, vol. 123, p. 102938, 2022.

[23] K. Falodiya and M. L. Das, "Security Vulnerability Analysis using Ontology-based Attack Graphs," *2017 14th IEEE India Council International Conference, INDICON 2017*, pp. 1–5, 2018.

[24] J. Lee, D. Moon, I. Kim, and Y. Lee, "A semantic approach to improving machine readability of a large-scale attack graph," *Journal of Supercomputing*, vol. 75, no. 6, pp. 3028–3045, 2019. [Online]. Available: https://doi.org/10.1007/s11227-018-2394-6

[25] S. Wu, Y. Zhang, and X. Chen, "Security Assessment of Dynamic Networks with an Approach of Integrating Semantic Reasoning and Attack Graphs," *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, pp. 1166–1174, 2018.

[26] V. Swarup, "Remediation graphs for security patch management," in *IFIP International Information Security Conference*. Springer, 2004, pp. 17–28.

[27] A. Roy, D. S. Kim, and K. S. Trivedi, "Attack countermeasure trees (act): towards unifying the constructs of attack and defense trees," *Secur. Commun. Networks*, vol. 5, pp. 929–943, 2012.

[28] M. Fraile, M. Ford, O. Gadyatskaya, R. Kumar, M. Stoelinga, and R. Trujillo-Rasua, "Using attack-defense trees to analyze threats and countermeasures in an atm: a case study," in *The Practice of Enterprise Modeling: 9th IFIP WG 8.1. Working Conference, PoEM 2016, Skövde, Sweden, November 8-10, 2016, Proceedings 9*. Springer, 2016, pp. 326–334.

[29] B. Kordy and W. Widel, "Efficient attack-defense tree analysis using pareto attribute domains," *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pp. 200–20 015, 2019.

[30] G. Gonzalez-Granadillo, E. Doynikova, J. Garcia-Alfaro, I. Kotenko, and A. Fedorchenko, "Stateful rori-based countermeasure selection using hypergraphs," *Journal of Information Security and Applications*, vol. 54, p. 102562, 2020.

[31] O. Stan, R. Bitton, M. Ezrets, M. Dadon, M. Inokuchi, Y. Ohta, T. Yagyu, Y. Elovici, and A. Shabtai, "Heuristic approach for countermeasure selection using attack graphs," in *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, 2021, pp. 1–16.

[32] D. Ivanov, M. Kalinin, V. Krundyshev, and E. Orel, "Automatic security management of smart infrastructures using attack graph and risk analysis," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, 2020, pp. 295–300.

[33] F. Cuppens, F. Autrel, Y. Bouzida, J. García, S. Gombault, and T. Sans, "Anti-correlation as a criterion to select appropriate counter-measures in an intrusion detection framework," *Annales Des Télécommunications*, vol. 61, pp. 197–217, 2006.

[34] W. Kanoun, N. Cuppens-Boulahia, F. Cuppens, and J. Araujo, "Automated reaction based on risk analysis and attackers skills in intrusion detection systems," in *2008 Third International Conference on Risks and Security of Internet and Systems*, 2008, pp. 117–124.

[35] D. Behbehani, M. Rajarajan, N. Komninos, and K. Al-Begain, "Detecting open banking api security threats using bayesian attack graphs," in *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 2022, pp. 789–796.

[36] K. A. Saint-Hilaire, C. Neal, F. Cuppens, N. Boulahia-Cuppens, and M. Hadji, "Optimal automated generation of playbooks," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2024, pp. 191–199.

[37] P. Cichonski, T. Millar, T. Grance, K. Scarfone *et al.*, "Computer security incident handling guide," *NIST Special Publication*, vol. 800, no. 61, pp. 1–147, 2012.

[38] C. Islam, M. A. Babar, and S. Nepal, "Automated interpretation and integration of security tools using semantic knowledge," in *Advanced Information Systems Engineering:*

31st International Conference, CAiSE 2019, Rome, Italy, June 3–7, 2019, Proceedings 31. Springer, 2019, pp. 513–528.

[39] G. B. Moreira, V. M. Calegario, J. C. Duarte, and A. F. P. dos Santos, "Csiho: An ontology for computer security incident handling," in *Anais do XVIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*. SBC, 2018, pp. 1–14.

[40] Z. Syed, A. Padia, T. Finin, L. Mathews, and A. Joshi, "Uco: A unified cybersecurity ontology," *UMBC Student Collection*, 2016.

[41] H. Hutschenreuter, S. D. Çakmakçi, C. Maeder, and T. Kemmerich, "Ontology-based cybersecurity and resilience framework." in *ICISSP*, 2021, pp. 458–466.

[42] A. A. Ganin, P. Quach, M. Panwar, Z. A. Collier, J. M. Keisler, D. Marchese, and I. Linkov, "Multicriteria decision framework for cybersecurity risk assessment and management," *Risk Analysis*, vol. 40, no. 1, pp. 183–199, 2020.

[43] B. Jordan and A. Thomson, "Cacao security playbooks version 1.0," *OASIS, Committee Specification*, vol. 2, 2021.

[44] P. Empl, D. Schlette, L. Stöger, and G. Pernul, "Generating ics vulnerability playbooks with open standards," *International Journal of Information Security*, pp. 1–16, 2023.

[45] K. A. Saint-Hilaire, C. Neal, F. Cuppens, N. Boulahia-Cuppens, and M. Hadji, "Optimal automated generation of playbooks," in *Data and Applications Security and Privacy XXXVIII*, A. L. Ferrara and R. Krishnan, Eds. Cham: Springer Nature Switzerland, 2024, pp. 191–199.

