






Automated Enrichment of Logical Attack Graphs via Formal Ontologies

Kéren Saint-Hilaire^{1,3} , Frédéric Cuppens^{2,3} , Nora Cuppens^{2,3} ,
and Joaquin Garcia-Alfaro¹  

¹ SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, Palaiseau, France
{keren.saint-hilaire, joaquin.garcia_alfaro}@telecom-sudparis.eu

² Polytechnique Montréal, Montréal, Canada

{frédéric.cuppens, nora.boulahia-cuppens}@polymtl.ca

³ Chair CRITiCAL, MITACS, Montréal, Canada

keren-a.saint-hilaire@polymtl.ca

Abstract. Attack graphs represent the possible actions of adversaries to attack a system. Cybersecurity experts use them to make decisions concerning remediation and recovery plans. There are different attack graph-building approaches. We focus on logical attack graphs. Networks and vulnerabilities constantly change; we propose an attack graph enrichment approach based on semantic augmentation post-processing of the logic predicates. Mapping attack graphs 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 us to verify whether changes lead the attacker to the initial goals or cause further damage to the system not anticipated in the initial graphs. We illustrate our approach using a specific cyber-physical scenario affecting smart cities.

Keywords: Cybersecurity · Attack Graph · Defense Graph · Ontology

1 Introduction

Automation of cybersecurity aims to protect critical systems from cyber-attacks, i.e., from illicit activities perpetrated by adversaries who are trying to alter and disrupt normal business processes. Automated remediation of cyber-attacks is a complex task to achieve, specially under real-time constraints [6, 11, 13]. The understanding of attack realization against a system is essential to automate such tasks. This can be accomplished by adapting attack graphs to counter adversarial paths before adversaries perpetrate the final steps of a cyber-attack.

Attack graph can be classified into logical, topological, and probabilistic families [1]. Logical attack graphs represent the adversarial activities as logical predicates, requiring from successful preconditions to be considered as successful,

i.e., they accurately describe how to judge whether the attack was or not successfully perpetrated. Topological families offer a higher-level view of possible attacks in an information system, representing an attack as a way of accessing new resources. Finally, probabilistic families assign probabilities to nodes and attack steps (e.g., using Bayesian theory).

In our work¹, we choose logical attack graphs. The reason for our choice is as follows. Both topological and probabilistic models provide less precision than logical models, f.i., in terms of explainability about attacks' performance. Indeed, logical attack graphs illustrate the causes of the attacks instead of snapshots of the attack steps [8]. This offers several advantages. For instance, the size of the graph increases in a polynomial manner, whereas in other approaches, it can increase exponentially. Moreover, causality relations between adversaries and systems are already represented in the logical statements of nodes and edges in a logical attack graph. In the other approaches, one may go through Boolean variables to identify the cause of an adverse situation that allows adversaries' actions in a stage, increasing processing and inference complexity. In the case of logical attack graphs, exploiting existing vulnerabilities on an asset is the main cause of the attack.

Our work tackles the following question: *how can real-time system monitoring enrich a priori logical attack graphs by considering embedded and implied inferences on expert knowledge bases?* We validate that a posteriori enrichment of the graphs makes it possible to fulfill certain preconditions that were not considered in the initial graph's generation. Semantic information about system vulnerabilities allows us to discover whether the system is now exposed to different situations that can augment the attack surface to newer detrimental events, causing even further damage.

We also conduct experimental work using the following setup. We use a scanner of vulnerabilities to discover and list vulnerabilities in a given monitored system. The results are consumed by MulVAL [9], a logic-based attack graph engine. We add system monitoring using a SIEM (Security Information and Event Management), enhanced with additional tools to trigger and post-process attack alerts. We also instantiate precise attacks to change the state of the system (i.e., exploitation of vulnerabilities) and use a recent implementation of VDO² to enrich the initial attack graph by augmenting the predicates of the initial graph with the semantic data of VDO and the alerts from Prelude-OSS. Alerts trigger a search within the graph and expand those paths related to successful vulnerability exploitation.

