Building a Cybersecurity Knowledge Graph with CyberGraph

Paolo Falcarin
Ca' Foscari University of Venice
Dipartimento di Scienze Ambientali, Informatica e Statistica (DAIS)
Venice, Italy
paolo.falcarin@unive.it

Fabio Dainese
Ca' Foscari University of Venice
Dipartimento di Scienze Ambientali, Informatica e Statistica (DAIS)
Venice, Italy
857661@stud.unive.it

Abstract

Software engineers and security professionals rely on a variety of sources of information, including known vulnerabilities, newly identified weaknesses, and threats, as well as attack patterns and current mitigations. Such information, spread across different places, results in an increased effort for developers in following all the cross-referenced data and finding appropriate solutions to their security issues in a timely manner. Software developers cannot have a good knowledge of the breadth of the different issues and vulnerabilities that are constantly increasing in time; the raising number of security issues to tackle cannot be matched by software developers which need more help from intelligent tools. Therefore, in this work, we present CyberGraph, a tool to automatically build and update a single, easily queryable cybersecurity knowledge graph by automatically linking heterogeneous data from different public repositories. The resulting unique integrated dataset, thanks to its magnitude, allows the execution of sophisticated queries that can quickly provide new insights and valuable perspectives.

Keywords

Cybersecurity, Knowledge Graph, Software Vulnerabilities, Visualization, Neo4j, MITRE

ACM Reference Format:
Paolo Falcarin and Fabio Dainese. 2024. Building a Cybersecurity Knowledge Graph with CyberGraph. In *2024 ACM/IEEE 4th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and 2024 IEEE/ACM Second International Workshop on Software Vulnerability (EnCyCriS/SVM '24), April 15, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3643662.3643962

1 Introduction

The last years have seen an increase in attacks against software applications, both in number and severity [11–13]. The underlying problems are existing flaws and vulnerabilities hidden in production code or legacy systems that might be eventually exploited when the software is deployed. In order to contrast this widespread problem, security professionals relies on a variety of sources of information, including known vulnerabilities, newly identified weaknesses and threats, as well as attack patterns, and current mitigations or fixes.

Over the years some organizations and national agencies spearheaded various initiatives focused on making this type of information accessible in well-defined structured formats: the National Institute of Standards and Technology (NIST)[1], the China Information Technology Evaluation Center (CNITSEC)[2] and the MITRE Corporation[3] are the more notable.

To that end, a number of standards have been established which detail high-level schemas for cybersecurity information. Such information are collected, published, maintained and reviewed by many different parties including manufacturers, government agencies, research institutions and industry experts.
The value of standardized formats for sharing these type of knowledge has long been recognized. Not only do they enable easier, coherent and homogeneous knowledge sharing, but they also spurred the creation of additional helping tools [20] and services [24] that otherwise wouldn't be possible. Furthermore, making these data publicly available allows for greater transparency and accountability around cyber security issues.
Despite all of that, the major drawback related to these well-curated and valuable datasets is the fact that even up to these days they still remain quite isolated and not easily searchable in one single place, forcing the users to navigate to separate websites in order to get the full picture of all the cross-referenced data related to a single instance. As a result, this approach significantly increases both the difficulty as well as the time needed to obtain an overview of the data pertinent to the intended queries that the end-users want to perform against these information.
The increasing number of cyber-incidents and overwhelming cybersecurity skills shortage highlights the existing gaps between industry needs and academic skills [5] but they also underline the need of support with intelligent tools in tackling security incidents and fixing security vulnerabilities . In this paper, we present CyberGraph, an open-source software tool for building and updating a cybersecurity knowledge graph by leveraging on all the standardized data available from public repositories and integrate them into a single easily queryable public endpoint. This will allow users to perform meaningful and sophisticated queries across all of the data collected in the system, providing potentially new insights and valuable perspectives that were not possible before and simultaneously raising awareness of potential threats.

