Proceedings of the TIEMS Annual Conference, December 2021, Paris, France

ONTOLOGY-BASED ATTACK GRAPH ENRICHMENT

Kéren A Saint-Hilaire, Joaquin Garcia-Alfaro, *Institut Polytechnique de Paris, Télécom SudParis, France*[1]

garcia_a@telecom-sudparis.eu, keren.saint-hilaire@telecom-sudparis.eu

Frédéric Cuppens, Nora Cuppens

Polytechnique Montréal, Canada[2]

frederic.cuppens@polymtl.ca, nora.boulahia-cuppens@polymtl.ca

Abstract

Attack graphs provide a representation of possible actions that adversaries can perpetrate to attack a system. They are used by cybersecurity experts to make decisions, e.g., to decide remediation and recovery plans. Different approaches can be used to build such graphs. We focus on logical attack graphs, based on predicate logic, to define the causality of adversarial actions. Since networks and vulnerabilities are constantly changing (e.g., new applications get installed on system devices, updated services get publicly exposed, etc.), we propose to enrich the attack graph generation approach with a semantic augmentation post-processing of the predicates. Graphs are now mapped to monitoring alerts confirming successful attack actions and updated according to network and vulnerability changes. As a result, predicates get periodically updated, based on attack evidences and ontology enrichment. This allows to verify whether changes lead the attacker to the initial goals or to cause further damage to the system not anticipated in the initial graphs. We illustrate the approach under the specific domain of cyber-physical security affecting smart cities. We validate the approach using existing tools and ontologies.

Keywords: Cybersecurity, Alerts, Vulnerabilities, Attack Graphs, Security Information and Event Management, Ontologies

1. Introduction

Cybersecurity consists of protecting systems, networks, and programs against cyber-attacks that aim to access, modify or destroy sensitive information, extort money from users, or disrupt normal business processes. Cyber-attacks against networks are rising. Computer and network attacks and their countermeasures become more and more complicated [16]. The understanding of attack realization against a system is essential. Attack graphs show possible paths that adversaries can use to reach their goals successfully. There exist various types of attack graph models, mainly [2]: logical, topological, and probabilistic models. Logical models represent an attack as a logical predicate requiring successful preconditions for the attack to be perpetrated. This type of model accurately represents the process by which humans judge whether an attack is either possible or not. Topological models offer a higher-level view of possible attacks in an information system, representing an attack as a way of accessing new resources.

[1] 19 place Marguerite Perey, 91120 Palaiseau

[2] 2500 Chem. de Polytechnique, Montréal, QC H3T 1J4, Canada

This paper is supported by European Union's Horizon 2020 research and innovation programme under grant agreement No. 883286, project IMPETUS (Intelligent Management of Processes, Ethics and Technology for Urban Safety).

Finally, probabilistic models, e.g., using Bayesian theories, assign probabilities to nodes and attack steps. In this paper, we choose to favor logical attack graphs. The rationale for our choice is as follows. Both topological and probabilistic models provide less precision than logical models, f.i., in terms of explainability about how attacks were performed. Indeed, logical attack graphs illustrate the causes of the attacks instead of snapshots of the attack steps [14]. 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, in a logical attack graph, causality relations between adversaries and systems are already represented in the logical statements of nodes and edges. In the other approaches, one may need to go through Boolean variables to identify the cause of an adverse situation that allows adversaries' actions in a stage, hence increasing processing and inference complexity. In the case of logical attack graphs, the exploitation of existing vulnerabilities on an asset is the main cause of the attack. Our work aims to tackle the following question: how can real-time system monitoring enrich a priori logical attack graphs by taking into account vulnerability and network configuration updates? We claim that a posteriori enrichment of the graphs would make possible to fulfill certain preconditions that were not taken into account in the generation of the initial graph. The new process may also allow to discover if the system is now exposed to different situations that can augment the attacks from the initial goals to other detrimental events, causing even further damages. The use of semantic information about system vulnerabilities leads our analysis.

Cybersecurity operators often rely on CVE (Common Vulnerability and Exposure) reports for information about vulnerable systems and libraries to prevent vulnerabilities exploitation. These reports include descriptions, disclosure sources, and manually-populated vulnerability characteristics. Characterizing software vulnerabilities is essential to identify the root cause of the vulnerability, as well as to understand its consequences and attack mechanisms. The use of ontologies [19] is a proper way to represent and communicate facts and relations between multiple agents. Several ontologies describing collections of publicly known software vulnerabilities exist. Examples include the SEPSES (Semantic Processing of Security Event Streams) ontology [11], which describes vulnerabilities extracted from the CVSS[1] (Common Vulnerability Scoring System) database; and the Vulnerability Description Ontology (VDO) [5], proposed by the National Institute of Standards and Technology (NIST), in an effort to characterizing software vulnerabilities in a standard manner. The inference abilities of those existing ontologies justify their use for the enrichment of logical attack graphs. Moreover, it can help to guarantee that the attack graph remains faithful to system updates. This includes processing evidences of vulnerability exploitation, i.e., mapping of monitoring alerts against semantic ontologies.

To validate our approach, we conduct experimental work using the following setup. We use a scanner of vulnerabilities (based on Nessus Essentials[2]) to discover and list vulnerabilities in a given monitored system. The results are consumed by Mulval [15], a logic-based attack graph engine. We add system monitoring, using Prelude-OSS[3], an open-source Security Information and Event Management (SIEM) system, 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 VDO[4] 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 a successful vulnerability exploitation.

Paper Organization — Section 2 provides a background of the subject and some preliminaries on the use of 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] https://www.first.org/cvss/

[2] https://www.tenable.com/products/nessus/nessus-essentials

[3] https://www.prelude-siem.com/en/oss-version/

[4] https://github.com/usnistgov/vulntology

2. Background

2.1 Literature on Attack Graphs

Cyber-attacks are frequently represented in the information security literature as attack graphs. The idea is to identify all those potential paths that an adversary can take in order to perpetuate the exploitation of a series of vulnerabilities and compromise an information system. Different approaches exist, with respect to the way how we can construct and use such attack graphs.