The paper is organized as follows. Section 2 provides a background of the subject and some preliminaries on using attack graphs. Section 3 presents our attack-graph enrichment approach. Section 4 provides the experimental results. Section 5 surveys related work. Section 6 concludes the paper.

¹ An early version of this work is available in Ref. [12].

² <https://github.com/usnistgov/vulntology>.

2 Background

2.1 Logical Attack Graph Modelling

We define preliminary concepts, such as Graph, and AND-OR Graph, as underlying requirements for logical attack graph modeling [1].

Definition 1 (Graph). *A Graph is a set V of vertices and a set E of unordered and ordered pairs of vertices, denoted by $G(V; E)$. An unordered pair of vertices is an edge, while an ordered pair is an arc. A graph containing edges alone is non-oriented or undirected; a graph containing arcs alone is called oriented or directed.*

In a directed graph:

- The parent or source of an arc $(v_1; v_2) \in A; v_1 \in V; v_2 \in V$, is v_1 .
- The child or destination of an arc $(v_1; v_2) \in A; v_1 \in V; v_2 \in V$, is v_2 .
- The incoming arcs of a node v are all the arcs for which v is the child:
 $\forall a = (v_1; v) \in A, \text{ with } v_1 \in V$.
- The outgoing arcs of a node v are all the arcs for which v is the parent:
 $\forall a = (v; v_2) \in A, \text{ with } v_2 \in V$.
- The indegree $deg^-(v)$ of a vertex $v \in V$ is the number of arcs in A whose destination is the vertex v : $deg^-(v) = \text{Card}(\{v_i; \forall v_i \in V; (v_i; v) \in A\})$.
- The outdegree $deg^+(v)$ of a vertex $v \in V$ is the number of arcs in A whose destination is the vertex v : $deg^+(v) = \text{Card}(\{v_i; \forall v_i \in V; (v_i; v) \in A\})$.
- A root is a vertex $v \in V$ for which $deg^-(v) = 0$ (no incoming arc).
- A sink is a vertex $v \in V$ for which $deg^+(v) = 0$ (no outgoing arc).

Definition 2 (AND-OR Graph). *An AND-OR graph is a directed graph where each vertex v is either an OR or an AND. A vertex represents a sub-objective, and according to its type (AND or OR), it requires either the conjunction or disjunction of its children to be fulfilled.*

According to Definitions 1 and 2, logical attack graphs are based on AND-OR logical directed graphs. The nodes are logical facts describing adversaries' actions or the prerequisites to carry them out. The edges correspond to the dependency relations between the nodes. Depending on the approach, various operators can be considered in a logical attack graph. The most popular operators are AND and OR. The AND operator describes the achievement's requirement of all the facts of its children for the logical fact of a node to be achieved. The OR operator describes the achievement condition of at least one fact of its children for the logical fact of a node to be achieved.

3 Proposed Approach

After the attack graph generation, using a priori knowledge about vulnerabilities and network data, both networks and vulnerabilities may evolve (i.e., the

configuration of system devices may change, software updates may be enforced, etc.). Hence, the network is not exposed to the same vulnerabilities as at the beginning of the attack graph generation process. It is essential to update the attack graph according to systems' changes. When enriching a logical attack graph, causality relations between adversaries and systems shall be represented in the logical statements of nodes and edges. We propose a logical attack graph enrichment approach based on ontologies to address these requirements.

3.1 Generation of the Attack Graph

Generating a logical attack graph requires the definition of rules describing causality relations. As an example, we consider code execution. Code execution on a machine allows an adversary access to a host. This scenario corresponds to the logical implication detailed by the following rule:

$$\boxed{\text{execCode}(h, a) \rightarrow \text{canAccesHost}(h)}$$

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

$$\boxed{\text{execCode}(h, a) \wedge \text{hasCredentialsOnMemory}(h, u) \rightarrow \text{harvestCredentials}(h, u)}$$

where $\text{harvestCredentials}(h, u)$ describes a series of credentials harvesting on host h , $\text{execCode}(h, a)$ the predicate that an adversary a is executing code on host h , and $\text{hasCredentialsOnMemory}(h, u)$ 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.