In order to construct the cybersecurity knowledge graph, we based our work on MITRE[4] and NIST[5], and their freely available datasets and metrics, namely: **CVE**[6] (*Common Vulnerabilities and Exposures*), **CNA**[7] (*CVE Numbering Authorities*), **CWE**[8] (*Common Weakness Enumeration*), **CAPEC**[9] (*Common Attack Pattern Enumeration and Classification*), **CPE**[10] (*Common Platform Enumeration*) and **CVSS**[11] (*Common Vulnerability Scoring System*). The CNA

[1]https://www.nist.gov/
[2]https://www.cnnvd.org.cn/
[3]https://www.mitre.org/
[4]https://www.mitre.org/
[5]https://www.nist.gov/
[6]https://cve.mitre.org/ or equivalently https://nvd.nist.gov/vuln
[7]https://www.cve.org/PartnerInformation/ListofPartners
[8]https://cwe.mitre.org/
[9]https://capec.mitre.org/
[10]https://cpe.mitre.org/ or equivalently https://nvd.nist.gov/products/cpe
[11]https://www.first.org/cvss/

EnCyCriS/SVM '24, April 15, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0565-6/24/04.
https://doi.org/10.1145/3643662.3643962

dataset is not available for download, but CyberGraph employs web scraping techniques to reconstruct it.

CyberGraph can build a knowledge graph from public repositories that could be used to:

- Identify and analyze relationships between vulnerabilities, software products, and security controls.
- Predict the impact of vulnerabilities on specific systems or organizations.
- Develop automated tools for vulnerability assessment and remediation.
- Generate insights into emerging trends in vulnerability exploitation.

The knowledge graph could also be used to predict the impact of vulnerabilities on specific systems or organizations by analyzing the historical data to identify patterns in the exploitation of vulnerabilities: using machine learning techniques to predict the likelihood of a particular vulnerability being exploited in a specific context, and developing targeted mitigation strategies.

2 Related Works

A better use of existing cybersecurity knowledge can provide strong support for cybersecurity operations, and the continuous increase of cybersecurity knowledge have fostered different studies on how to manage proficiently such huge amount of information. Different cybersecurity data standards, terminologies, taxonomies, and ontologies have been developed by various industrial, academic and government bodies [3, 9, 21, 22, 29] . Most of the related works appears to be restricted to a particular scope and/or mainly dedicated to presenting modern methods in order to extrapolate additional relationships between data entries. Xiang Li et al. [18] designed an effective data mining algorithm to obtain the basic characteristics of vulnerabilities based on the CVE and CWE databases. Zhuobing Han et al. [10] designed a knowledge graph embedding method that combined descriptions and structural knowledge. Hongbo Xiao et al. [30] embedded the relationship information and descriptive information of software security entities, using a CNN encoder to predict the relationship of software security entities based on the CVE, CWE, and CAPEC databases. Liu Yuan et al. [31] built a cybersecurity knowledge graph based on the CVE, CWE, and CAPEC databases and predicted the relationship between entities.

Our approach, on the other hand, is focused on constructing a comprehensive solution *cybersecurity knowledge graph*, hence intrinsically aiming at a broader scope, in order to cover all the possible end-users needs.

Xiao et al. [30] have built a knowledge graph having *CWEs* and *CAPECs* as core concept knowledge and *CVEs* as peripheral instance knowledge. On top of that a new embedding approach was developed to generate predictive embeddings of entities based on existing descriptions and relationships: this work produced a knowledge graph with a significantly limited number of entries. Other studies (e.g. [15, 16, 23]) aimed at extrapolating additional relationships upon given cybersecurity related descriptions, whether starting from a structured dataset or not, and incorporating them into a knowledge graph.

A different scope and approach has been proposed in [17], where the focus was on building a *Neo4j* cybersecurity knowledge graph specifically design to better reflect modern complex *network* attacks, such as *distributed denial-of-service* (*DDoS*): in this case no publicly available tool has been released.

A significant work was accomplished by Shen et al. [27] who developed a cybersecurity knowledge graph specifically targeting *Industrial Control Systems (ICS)* vulnerabilities starting from part of the same dataset considered in our study. In addition, they also employed advanced extrapolation description-features methods in order to obtain more results.