Early literature on attack graphs used them to determine whether specific goal states can be reached by an adversary who is attempting to penetrate a system [13]. The starting vertex of the graph represents the initial state of the adversary in the network. The remaining vertices and edges may represent the actions conducted by the adversary, and the system changes due to the adversary actions. Actions may represent adversarial execution of vulnerabilities in the system. A series of actions may represent the adversary steps toward an escalation of privileges in the system, e.g., to obtain enough privileges on different devices or network components in the system. Actions can be combined using either OR (disjunctive) or AND (conjunctive) logic predicates [17], as well as other attributes, such as the costs associated to the actions, their likelihood and probability of success, etc. In the end, a complete attack graph is expected to show all the possible sequences of actions that will allow the adversary to successfully perpetrate the attack (e.g., to penetrate into the system). Some other representations and uses are possible. For instance, instead of using vertices to represent system states and edges to represent attack actions, we can represent actions as vertices and system states as edges; instead of using single adversary starting locations, we can also assume multiple adversary starting locations or multiple targets and goals, etc.

Other models use directed graphs. For instance, topological attack graphs directly use topological nodes to represent information about systems' assets. The edges represent an adversary's steps to move from a topological parent node to a topological child node. The type of attack (exploitation of a vulnerability, credential thief, etc.) related to an attack step describes how the adversary can move between nodes.

The set of conditions associated with an attack step depends on the type of attack. A sensor can be associated with an attack step, a sensor that may alert that this attack has been detected. Similarly, probabilistic models using Bayesian networks can also represent attack graphs via directed acyclic graphs. Nodes represent random variables and edges represent probabilistic dependencies between variables [3]. An example is BAM (Bayesian Attack Model) [4], which builds upon Bayesian attack trees. Nodes represent transitions, conditions, and sensors. Each node represents a Boolean random variable with two mutually exclusive states. A Bayesian transition node represents the random variable that describes the success or fail of a transition. A Bayesian condition node represents the random variable that describes if the condition is fulfilled. A Bayesian sensor node is a random variable that describes the state of a sensor. These nodes are linked with edges, which indicates that the child node conditionally depends on the state of its parents.

Compared to Bayesian networks, logical attack graphs provide some practical advantages. First, the use of acyclic graphs in Bayesian networks requires from heuristics to suppress paths that an adversary can follow. The inference of a Bayesian attack graph is very complex, since it needs to delve into the Boolean variables and follow several steps upstream to identify the adverse situation causes that enable an adversary's action at a stage. This leads to performance problems. In logical attack graphs, the causality is specified as graph edges. Thus, the inference of a logical attack graph is straightforward. Logical attack graphs are also more elaborated than topological attack graphs. In the sequel, we elaborate further on logical attack graph modeling.

2.2 Logical Attack Graph Modelling

We define next some preliminary concepts such as Graph, Directed Graph, and AND-OR Graph, as underlying requirements for logical attack graph modeling [1, 2].

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.

Definition 2 (Directed Graph) A directed graph G(V; A) consists of a non-empty set V of vertices and a set E of arcs formed by pairs of elements of V. In a directed graph:

- The parent or source of an arc(v_1; v_2) ∈ A; v_1 ∈ V; v_2 ∈ V, is v_1.
- The child or destination of an arc(v_1; v_2) ∈ A; v_1 ∈ V; v_2 ∈ V, is v_2.
- The incoming arcs of a node v are all the arcs for which v is the child:

  ∀a = (v_1; v) ∈ A, with v_1 ∈ V.

- The outgoing arcs of a node v are all the arcs for which v is the parent:

  ∀a = (v; v_2) ∈ A, with v_2 ∈ V.

- the indegree deg (v) of a vertex v ∈ V is the number of arcs in A whose destination is the vertex v: $deg^-(v)$= Card({vi; ∀vi ∈ V; (vi; v) ∈ A}).
- the outdegree $deg^+(v)$ of a vertex v ∈ V is the number of arcs in A whose destination is the vertex v: $deg^+(v)$= Card({vi; ∀vi ∈ V; (vi; v) ∈ A}).
- a root is a vertex v ∈ V for which deg (v) = 0 (no incoming arc).
- a sink is a vertex v ∈ V for which deg+(v) = 0 (no outgoing arc).

Definition 3 (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. A root node n of an ANDOR graph can be called a precondition as it does not require any other node n to be fulfilled.

According to Definitions 1, 2, and 3, logical attack graphs are based on AND-OR logical directed graphs. The nodes are logical facts describing adversaries' actions or the pre-requisites to carry them out. The edges correspond to the dependency relations between the nodes. Various operators can be considered in a logical attack graph depending on the approach. The more popular operators are AND and OR. AND operator describes the achievement's requirement of all the facts of its children for the logical fact of a node to be achieved. OR operator describes the achievement's requirement of at least one fact of its children for the logical fact of a node to be achieved.

3. Our Approach

We assume that after the generation of a (proactive) attack graph, 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 the beginning of the attack graph generation process. It is essential to update the attack graph according to systems' changes. When updating 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. Before moving forward with our approach, we start by introducing a representative use case that will help up to explain the rationale of our approach. Examples based on the use case scenario are used to exemplify how our approach conducts the generation and enrichment of attack graphs, as well as other tasks, such as periodic monitoring and ontology analysis.

3.1 Use Case Scenario

This section describes a use case scenario provided by smart city stakeholders. We provide first the general context associated to the scenario; then we focus more in detail on possible attack consequences described by the stakeholders.