3.2 Monitoring the Information System

To update the attack graph based on the real-time state of the system, we need also to monitor the information system. The monitoring process output can get continuously mapped with the initial nodes of the attack graph to find out if a vulnerability is being exploited. The mapped information consists of the port, the IP address, and the device's protocol. The mapping is prioritized based on the severity of the alert and the Common Vulnerability Scoring System (CVSS) score of the CVE. Our approach prioritizes the CVE with the highest CVSS version 2.0 score for an alert that concerns various vulnerabilities to deduce its impacts using a vulnerability ontology. In Table 1, a system contains three vulnerabilities concerning remote desktop protocol and exploitable using port 3389 and protocol TCP. In this case, our approach prioritizes CVE-2019-0708 since its score is higher. Next, we provide more details about this process using semantic information about concrete vulnerabilities.

Table 1. Comparison between different 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

3.3 Vulnerabilities and Ontologies

Vulnerability information is necessary for the attack graph generation and enrichment process. Vulnerability description in standardized databases provides information about the preconditions and post-conditions and practical ways to be exploited. The vulnerability information allows semantically expressing the adversary’s actions toward the adversarial goals, such as pre- and post-conditions. It is also necessary to consider information about exploited vulnerabilities to update logical attack graphs in real time. The vulnerability information is a text written in natural language. This information needs to be transformed into machine-readable text. The use of an ontology is necessary to represent the machine-readable text and thus ensure its homogeneity. An ontology makes it possible to represent a domain in a structured way. The ontology facilitates interoperability between information from the attack graph and the extracted information from vulnerability databases. It is possible to make queries on the ontology to infer new knowledge necessary for the ontological enrichment of the attack graph.

Let us take CVE-2002-0392 as an example. The information from its description: “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” can be classified as represented in Table 2. The information from the table can lead to the construction of an ontology with the classes *CVE-ID*, *Product*, *AttackType*, *Method*, *Impact* and the properties *concernsProduct*, *hasRemoteType*, *hasMethod*, *resultsInImpact*.

Table 2. Classification of CVE-2002-0392 characteristics.

CVE-ID	Product	Remote Type	Method	Impact
CVE-2002-0392	Apache	remote	Code Execution	Privilege Escalation

3.4 Enrichment of Attack Graphs

Algorithm 1 represents our proposed approach for enriching attack graphs based on a vulnerability ontology and monitoring system information. When a threat

exists on a vulnerable component of the monitored system, it is necessary to look through the vulnerability characteristics to find its post-conditions. Depending on the attack goal and the deduced impact, new rules are generated allowing the logical reasoner to find a new path from the additional consequence to the attack goal, see below.

$$\boxed{\begin{array}{l} shutdown(host) \rightarrow physicalDamage(bus) \\ execCode(host, user) \rightarrow shutdown(host) \end{array}}$$

These post-conditions allow the attack graph to be enriched with new paths. For an attack that aims to cause physical damage and an exploited vulnerability that can cause a service interruption that can be shutdown, 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.

Algorithm 1: Attack Graph Enrichment Process

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

4 Experimental Approach

4.1 Use Case Scenario

Next, we describe a use case scenario provided by smart city stakeholders. A denial-of-service attack against a municipality network is perpetrated. Commu-

nication between machines and sensors is interrupted, causing further delays in the city’s transportation service. People fleeing the area start fighting, forcing the authorities to close all transportation services. The violence in public transportation on a given bus affects the health of several passengers. This scenario is taken into account in our experiments.

4.2 Setup

We instantiate the scenario depicted in Fig. 1 to validate our approach. It represents a cyber-physical system monitored by a SIEM, based on Prelude-OSS³. We use a virtual machine representing the starting device of the scenario, another machine to instantiate the breach point, and a third one representing the critical asset.