Finally, the *OWASP Dependency-Check* [20] is a *Software Composition Analysis* tool specifically designed to scan and analyze software components in order to identify potential security risks due to the dependency on any known vulnerable libraries. In output it will provide a minimal report containing any discovered vulnerable dependencies along with the appropriate *CVE* descriptions and general severity scores. Despite the similarities, the general usage is quite different from our proposed approach, since one can use *OWASP Dependency-Check* solely to list the current vulnerabilities of software libraries included in a project, and the generated reports are quite data-limited in comparison to what one can extract from our knowledge graph.

In addition to using existing structured cybersecurity data, researchers have extracted information from unstructured cyber-threat intelligence documents.

T. Satyapanich et al. [26] proposed the CASIE model, which can extract events from cybersecurity texts. Peipei Liu et al. [19] designed semantic augmentation networks able to extract cybersecurity concepts from unstructured texts. All of the above studies extract existing security knowledge from different perspectives, and can provide important guidance for future network protection measures: all the previous approaches, due to the limitations of knowledge extraction, are usually incomplete.

By mining the existing information in knowledge graph, it is usually possible to find unknown facts. This process can predict whether there are missing relationship edges between entities in the knowledge graph in order to expand the knowledge graph and correct errors [25].

In the domain of software protection, Basile et al. [4] defined a meta-model to construct a formal knowledge base of reverse engineering attacks, while Ceccato et al. created a taxonomy of attacks and protections with qualitative analysis reports of penetration testers [6] and data from a public challenge [7]: these approaches are time-consuming and not fully automated, and limited to a specific subset of attacks and protections.

3 Cybersecurity Knowledge Graph

The technologies used during the initial development phase had to be carefully selected so that they could provide the necessary features and functionality while being reliable and robust enough to handle the expected complex data and workloads.
To that extent we decided to use a node-base database as *Neo4j*[12]. Neo4j is one of the world's leading graph database, offering an efficient, highly scalable, reliable and secure way to store, manage and query large amounts of interconnected data. The powerful native graph storage and processing engine, combined with the *Cypher*[13] query language, makes it easy to quickly build, deploy and maintain a graph database, no matter the size of the project.

The resulting cybersecurity knowledge graph vocabularies and schema are based on pre-existing schemas and relative vocabularies currently used to publish instance data. This design choice was made with the primary goal of including all information from the

[12]https://neo4j.com/
[13]https://neo4j.com/developer/cypher/

original data sources and making the knowledge graph as detailed as possible to meet a wide range of user needs. By utilizing this approach, our semantic resource is made more user-friendly for those familiar with the original data resources. Additionally, our vocabularies are able to refer back to the original documentation and examples of the individual initial datasets, making it even easier for users to understand and access the data. Altogether, this methodology enabled us to create a knowledge graph that is both granular and accessible, providing users with a high degree of accuracy and usability.

We are going to only break down the developed *CVE subgraph schema*[14] due to space limitations. However, you can expect an equivalent level of details and data for the other subgraphs as well.

Just for reference, in *Figure 1* you can appreciate a simplified version - i.e. without considering certain node labels, relationships and any hierarchical *CNA, CWE, CAPEC* internal structure - of the overall knowledge graph schema.

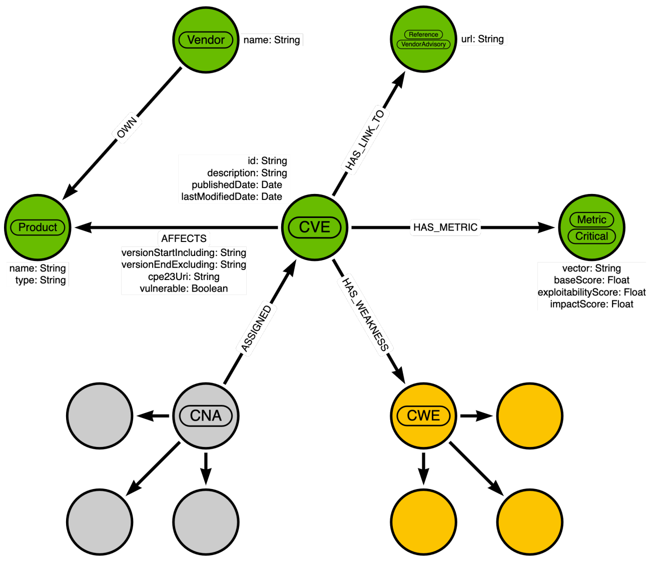


Figure 2: CVE subgraph data model.