3.1.1 General Description

An infectious disease spreads across multiple continents. Health authorities impose unexpected lockdowns on several countries. When the situation seems over, politicians decide to apply some unpopular restrictions, to prevent new spreading waves of the disease. The population gets furious. Violent groups connected through the internet take it as an opportunity to launch attacks against assets associated to public services. Their goal is to cause panic among the population.

3.1.2 Panic and Violence on a Transportation Service

Politicians decide to engage in another period of lockdown. Protesters are loudly shouting outside a municipal building. Social media respond positively to the movement. A mass of citizens arrives at the location. Public transportation in the area is heavily affected, causing long delays. Tensions and altercations rise with the increase of protesters. A fake alert, pretending to come from the municipality network, forces people to evacuate the area. People get injured. Images and videos of altercations, evacuation, and car fires are posted on social media. At the same time, a denial-of-service cyber-attack against the municipality network is perpetrated. Machines and sensors get out-of-service, causing further delays in the transportation service of the city. People trying to leave the area start fighting, forcing the authorities to close all transportation services. The mass of people in a given bus affects the health of several passengers.

3.2 Generation of the Attack Graph

The generation of 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 to have access to a host. This scenario corresponds to the logical implication detailed by the following rule:

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

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

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

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

Monitoring the Information System

In order to update the attack graph based on the real-time state of the system, we can also monitor the information system. The output of the monitoring process can get continuously mapped with the initial nodes of the attack graph, in order to find out if a vulnerability is being exploited. The mapping between the monitored system and the attack graph is described below:

$$\boxed{\forall n \in N : (vulExists(h, x, y, z) \wedge networkServiceInfo(h, s, p, a, u) \rightarrow F_1}$$

where (vulExists(h, x, y, z) describes the existence of a vulnerability x on host h which allows action y resulting in z. Likewise, networkServiceInfo(h, s, p, a, u) describes that user u has a session open on host h where a given service product p is installed, using port a and protocol p.

The evaluation of a successful mapping implies finding further details about specific vulnerabilities. We propose to use a vulnerability ontology to conduct such a process, represented by F1. Next, we provide some more details about this process using semantic information about concrete vulnerabilities.

3.4 Vulnerabilities and Ontologies

Vulnerability information is necessary for both the attack graph generation process and the updates. Vulnerabilities enable the adversary to take actions towards the initial adversarial goals, or alternative actions affecting the system in different ways. Lists of uniquely identifiers in CVE (Common Vulnerabilities and Exposures), a collection of publicly known software vulnerabilities, are complemented with valuable descriptions about the vulnerability, its preconditions and postconditions, and practical ways to be exploited. The information contained in CVE's descriptions can also lead to other valuable characterizations, for example, impact to the system if the vulnerability is exploited.

An ontology is a formal description of a field of knowledge and is represented by descriptive logic. An ontology brings semantic support and unifies unstructured data. Ontologies have been widely used in the field of cybersecurity, for instance, to represent vulnerability classes and their inner relations. Table 1 shows an example, representing the classification of a given vulnerability listed in CVE (with identifier CVE-2002-0392). Ontologies also offer inference abilities, which we will use to enrich logical attack graphs when the exploitation of a vulnerability is being reported during the monitoring process of a vulnerable system.

Table 1: Classification of CVE-2002-0392 characteristics

CVE-ID	Product	Type	Action	Impact
CVE-20020392	Apache	Remote	Code Execution	Privilege Escalation

3.5 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 postconditions. Those post-conditions enrich the attack graph with new paths. The inference rules allow knowing what those new paths can bring to the attack goal. As an example, Algorithm 2 shows an inference rule based on the definition of a Directed Graph in Definition 2, and deduces the consequence of restarting a device in the scenario of Section 3.1 (i.e., a cyber-attack on a municipality network that takes a given device out-of-service for a while). During the attack, the lack of communication between an application and a server causes a problem in the logistics of the transportation service. There is a delay in the bus service. This scenario is not anticipated in the initial graph. It is necessary to update the graph and add a new path that allows the adversary to reach the goal (i.e., cause panic and violence of people). Updating the graph will help the operators to inform the authorities and explore the best remediation strategy to mitigate the damages as soon as possible. Therefore, it is necessary to define inference rules like the one shown in Algorithm 2, to update the graph in such a situation.

Algorithm 1: Enrichment of a proactive attack graph based on a vulnerability ontology and monitored system information

h_1: A threat exists on a vulnerable component of the monitored system.

h_2: Post-conditions of the exploited vulnerability are found in the ontology.

P_1: Add new path on the attack graph.

$$(h_1 \wedge h_2) \rightarrow P_1$$

Algorithm 2: Inference rule for mass on buses scenario v_1: Node corresponds to reboot of a machine.

v_2: Node corresponds to mass on buses.

The child or destination of an arc $(v_1; v_2) \in A$; $v_1 \in V$; $v_2 \in V$, is v_2.

$$(v_1 \land v_2) \rightarrow (v_1; v_2)$$

The inference rule

is: ——v^1 v_{-2} r $(v_1;$

$v_2)$

4. Implementation

4.1 Setup

In order to validate our approach, we instantiate the scenario depicted in Figure 1. It represents a cyber-physical system monitored by a Security Information and Event Management (SIEM) system, based on Prelude-OSS. 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. We use Nessus Essentials to discover and list vulnerabilities in the monitored system. Data from Nessus is consumed by MulVAL [15], a reasoning engine based on logical programming, to generate a logicbased attack graph. We also use a practical implementation of NIST's Vulnerability Description Ontology (VDO) [5], and Prelude-OSS to map the information contained in VDO into the attack graph, upon reception of Prelude-OSS' alerts.