The rationale of the scenario depicted in Fig. 1 is as follows. An adversary successfully executes arbitrary code on the starting device by connecting remotely through Remote Desktop Protocol (RDP), a network service that provides users with graphical means to control computers remotely). The adversary can then read the memory of the starting device. The administrator’s credentials are saved in the memory of the starting device. Then, the adversary harvests those credentials. We assume the administrator can connect to all the machines in the domain to manage them remotely. Then, an adversary capable of reusing the credentials can log onto the breach point and remotely connect to the critical asset. To eavesdrop network traffic, the adversary perpetrates a DNS Poisoning attack [5]. The adversary also performs integrity attacks to modify application-level information, such as the bus schedule and routes, to perturb the influence of traffic and cause a congestion increase. This causes citizens to take the wrong buses at the wrong time, leading to panic and violence mentioned in Sect. 4.1. In parallel, the adversary reuses the domain credentials to steal some other access keys and impersonate other users (shown in Fig. 1 with steps *Access Keys Stealer* and *User Compromise*).

W.r.t. Fig. 2, we use Nessus Essentials⁴ scanner (Step 1) to discover and list of vulnerabilities in the monitored system. Data from Nessus is consumed by MulVAL [9], a reasoning engine based on logical programming (Step 2), to generate a logic-based attack graph (Step 3). We use Prelude-ELK⁵, an extended version of Prelude-OSS, to monitor the system in real time. When an alert is generated, the procedural processor maps the information from the alert with the attack graph nodes’ information (Step 4) to determine if the alert concerns a discovered vulnerability in the system. The procedural processor queries the impacts of the exploited vulnerability on a vulnerability knowledge graph (Step 5). For a new impact deduced (Step 6), the procedural processor adds a new rules to the logical reasoner (Step 7). The attack graph is regenerated with a new path (Step 8).

³ <https://www.prelude-siem.com/en/oss-version/>.

⁴ <https://www.tenable.com/products/nessus/nessus-essentials>.

⁵ <https://github.com/Kekere/prelude-elk>.

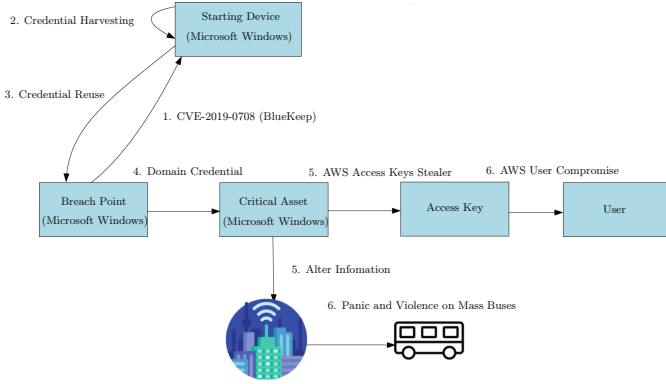


Fig. 1. Cyber-physical attack scenario. An adversary exploits the vulnerability associated with CVE-2019-0708 on a Starting Device. Then, administrator credentials are harvested from the device’s memory and reused by the adversary to take control of a critical asset. The attack affects physical and digital elements associated with the system (e.g., people and services).

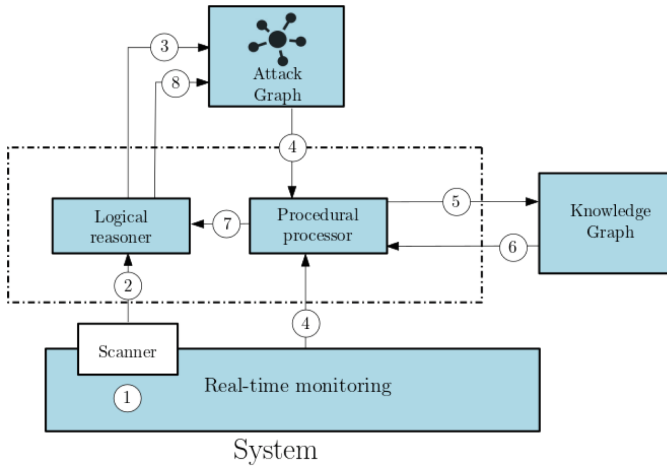


Fig. 2. The Attack Graph Enrichment Process.