In *Figure 2* you can appreciate the overall CVE subschema. It is composed by the following admissible *labels*:

- **CVE**: label used to identify a node as a *CVE* (*Common Vulnerabilities and Exposures*) entry. It admits four property keys:
  - *id*: a string reporting the relative *CVE ID*;
  - *description*: a string describing the overall vulnerability pertinent issues;
  - *publishedDate*: a *date* format value (*YYYY-MM-DD[timezone hour]*) identifying the publishing time of the relative CVE entry;
  - *lastModifiedDate*: a *date* format value (*YYYY-MM-DD[timezone hour]*) identifying the last modification time made to the relative CVE entry information;
- **Metric**: label used to identify the possible related CVE severity metrics. It is based out of the *CVSS v3.1*[15] scoring system. It admits four property keys:
  - *baseScore*: a numerical value, ranging from 0 up to 10, describing the severity of the related CVE (with higher score indicating higher severity). It is computed based on the values of the *scoring vector*;
  - *exploitabilityScore*: a numerical value, ranging from 0.1 up to 3.9, describing how likely the related vulnerability will be exploited. It is computed based on certain metrics of the *scoring vector* and constitutes a sub-part of the overall *base score* value;
  - *impactScore*: a numerical value, ranging from 1.4 up to 6, describing the impact on the vulnerable component subsequently a successful exploitation. It is computed based on certain metrics of the *scoring vector* and constitutes a sub-part of the overall *base score* value;
  - *vector*: a string reporting the relative *CVSS v3.1* scoring *base metrics*. Here's an example of a scoring vector:

    AV:L/AC:H/PR:L/UI:R/S:U/C:N/I:H/A:N

    Its structure is composed by the following fields:
    * *Attack Vector* (AV): metric that reflects the context of exploitation. It takes into account the number of potential attackers by measuring the remoteness of an attack, with more remote attacks resulting ultimately in a larger *base score*. This is due to the assumption that remote attacks are more likely to occur than those that require physical access to a device. This field can assume the following admissible values:
      · *Network* (N): the vulnerable component is connected to the network infrastructure, making it exposed to threats from the entire Internet (broadest option);
      · *Adjacent* (A): the vulnerable component is connected to the network, however the attack is limited to a nearby topology. This means that it must be launched from the same shared physical or logical network (e.g. Bluetooth, Wi-Fi, administrative VPN, etc.);
      · *Local* (L): the vulnerable component is not network stack bounded and is subject to an attack that involves read/write/execute capabilities (e.g. accessing locally or remotely the system through keyboard, console, SSH, etc.);
      · *Physical* (P): the vulnerable component is subject to an attack that involves physical contact or manipulation, which may be brief or persistent;
    * *Attack Complexity* (AC): metric that assesses the conditions beyond the attacker's control which are necessary in order to exploit the vulnerability (this exclude any required user interaction, which are instead captured under the *User Interaction* metric). Furthermore, assume that the vulnerable component is in the required configuration (if any). The easier the attack, the higher the *base score*. This field can assume the following admissible values:
      · *Low* (L): no particular conditions are required, thus an attacker can expect to succeed each time they attempt an attack;
      · *High* (H): a successful attack is highly dependent on factors outside of the attacker's control, thus requiring them to invest effort into the preparation and execution of the attack in order to obtain a successful outcome (e.g. repeated exploitation to win a race condition, environmental knowledge gathering such as shared secrets or sequence numbers, etc.);
    * *Privileges Required* (PR): metric that indicates the level of privileges required prior to exploit a vulnerability, with the highest *base score* given when no privileges are required. This field can assume the following admissible values:

[14]Graph schema developed based on the *Labeled Property Graphs* (LPGs) model. For more information, please refer to the *Neo4j graph concepts introductory section*.
[15]https://www.first.org/cvss/v3.1/specification-document

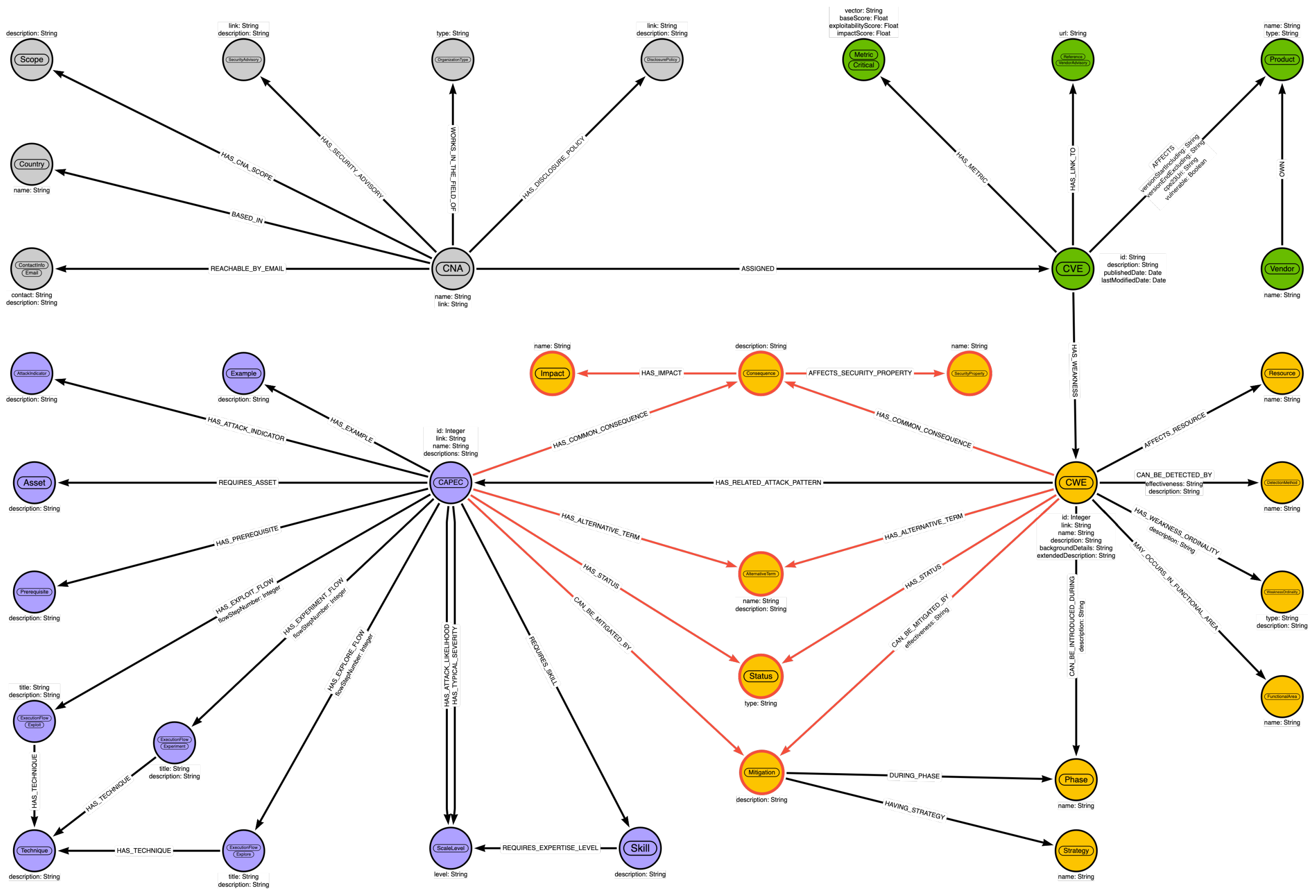


Figure 1: Simplified overall data model. To be noted that the nodes are color-coded according to their cluster category: gray for CNA nodes, green for CVE nodes, purple for CAPEC nodes, and yellow for CWE nodes. Moreover, the red outlined nodes indicate that these specific entities are usable across the designated diverse cluster categories.