The rationale of the scenario depicted in Figure 1 is as follows. An adversary succeeds to execute arbitrary code on the starting device by connecting remotely through RDP (Remote Desktop Protocol, a network service that provides users with graphical means to remotely control computers). The adversary can then read the memory of the starting device. The credentials of the administrator are saved in the memory of the starting device. Then, the adversary harvests those credentials. We assume that the administrator can connect to all the machines in the domain, to remotely manage them. Then, an adversary capable of reusing the credentials can log onto the breach point and remotely connect to the critical asset. The adversary also perpetrates a DNS Poisoning attack [9], in order to eavesdrop network traffic. The adversary also perpetrates some integrity attacks, in order 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 taking the wrong buses at the wrong time, leading into the situation of panic and violence mentioned in Section 3.1. In parallel, the adversary reuses the domain credentials to steal some other access keys and impersonate other users (shown in Figure 1 with steps Access Keys Stealer and User Compromise). This parallel scenario leads to the exploitation of other vulnerabilities and an eventual denial-of-service attack.

4.1.1 MulVAL

Based on the scenario shown in Figure 1, we create input data for MulVAL, as well as interaction rulesets associated to VDO. We encode the new interaction rules as Horn clauses [15]. 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 Figure 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 log into the breach point with the administrator credentials 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'*.

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

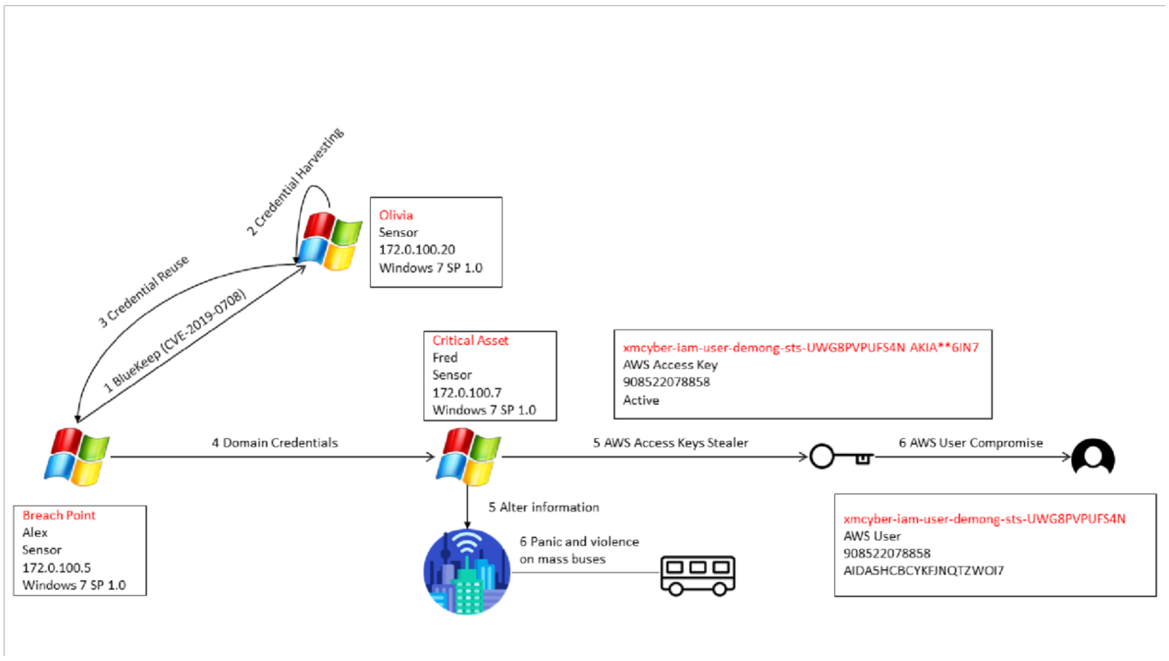


Figure 1: Cyber-physical attack scenario. An adversary exploits the vulnerability associated to CVE2019-0708 on a Starting Device. Then, administrator credentials are harvested from the device's memory, and reused by the adversary to take control over a critical asset. The attack affects both physical and digital elements associated to the system (e.g., people and services).

4.1.2 Ontology

We use VDO, an ontology of CVEs proposed by NIST. Figure 2, from [8], represents various attributes of the VDO ontology, for characterization of software vulnerabilities. Various attributes, such as Attack Threater, Impact Method and Logical Impact are mandatory. The value of Attack Threater characterizes the area or place from which an attack must occur. 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.