[46] S. Fenz and A. Ekelhart, "Formalizing information security knowledge," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ser. ASIACCS '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 183–194. [Online]. Available: https://doi.org/10.1145/1533057.1533084

[47] A. Herzog, N. Shahmehri, and C. Duma, "An ontology of information security," *Int. J. Inf. Secur. Priv.*, vol. 1, pp. 1–23, 2007.

[48] BSI. (2004) It grundschutz manual.

[49] P. E. Kaloroumakis and M. J. Smith, "Toward a knowledge graph of cybersecurity countermeasures," Technical report, Tech. Rep., 2021.

[50] K. A. Akbar, S. M. Halim, Y. Hu, A. Singhal, L. Khan, and B. Thuraisingham, "Knowledge mining in cybersecurity: From attack to defense," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2022, pp. 110–122.

[51] F. K. Kaiser, L. J. Andris, T. F. Tennig, J. M. Iser, M. Wiens, and F. Schultmann, "Cyber threat intelligence enabled automated attack incident response," in *2022 3rd International Conference on Next Generation Computing Applications (NextComp)*. IEEE, 2022, pp. 1–6.

[52] M. Rodríguez, G. Betarte, and D. Calegari, "A process mining-based approach for attacker profiling," in *2021 IEEE URUCON*. IEEE, 2021, pp. 425–429.

[53] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, "Fast malware classification by automated behavioral graph matching," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW '10. New York, NY, USA: Association for Computing Machinery, 2010.

[54] Z. Li, J. Zeng, Y. Chen, and Z. Liang, "Attackg: Constructing technique knowledge graph from cyber threat intelligence reports," in *European Symposium on Research in Computer Security*. Springer, 2022, pp. 589–609.

[55] M. Azmy, P. Shi, J. Lin, and I. F. Ilyas, "Matching entities across different knowledge graphs with graph embeddings," *arXiv preprint arXiv:1903.06607*, 2019.

[56] M. Pershina, M. Yakout, and K. Chakrabarti, "Holistic entity matching across knowledge graphs," in *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 2015, pp. 1585–1590.

[57] K. Saint-Hilaire, F. Cuppens, N. Cuppens, and J. Garcia-Alfaro, "Automated enrichment of logical attack graphs via formal ontologies," in *ICT Systems Security and Privacy Protection*, N. Meyer and A. Grocholewska-Czuryło, Eds. Cham: Springer Nature Switzerland, 2024, pp. 59–72.

[58] Q. Hu, M. R. Asghar, and N. Brownlee, "Measuring ipv6 dns reconnaissance attacks and preventing them using dns guard," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 350–361.

[59] X. Ou, S. Govindavajhala, and A. W. Appel, "Mulval: a logic-based network security analyzer," in *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, ser. SSYM'05. USA: USENIX Association, 2005, p. 8.

[60] D. Gonzalez, H. Hastings, and M. Mirakhorli, "Automated Characterization of Software Vulnerabilities," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 135–139.

[61] K. A. Saint-Hilaire, C. Neal, F. Cuppens, N. Cuppens-Boulahia, and M. Hadji, "Matching Knowledge Graphs for Cybersecurity Countermeasures Selection," in *6th International Conference Science of Cyber Security (SciSec)*, ser. Lecture Notes in Computer Science, Copenhagen, Denmark, Aug. 2024. [Online]. Available: https://hal.science/hal-04671226

[62] V. Šulc, "Current ransomware trends," *International Days of Science*, vol. 31, 2021.

[63] A. W. Malik, Z. Anwar, and A. U. Rahman, "A novel framework for studying the business impact of ransomware on connected vehicles," *IEEE Internet of Things Journal*, vol. 10, no. 10, pp. 8348–8356, 2023.

[64] M. A. Mos and M. M. Chowdhury, "The growing influence of ransomware," in *2020 IEEE International Conference on Electro Information Technology (EIT)*, 2020, pp. 643–647.

[65] H. Alshaikh, N. Ramadan, and H. Ahmed, "Ransomware prevention and mitigation techniques," *Int J Comput Appl*, vol. 177, no. 40, pp. 31–39, 2020.

[66] A. Adamov and A. Carlsson, "The state of ransomware. trends and mitigation techniques. 2017 ieee east-west design test symposium(ewdts), 1-8," 2017.

[67] M. Aljaidi, A. Alsarhan, G. Samara, R. Alazaidah, S. Almatarneh, M. Khalid, and Y. A. Al-Gumaei, "Nhs wannacry ransomware attack: Technical explanation of the vulnerability, exploitation, and countermeasures," in *2022 International Engineering Conference on Electrical, Energy, and Artificial Intelligence (EICEEAI)*. IEEE, 2022, pp. 1–6.

[68] D. A. Van Veldhuizen, G. B. Lamont *et al.*, "Evolutionary computation and convergence to a pareto front," in *Late breaking papers at the genetic programming 1998 conference*. Citeseer, 1998, pp. 221–228.

[69] K. A. Saint-Hilaire, C. Neal, F. Cuppens, N. Cuppens-Boulahia, and F. Bassi, "Attack-Defense Graph Generation: Instantiating Incident Response Actions on Attack Graphs," in *IEEE CPS proceedings*, Sanya, China, Dec. 2024, p. to appear. [Online]. Available: toappear

[70] X. Ou, S. Govindavajhala, A. W. Appel *et al.*, "Mulval: A logic-based network security analyzer." in *USENIX security symposium*, vol. 8. Baltimore, MD, 2005, pp. 113–128.

[71] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 336–345. [Online]. Available: https://doi.org/10.1145/1180405.1180446