MulVAL. Based on the scenario shown in Fig. 1, we create input data for MulVAL and interaction rulesets associated with the vulnerability and the proposed scenario. We encode the new interaction rules as Horn clauses [9]. The first line corresponds to a first-order logic conclusion. The remaining lines represent the enabling conditions. The clauses below correspond to the following statement from the scenario shown in Fig. 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

credentials of the administrator 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).
```

Ontology. We use VDO, an ontology of CVEs proposed by NIST. Figure 3, from [4], represents various attributes of the VDO ontology for characterizing software vulnerabilities. Various features, such as *Impact Method*, and *Logical Impact* are mandatory. *Impact Method* describes how a vulnerability can be exploited. *Logical Impact* describes the possible impacts a successful exploitation of the Vulnerability can have. For each CVE affecting the monitored system, we can fulfill the classes of information from the ontology according to the description and metrics of the CVE.

Prelude-ELK. We use an extended⁶ version of Prelude-OSS’s LML (Log Monitoring Lackey) and third-party sensors such as Suricata⁷ to monitor and process Syslog messages generated from different hosts on heterogeneous platforms. We install Rsyslog Windows Agent⁸ and Suricata on each virtual machine to monitor them with the ELK extension of Prelude-OSS. The results are processed in real-time, mapping the alerts and VDO’s data while conducting our attack graph enrichment process.

Procedural Processor. We create a procedural processor where we upload the input required for the attack graph generation. The engine displays a web visualization of the attack graph. The server matches the last alert’s IP address, port, and protocol with the attack graph. When a vulnerability is likely to be exploited, the engine consults the vulnerability ontology to deduce other impacts of the exposure. The tool generates new rules according to ontology inference.

⁶ <https://github.com/Kekere/prelude-elk>.

⁷ <https://suricata.io/>.

⁸ <https://www.rsyslog.com/windows-agent/>.

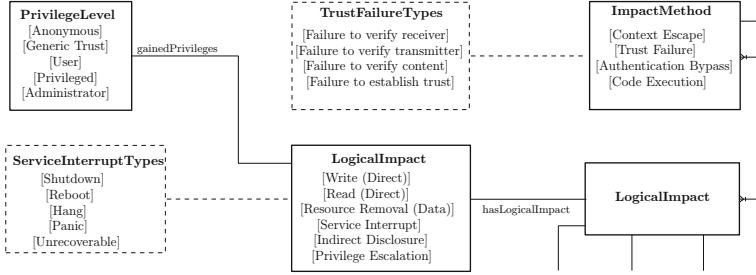


Fig. 3. Sample classes associated to VDO (Vulnerability Description Ontology).

In this paper, we do not consider the impact of an exploited vulnerability on other vulnerabilities in the system nor countermeasures actions to mitigate the attacker’s actions. We will focus on this in future work.

4.3 Results

Figure 4(a), represents the attack graph generated for the scenario depicted in Fig. 1. The goal, represented by Node 1, is to cause panic and violence (see the use-case scenario described in Sect. 4.1). 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; the adversary location is represented by Node 18. Node 20 concerns 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.

In real-time, alerts are processed with Prelude-ELK (see Sect. 4.2). The procedural processor matches precondition nodes’ information with alert information, such as IP address, protocol, and port. The processor makes a query on VDO to deduce other post-conditions associated with *CVE-2019-0708* as represented in Listing 1.1. A list of deduced impacts from the SPARQL query is represented in Table 3. One consequence concerns privilege escalation, and the other concerns interrupting the communication between the affected device and other devices. Important information is not communicated on time for the citizens, including public transportation users. The service interruption can also cause violence on public transportation. Therefore new paths are added to the attack graph based on the deduced impacts. As a result, an enriched attack graph is derived.

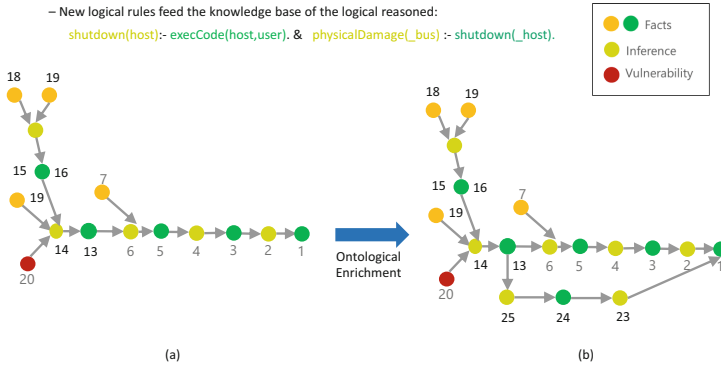


Fig. 4. Sample results. (a) Attack graph generated for the scenario leading to violence on buses. (b) The same attack graph is once enriched with data from the ontology.