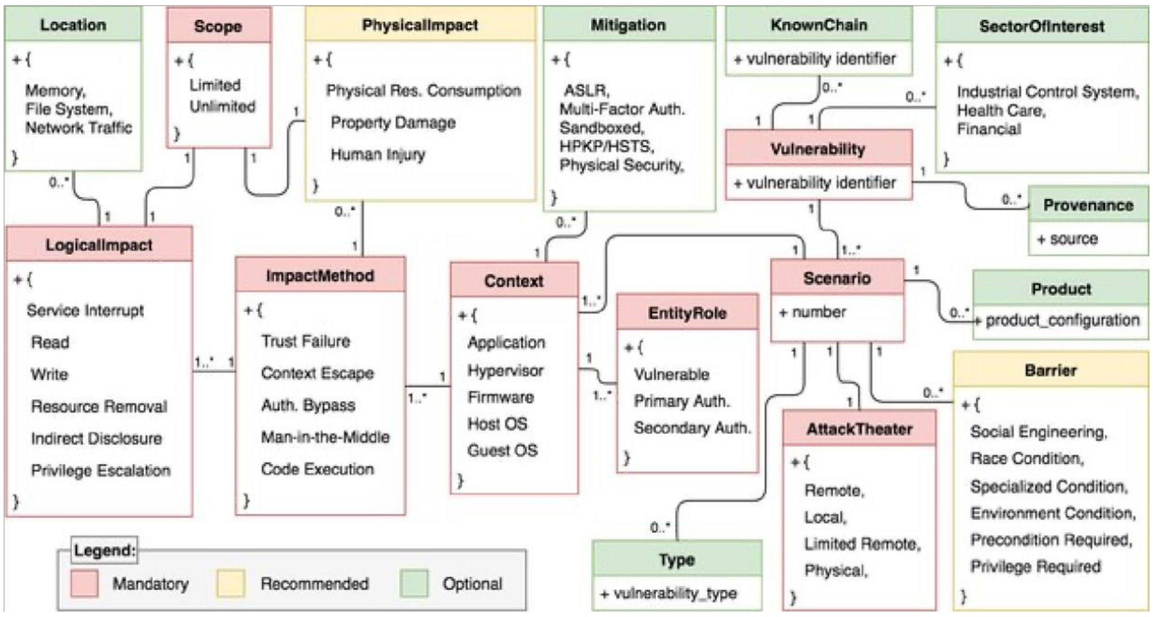

Location
+{ Memory, File System, Network Traffic }
Scope
+{ Limited Unlimited }
PhysicalImpact
+{ Physical Res. Consumption Property Damage Human Injury }
Mitigation
+{ ASLR, Multi-Factor Auth. Sandboxed, HPKP/HSTS, Physical Security, }
KnownChain
+ vulnerability identifier
SectorOfInterest
+{ Industrial Control System, Health Care, Financial }
Vulnerability
+ vulnerability identifier
Provenance
+ source
LogicalImpact
+{ Service Interrupt Read Write Resource Removal Indirect Disclosure Privilege Escalation }
ImpactMethod
+{ Trust Failure Context Escape Auth. Bypass Man-in-the-Middle Code Execution }
Context
+{ Application Hypervisor Firmware Host OS Guest OS }
EntityRole
+{ Vulnerable Primary Auth. Secondary Auth. }
Scenario
+ number
Product
+ product_configuration
Barrier
+{ Social Engineering, Race Condition, Specialized Condition, Environment Condition, Precondition Required, Privilege Required }
AttackTheater
+{ Remote, Local, Limited Remote, Physical, }
Type
+ vulnerability_type
Legend:
Mandatory
Recommended
Optional
0..*
1
1..*

Figure 2: The NIST Vulnerability Description Ontology (VDO) [8]. This figure represents the different classes of NIST vulnerability ontology with their properties.

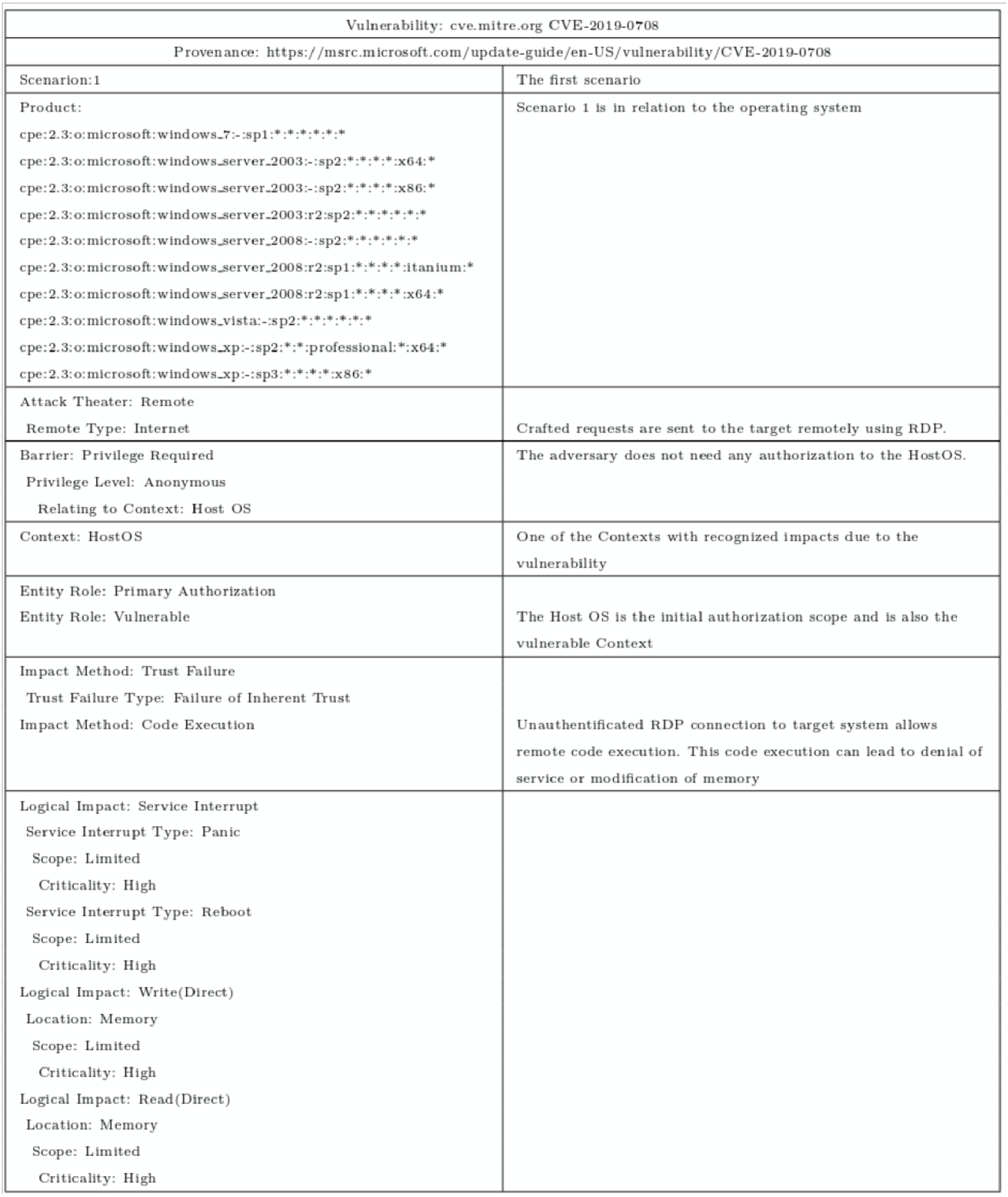
Table 2 corresponds with the attributes associated with *CVE-2019-0708*. A simplified description of *CVE-2019-0708* according to VDO would state, among other details, that '*a remote code execution vulnerability exists in Remote Desktop Services, formerly known as Terminal Services, which can be exploited by an unauthenticated attacker connecting to the target system using TCP or UDP traffic and sending specially crafted requests.'*.