· None (N): no particular privileges required in order to launch an attack;
· Low (L): only basic user capabilities needed that typically allow for the manipulation of settings and files owned by a user (or equivalently be able to access non-sensitive resources);
· High (H): significant privileges required (e.g. administrative) in order to launch an attack;

∗ User Interaction (UI): metric that measures the need for a user, other than the attacker, to be involved in order for a vulnerable component to be successfully exploited. The highest *base score* is given when no user interaction is required. This field can assume the following admissible values:
· None (N): no external user interaction needed;
· Required (R): the exploitation requires an external user to perform certain actions (e.g. only viable when a system administrator is installing an application);

∗ Scope (S): metric that determines whether a vulnerability's impact breaches the security boundary of one security scope and affects components that are outside of its jurisdiction. It is formally defined in terms of *security authorities*, which are mechanisms (e.g. OS, sandboxes, etc.) that regulate and manage the access policies to restricted resources (e.g. CPU, memory, etc.). All the entities regulated by a single security authority are seen to be under one security scope. A *scope change* occurs whenever a vulnerability impacts a component outside of its scope, thus leading to higher *base score*. This field can assume the following admissible values:
· Unchanged (U): the vulnerable components and the affected ones are regulated by the same *security authority* (no scope change occurred);
· Changed (C): the vulnerable components and the affected ones are regulated by separate *security authorities* (scope change occurred);

∗ Confidentiality (C): metric that assesses the impact to the information confidentiality subsequently a successful exploit. Higher loss of confidentiality will result in a higher *base score*. This field can assume the following admissible values:
· None (N): no loss of confidentiality;
· Low (L): confidentiality breached, but the attacker have gain access to a limited amount of information and/or it doesn't have control over what information have been retrieved;
· High (H): total loss of confidentiality (by amount or due to the secret retrieved, e.g. administrator password);

∗ Integrity (I): metric that assesses the impact to the information integrity subsequently a successful exploit. Higher integrity impact will result in a higher *base score*. This field can assume the following admissible values:

Rating	Base Score
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Table 1: Mapping of the qualitative CVE severity ratings to the corresponding base score value ranges.

  · *None* (N): no loss of integrity;
  · *Low* (L): integrity breached, but the attacker have limited range of modification and/or it doesn't have control over the result of a modification;
  · *High* (H): total loss of integrity;
 ∗ *Availability* (A): metric that assesses the impact to the availability subsequently a successful exploit. Higher availability impact will result in a higher *base score*. This field can assume the following admissible values:
  · *None* (N): no impact to availability;
  · *Low* (L): availability affected through reduced performance, but the attacker is not able to fully deny the access to the service;
  · *High* (H): total loss of availability;

- **None/Low/Medium/High/Critical**: additional labels that they must be used combined with the *Metric* one representing *qualitatively* the severity expressed by the *base score* value property key. All the available ratings are mapped to a particular base score range, as reported in *Table 1*. These specializations are mutually exclusive with each other and they have been developed to speed up some possible future queries.
- **Reference**: label used to identify any possible useful external reference linked to the relative CVE entry. It admits a single property key:
  – *url*: an external hyperlink pointing to the related resource webpage;
- **Product**: label used to identify the product related to the CVE entry. It admits two property keys:
  – *name*: a string reporting the name of product;
  – *type*: a string reporting the type of the related product. It can either be "*Hardware*", "*Application*" (i.e. software) or "*Operating System*";
- **Vendor**: label used to identify the vendor/organization that owns the *Product* related to the CVE entry. It admits a single property key:
  – *name*: a string reporting the name of the vendor/organization;

Additionally, the following labels must be used combined with the *Reference* one to further discern the type of external resource linked to the related CVE entry. These specializations are *not* mutually exclusive with each other and they have been developed to speed up some possible future queries.