```

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 ?vvalid .
  ?vvalid :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 1.1. SPARQL query

Table 3. List of impacts deduced.

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

Figure 4(b) represents such an enriched attack graph. The three nodes highlighted with the red square correspond to the new nodes added to the enriched attack graph 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 satisfied. 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 Fig. 4, the two added nodes represent a new path the adversary can take to cause panic and violence. As we can see, the enriched graph is now acyclic. The new path is shorter than the predicted one. The adversary can reach the goal, represented by Node 1, much sooner than expected. This difference would make operators aware that applying a remediation plan is more urgent. Therefore, the experts can prioritize remediation actions that prevent the more impactful attacker’s actions.

5 Related Work

5.1 Attack Graph Generation Approaches

In [3], Gosh and Gosh propose a planning-based approach for generating minimal attack graphs. The planner generates acyclic paths from information, vulnerabilities, and initial network configuration, combining them, resulting in a minimal attack graph. Minimal attack graphs do not contain redundant nodes and edges. This approach makes it possible to generate an attack graph in polynomial time, regardless of the distribution of vulnerabilities on the attack graph. The initial network configuration and exploit description are the inputs for generating a minimal attack graph using a planner.

Roschke et al. [10] propose an approach to generate logical attack graphs based on logic programming. The input information for the generation of graphs is system and vulnerability information. They present the integration of an IDS in the graph generation process, complementing the process with data fusion and correlation. This improves the quality of the alerts and the quality of their correlation. This correlation makes it possible to prioritize and label alerts. Logic programming-based approaches are more flexible regarding semantic correspondence with other knowledge bases.

Compared to those approaches mentioned above, we monitor the network to update the attack graph based on state change of the network and generate attack graphs based on network information received from Nessus scans. We also enrich the attack graph based on vulnerability information from CVEs and alerts received from a SIEM. We use a logical attack graph generation approach. With a logical approach, the inference is more straightforward. Moreover, the semantics abilities enhance attack graph enrichment with ontology. We use a vulnerability ontology to correlate alerts with the system and vulnerability information.

5.2 Ontology and Attack Graph Generation

Recently, ontologies have been used in different approaches of attack graph generation. Falodiya et al. [2] propose an algorithm that traverses a semantic attack graph to add the information extracted from the graph into an ontology. This ontological approach makes it possible to store other information, such as the countermeasures available for a vulnerability and their cost, as well as the

anti-forensic measures that the attacker, the vulnerability information can use, and the CVSS Score. Analyzing this information using an ontological approach becomes significantly easier as the network size increases.

Lee et al. [7] propose an ontological representation of attack graphs. An ontology that represents attack graphs for a simple network environment is created using RDF schema and OWL. Classes and relationships are created from the multi-requisite graph model. States, vulnerabilities, and prerequisites are represented as classes. This approach improves the machine readability of large-scale attack graphs and thus automates network security assessment. Ontological structures facilitate security assessment. Experts can get the information needed for risk analysis without analyzing the entire attack graph. This task done by the experts can be time-consuming when the graph is large.