4.1.3 Prelude-ELK

We use an extended version of Prelude-OSS with ELK (Elasticsearch, Logstash, and Kibana). The code is available online[7]. Elasticsearch allows us indexing and processing unstructured data. It also provides a distributed web interface to access the resulting information. Logstash is the parsing engine associated with Elasticsearch for collecting, analyzing, and storing logs. It can integrate many sources simultaneously. Finally, Kibana is a data visualization platform that provides visualization functionalities on indexed content in Elasticsearch. Users can create dashboards with charts and maps of large volumes of data.

The addition of the ELK stack into Prelude-OSS allows the injection and visualization of third-party logs received from both system and network components via TCP/IP messages. The collection of data can still be combined with the classic collection and visualization tools of Prelude-OSS. For instance, we can keep using Prelude's LML (Log Monitoring Lackey) and third-party sensors to monitor and process Syslog messages generated from different hosts on heterogeneous platforms. In addition, we also extend Prelude-OSS with Suricata[8], as a third-party sensor reporting alerts on the exploitation of known vulnerabilities. We install Rsyslog Windows Agent[9] and Suricata on each virtual machine, in order to monitor them with the ELK extension of Prelude-OSS. Logstash inserts the alerts into Elasticsearch (in JSON format). The results are processed in real-time, mapping the alerts and VDO's data while conducting our attack graph enrichment process.

Table 2: Attributes associated with CVE-2019-0708.



Vulnerability: cve.mitre.org CVE-2019-0708	
Provenance: https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2019-0708	
Scenarion:1	The first scenario
Product: cpe:2.3:o:microsoft:windows_7:-:sp1:*:*:*:*:* cpe:2.3:o:microsoft:windows_server_2003:-:sp2:*:*:*:x64:* cpe:2.3:o:microsoft:windows_server_2003:-:sp2:*:*:*:x86:* cpe:2.3:o:microsoft:windows_server_2003:r2:sp2:*:*:*:*:* cpe:2.3:o:microsoft:windows_server_2008:-:sp2:*:*:*:*:* cpe:2.3:o:microsoft:windows_server_2008:r2:sp1:*:*:*:itanium:* cpe:2.3:o:microsoft:windows_server_2008:r2:sp1:*:*:*:x64:* cpe:2.3:o:microsoft:windows_vista:-:sp2:*:*:*:*:* cpe:2.3:o:microsoft:windows_xp:-:sp2:*:*:professional:*:x64:* cpe:2.3:o:microsoft:windows_xp:-:sp3:*:*:*:x86:*	Scenario 1 is in relation to the operating system
Attack Theater: Remote Remote Type: Internet	Crafted requests are sent to the target remotely using RDP.
Barrier: Privilege Required Privilege Level: Anonymous Relating to Context: Host OS	The adversary does not need any authorization to the HostOS.
Context: HostOS	One of the Contexts with recognized impacts due to the vulnerability
Entity Role: Primary Authorization Entity Role: Vulnerable	The Host OS is the initial authorization scope and is also the vulnerable Context
Impact Method: Trust Failure Trust Failure Type: Failure of Inherent Trust Impact Method: Code Execution	Unauthentificated RDP connection to target system allows remote code execution. This code execution can lead to denial of service or modification of memory
Logical Impact: Service Interrupt Service Interrupt Type: Panic Scope: Limited Criticality: High Service Interrupt Type: Reboot Scope: Limited Criticality: High Logical Impact: Write(Direct) Location: Memory Scope: Limited Criticality: High Logical Impact: Read(Direct) Location: Memory Scope: Limited Criticality: High	

4.1.4 Web Interface

We create a web interface using PHP, Javascript, JQuery, D3.JS, and HTML, where we upload the XML outputs of Mulval. The inference engine converts the XML into JSON. From this JSON, the engine displays a web visualization of the attack graph. The server consults the Elasticsearch index in real-time. The last alert's IP address, port, and protocol match the attack graph. When a vulnerability is likely to be exploited, the engine consults the vulnerability ontology to find other post-conditions for the vulnerability. The tool updates the attack graph according to the ontology.

4.2 Results

Figure 3(a) represents the attack graph generated for the scenario depicted in Figure 1. The goal, represented by Node 1, is to cause panic and violence (see use-case scenario described in Section 3.1). A red node represents the existence of a vulnerability on a device. An orange node represents network configuration, e.g., characteristics of a device, connection between two devices in the network, etc. When the preconditions are respected, a yellow node represents the inference rules leading to a fact. Facts are represented by green nodes. For instance, Node 26 represents remote connectivity of the Starting Device in Figure 1, which can be remotely accessed using RDP (Remote Desktop Protocol) services. Node 27 concerns the location of the adversary in the network. Node 25 represents the rule that leads the adversary to gain direct network access (i.e., Node 24) on the starting device, when preconditions on Nodes 26 and 27 are met. Node 29 concerns the existence of the vulnerability CVE2019-0708 on the starting device. CVE-2019-0708 consists of a remote code execution vulnerability. Node 28 concerns the network configuration of the Starting Device. Some other practical details encoded in the graph correspond to the operating systems (Windows 7), open TCP ports (3389), and the identity of the user at the Starting Device (username olivia). Finally, Node 23 has Nodes 24, 28, and 29 as main preconditions.

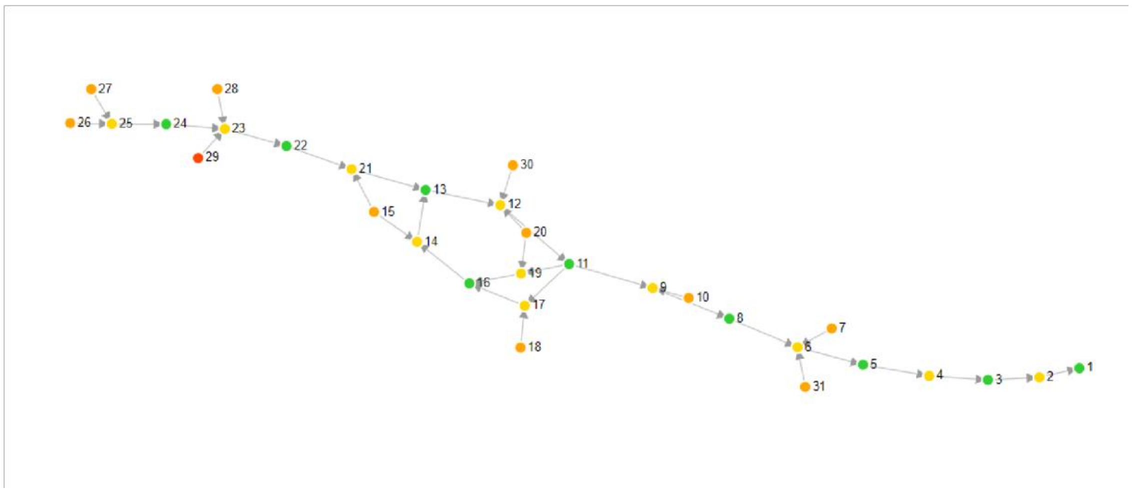
Alerts are processed with Prelude-ELK (cf. Section 4.1.3) in real-time. The inference engine finds exploited nodes based on network information associated to a victim device (i.e., the Starting Device in Figure 1), such as IP address, protocol, and port. The service consults VDO (i.e., our vulnerability ontology) to find other post-conditions associated with CVE-2019-0708. For instance, it compares the operating system associated to the victim device against the list of products listed in CVE-2019-0708.

As a result, an enriched attack graph is derived. Figure 3(b) represents such an enriched attack graph. The four nodes highlighted with the red square correspond to the new nodes added to the enriched attack graph. They represent the logical impacts derived from the ontology. Node 33 describes the Starting Device being restarted. Node 35 describes a system crash of the Starting Device (i.e., Starting Device stops functioning properly). The consequence of Nodes 33 and 35 (i.e., Starting Device restarting or unavailable) leads to the mass on buses scenario (i.e., by inference, Nodes 33 and 35 are targeting now Node 2, which is the rule concerning mass on buses). In Figure 3(b), the four added nodes represent a new path that the adversary can take to cause panic and violence. As we can see, the enriched graph is now an acyclic graph. 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 it is more urgent to apply a remediation plan.

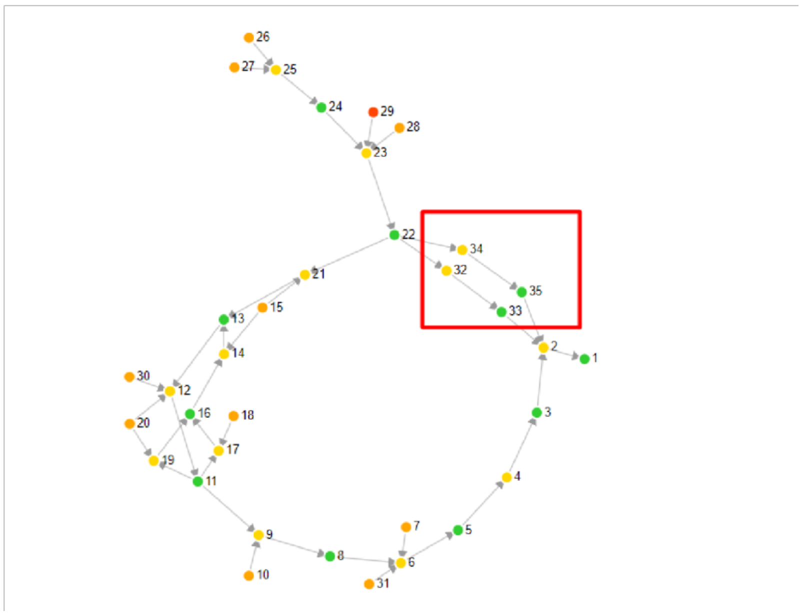
[7] https://github.com/Kekere/prelude-elk

[8] https://suricata.io/

[9] https://www.rsyslog.com/windows-agent/



(a) Initial attack graph. The adversary gains network access on Node 25. When all the preconditions are met on each stage, the adversary can take actions represented by green nodes to reach Node 1, which represents the adversarial goal (i.e., panic and violence on mass buses).



(b) Enriched attack graph. A new path towards Node 1 (panic and violence on mass buses), is discovered using the ontology. The adversary can now take a much shorter path to reach the final goal.

Figure 3: Sample results. (a) Attack graph generated for the scenario of mass on buses. (b) The same attack graph, once enriched with data from the ontology. In both graphs, a red node represents the existence of a vulnerability on a given resource. An orange node represents network configuration (e.g., characteristic of a machine or connection between two machines in the network). When pre-conditions are respected, a yellow node represents an inference rule that leads to a fact (represented by a green node).

5. Related Work

5.1 Attack Graph Generation Approaches

Work by Ghosh and Ghosh [7] propose a planning-based approach for attack graph generation and analysis. In this approach, initial network configuration and description of exploits serve as input for the minimal attack graph generation. Shirazi et al. [18] present an approach for modeling attack-graph generation and analysis problems as a planning problem. They present a tool called AGBuilder that generates attack graphs using the Planning Domain Definition Language (PDDL) from extracted vulnerability information.