In our approach, ontologies contribute to attack-graph enrichment. We use a standardized ontology proposed by the National Institute of Standards and Technology (NIST), the Vulnerability Database Ontology (VDO), for the attack graph enrichment. VDO provides mandatory classes such as *Logical Impact* and *Product*, which we use to map alerts with attack graph nodes. New attack paths can be discovered for a given CVE in VDO. The semantic abilities of logical attack graphs and ontologies also allow us to update the graphs. This improves the automation of the enrichment process (i.e., cybersecurity operators do not have to modify inputs to update the attack graphs manually).

6 Conclusion

We have proposed an ontology-based approach for attack graph enrichment. We use logical graph modeling, in which attacks are represented with predicates. Successful precondition validation means successful attack perpetration. Compared to similar approaches, such as topological and probabilistic attack graphs, our approach simplifies inference since graphs' edges now specify causality. We have implemented the proposed approach using existing software. We have validated the approach based on a cyber-physical use-case proposed by smart-city stakeholders. We have validated the full approach, from generating an initial attack graph (using network vulnerability scans), to enriching the graph (mapping monitoring alerts and ontology semantics in real-time). The predictions of the initial graph are successfully updated into the enriched graph based on inferences by an ontology thanks to expert and monitoring knowledge. In the future, we will evaluate the performance of the proposed approach. We will also focus on presenting remediation actions to block the attacker's actions.

Acknowledgements. We acknowledge financial support from the European Commission (H2020 IMPETUS project, under grant agreement 883286) and the Chair CRIT-iCAL, funded by MITACS (Canada).

References

1. Aguessy, F.-X., Bettan, O., Blanc, G., Conan, V., Debar, H.: Hybrid risk assessment model based on Bayesian networks. In: Ogawa, K., Yoshioka, K. (eds.) IWSEC 2016. LNCS, vol. 9836, pp. 21–40. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44524-3_2
2. Falodiya, K., Das, M.L.: Security vulnerability analysis using ontology-based attack graphs. In: 2017 14th IEEE India Council International Conference, INDICON 2017, pp. 1–5 (2018)
3. Ghosh, N., Ghosh, S.: A planner-based approach to generate and analyze minimal attack graph. *Appl. Intell.* **36**(2), 369–390 (2012)
4. Gonzalez, D., Hastings, H., Mirakhorli, M.: Automated characterization of software vulnerabilities. In: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 135–139. IEEE (2019)
5. Hu, Q., Asghar, M.R., Brownlee, N.: 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), pp. 350–361. IEEE (2018)
6. Kaiser, F.K., Andris, L.J., Tennig, T.F., Iser, J.M., Wiens, M., Schultmann, F.: Cyber threat intelligence enabled automated attack incident response. In: 2022 3rd International Conference on Next Generation Computing Applications (NextComp), pp. 1–6. IEEE (2022)
7. Lee, J., Moon, D., Kim, I., Lee, Y.: A semantic approach to improving machine readability of a large-scale attack graph. *J. Supercomput.* **75**(6), 3028–3045 (2019)
8. Ou, X., Boyer, W.F., McQueen, M.A.: A scalable approach to attack graph generation. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 336–345 (2006)
9. Xinming, O., Govindavajhala, S., Appel, A.W.: MulVAL: a logic-based network security analyzer. In: USENIX Security Symposium, Baltimore, MD, vol. 8, pp. 113–128 (2005)
10. Roschke, S., Cheng, F., Meinel, C.: Using vulnerability information and attack graphs for intrusion detection. In: 2010 6th International Conference on Information Assurance and Security, IAS 2010, pp. 68–73 (2010)
11. Roschke, S., Cheng, F., Schuppenies, R., Meinel, C.: Towards unifying vulnerability information for attack graph construction. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 218–233. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04474-8_18
12. Saint-Hilaire, K., Cuppens, F., Cuppens, N., Garcia-Alfaro, J.: Ontology-based attack graph enrichment. In: 2021 TIEMS (The International Emergency Management Society) Annual Conference, Paris, France (2021). <https://arxiv.org/abs/2202.04016>
13. Stan, O., et al.: Heuristic approach for countermeasure selection using attack graphs. In: 2021 IEEE 34th Computer Security Foundations Symposium (CSF), pp. 1–16 (2021)