Roschke et al. [16] present an approach of vulnerability information extraction for attack graph generation using MulVAL. The proposal integrated attack graph workflows with SIEM alerts in terms of data fusion and correlation. Alerts are filtered based on vulnerability and system information of the attack graph. The correlation process can reveal a new way the network can be attacked. In such a case, the attack graph is updated.

Compared to those aforementioned approaches, 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 instead of a planning-based attack graph generation one. 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

Falodiya et al. [6] propose an ontology-based approach for attack graphs. The idea is to use an exploit dependency attack graph, an equivalence of logical attack graphs. The work presents an algorithmic solution to traverse the attack graph and add the extracted data into the ontology.

Lee et al. [12] also propose an approach for converting an attack graph into an ontology. Their formalism is based on an attack-graph approach by Ingols et al. [10]. The extraction of semantics from the graph is then labeled to build an RDF (Resource Description Framework) graph. Using RDF schema is beneficial for inferences from data and enhance searching. Wu et al. [20] propose as well an attack graph generation approach based on the inference ability of cybersecurity ontologies.

In our approach, we use these abilities of semantic languages to enrich logical attack graphs easily. We use a NIST standardized ontology (VDO, for Vulnerability Description Ontology) as the primary source of such vulnerability semantics. VDO corresponds well with the logical attack graph approach. It 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 when looking for other logical impacts of a given CVE in VDO. With this approach, we avoid recomputing the attack graph from scratch in the reasoning engine each time the system state change. The semantic abilities of logical attack graphs and ontologies also allow us to take an incremental approach to update the graphs. This improves the automation of the enrichment process (i.e., cybersecurity operators do not have to manually modify inputs to update the attack graphs).

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 represents successful attack perpetration. Compared to other similar approaches, such as topological and probabilistic attack graphs, our approach simplifies the inference process, since graphs' edges specify now 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 the generation of an initial attack graph (using network vulnerability scans), to the enrichment of the graph (mapping monitoring alerts and ontology semantics in real-time). The predictions of the initial graph get successfully updated into the enriched graph, based on attack evidences and semantic augmentation.text.

Acknowledgments - We acknowledge financial support from the European Commission (H2020 IMPETUS project, under grant agreement 883286).

References

[1] Encyclopaedia of Mathematics, Supplement Volume 1. 1997.

[2] François-Xavier Aguessy. Dynamic Risk Assessment and Response Computation using Bayesian Attack Models. PhD thesis, Telecom SudParis, France, 2016.

[3] François-Xavier Aguessy, Olivier Bettan, Gregory Blanc, Vania Conan, and Hervé Debar. Hybrid risk assessment model based on bayesian networks. In International Workshop on Security, pages 21–40. Springer, 2016.

[4] François-Xavier Aguessy, Lucie Gaspard, Olivier Bettan, and Vania Conan. Remediating Logical Attack Paths Using Information System Simulated Topologies. In C&ESAR 2014, C&ESAR 2014, page 187, Rennes, France, November 2014.

[5] Harold Booth and Christopher Turner. Vulnerability description ontology (VDO): a framework for characterizing vulnerabilities. Technical report, National Institute of Standards and Technology, 2016.

[6] Komal Falodiya and Manik Lal Das. Security Vulnerability Analysis using Ontologybased Attack Graphs. 2017 14th IEEE India Council International Conference, INDICON 2017, pages 1–5, 2018.

[7] Nirnay Ghosh and Soumya Ghosh. A planner-based approach to generate and analyze minimal attack graph. Applied Intelligence, 36(2):369–390, 2012.

[8] Danielle Gonzalez, Holly Hastings, and Mehdi Mirakhorli. Automated Characterization of Software Vulnerabilities.

[9] Qinwen Hu, Muhammad Rizwan Asghar, and Nevil 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), pages 350–361. IEEE, 2018.

[10] Kyle Ingols, Richard Lippmann, Keith Piwowarski, and Wood Street. Practical Attack Graph Generation for Network Defense.

[11] Elmar Kiesling, Andreas Ekelhart, Kabul Kurniawan, and Fajar Ekaputra. The sepses knowledge graph: an integrated resource for cybersecurity. In International Semantic Web Conference, pages 198–214. Springer, 2019.

[12] Jooyoung Lee, Daesung Moon, Ikkyun Kim, and Youngseok Lee. A semantic approach to improving machine readability of a large-scale attack graph. Journal of Supercomputing, 75(6):3028–3045, 2019.

[13] R. P. Lippmann and Kyle Ingols. An annotated review of past papers on attack graphs. Technical report, Massachusetts Institute of Technology, Lincoln Laboratory, 2005.

[14] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. Proceedings of the ACM Conference on Computer and Communications Security, (January 2006):336–345, 2006.

[15] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. Mulval: A logic-based network security analyzer. In USENIX Security Symposium, 2005.

[16] Sebastian Roschke, Feng Cheng, Robert Schuppenies, and Christoph Meinel. Towards unifying vulnerability information for attack graph construction. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5735 LNCS:218–233, 2009.

[17] Bruce Schneier. Modelling security threats. Dr. Dobb's Journal, 1999.

[18] Hossein Shirazi, Bruhadeshwar Bezawada, Indrakshi Ray, and Charles Anderson. Adversarial sampling attacks against phishing detection. In IFIP Annual Conference on Data and Applications Security and Privacy, pages 83–101. Springer, 2019.

[19] Mike Uschold and Michael Gruninger. Ontologies: Principles, methods and applications. The knowledge engineering review, 11(2):93–136, 1996.

[20] Songyang Wu, Yong Zhang, and Xiao 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), pages 1166–1174, 2018.