- **Patch**: the reference contains an update to the product that fixes the related vulnerability;
- **VendorAdvisory**: the reference contains the related advisory from the vendor/publisher of the product or the parent company that owns the vendor;
- **ThirdPartyAdvisory**: the reference contains the related advisory from an organization that is *not* the vulnerable product's vendor or publisher;
- **Exploit**: the reference contains an in-depth/detailed description of steps on how to exploit the related vulnerability or any legitimate PoC[16] code or exploit kit;
- **PermissionsRequired**: the reference hyperlink provided is blocked by a login page. Only users with valid credentials can consume the information;
- **ReleaseNotes**: the reference is in the format of a vendor or open source project's release notes or change log;
- **MailingList**: the reference is from a mailing list;
- **IssueTracking**: the reference is a post from a bug tracking tool such as *MantisBT*, *Bugzilla* or *JIRA*;
- **BrokenLink**: the reference hyperlink is returning 404 errors or the site is no longer online;
- **USGovernmentResource**: the reference is from a U.S. Government agency or organization (typically from a *.gov* or *.mil* top-level domain);
- **NotApplicable**: the reference hyperlink is *not* applicable to the related vulnerability and was likely associated by the CVE Program accidentally;
- **Mitigation**: the reference contains information on possible steps to implement in order to mitigate the related vulnerability in the event that a patch can't be applied or is not yet available;
- **TechnicalDescription**: the reference contains in-depth technical information about the related vulnerability and its exploitation process. It can be in the form of a presentation or whitepaper;
- **PressMediaCoverage**: the reference is from a public media outlet/entity such as a newspaper, magazine, web log or social media (*not* from individuals personal social media account though);
- **ToolSignature**: the reference contains data that can be used directly by a scanning tool;
- **Product**: the reference contains information describing the related product for the purpose of *CPE* or *SWID*[17];
- **VDBEntry**: the reference leads to a *VDB entry*[18].

Moreover, the developed CVE subschema allows for the following *relationships*:

- **HAS_METRIC**: relationship used uniquely to connect the *CVE* and *Metric* nodes. It doesn't allows for any additional property key.
- **HAS_LINK_TO**: relationship used uniquely to connect the *CVE* and *Reference* nodes. It doesn't allows for any additional property key.
- **OWN**: relationship used uniquely to connect the *Vendor* and *Product* nodes. It doesn't allows for any additional property key.
- **AFFECTS**: relationship used uniquely to connect the *CVE* and *Product* nodes. It admits four property keys:
  – *versionStartIncluding*: a string reporting from which version number onwards the product is affected by the related CVE entry vulnerability;
  – *versionEndExcluding*: a string reporting up to which excluding version number the product is affected by the related CVE entry vulnerability. If it is *not* present, it means that even the most currently released version is affected by the specified vulnerability;

[16] *PoC* (or *Proof of Concept*) code, in the cyber security field, refers to code developed to test security vulnerabilities and demonstrate the exploitability of a system.
[17] https://csrc.nist.gov/projects/Software-Identification-SWID
[18] *VDBs* (or *Vulnerability Databases*) are freely available public datasets that provide broad cybersecurity vulnerability coverage (i.e. not limited to a single organization). For more information visit https://www.first.org/global/sigs/vrdx/vdb-catalog.

- *vulnerable*: a *boolean* value indicating if the related product is vulnerable by the specified CVE or not (used to discern if the product is actually vulnerable or it is just required in order to leverage the CVE vulnerability);
- *cpe23Uri*: a string reporting the relative *CPE v2.3*[19] vector used to identify unambiguously and in a standardize way the related IT product classes. Here's an example of a CPE URI string:

  cpe:2.3:a:apple:icloud:6.1.1:beta1:*:*:*:macos:x86:*

  Its colon-separated naming specification adheres to the following ordered structure:
  - Standard initial *"cpe"* string;
  - *CPE* version (e.g. 2.3);
  - ***Part***: product type (*a*: application, *o*: operating system, *h*: hardware);
  - ***Vendor***: vendor name;
  - ***Product***: product name;
  - ***Version***: release version of the product;
  - ***Update***: update, service pack or point release of the product;
  - ***Edition***: edition-related terms applied by the vendor to the product (this attribute is considered *deprecated* in the current v2.3 version, but kept for backward compatibility);
  - ***Language***: language of the product (in *RFC 5646* format, e.g. en-us, it-it, etc.);
  - ***SW_Edition***: terms to distinguish how the product is tailored to a particular class of end users or market (e.g. online, enterprise, etc.);
  - ***Target_SW***: target software of the product (e.g. macOS, Windows, etc.);
  - ***Target_HW***: target hardware of the product (e.g. x86, arm, etc.);
  - ***Other***: any other information not captured by the previous attributes;

  Be also aware that the symbol * is used as a wildcard in order to not specify a particular attribute (either because it cannot be applied to the related product or with the purpose of indicating all the underlying admissible values, e.g. in case of *Target_HW*, it means that all the instruction set architectures are involved).

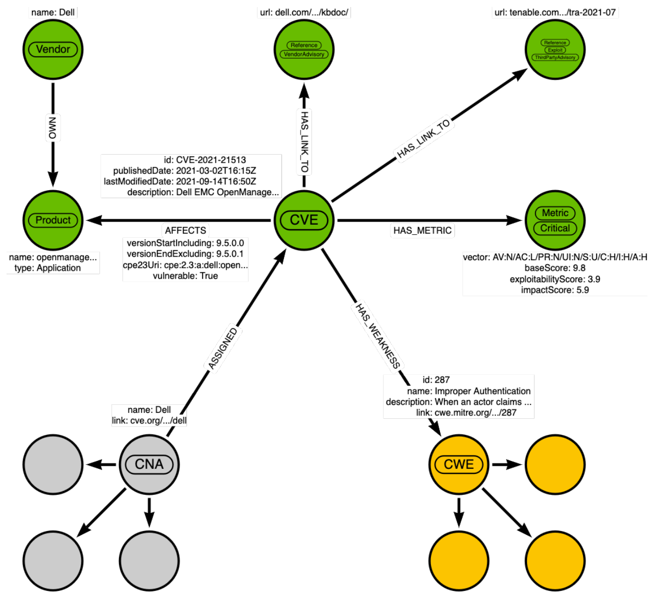

name: Dell
Vendor
OWN
url: dell.com/.../kbdoc/
Reference
VendorAdvisory
url: tenable.com.../tra-2021-07
Reference
Exploit
ThirdPartyAdvisory
HAS_LINK_TO
HAS_LINK_TO
id: CVE-2021-21513
publishedDate: 2021-03-02T16:15Z
lastModifiedDate: 2021-09-14T16:50Z
description: Dell EMC OpenManage...
Product
CVE
AFFECTS
versionStartIncluding: 9.5.0.0
versionEndExcluding: 9.5.0.1
cpe23Uri: cpe:2.3:a:dell:open...
vulnerable: True
name: openmanage...
type: Application
HAS_METRIC
Metric
Critical
vector: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
baseScore: 9.8
exploitabilityScore: 3.9
impactScore: 5.9
ASSIGNED
HAS_WEAKNESS
name: Dell
link: cve.org/.../dell
id: 287
name: Improper Authentication
description: When an actor claims ...
link: cwe.mitre.org/.../287
CNA
CWE

Figure 3: Example of a *CVE* node instance from the final cybersecurity knowledge graph.

- **HAS_WEAKNESS**: relationship used uniquely to connect *CVE* and *CWE* nodes. It fundamentally shows which standardized security flaws are related to the CVE entry. It doesn't allows for any additional property key.

Ultimately, to consolidate the presented knowledge, in *Figure 3* you can appreciate an example of a *CVE* node instance from the actual final cybersecurity knowledge graph.

4 Knowledge Graph Search Use Case

In order to provide a more exhaustive overview of the benefits that the developed cybersecurity knowledge graph can offer, in this section, we will present a typical search scenario in which the knowledge graph database can be utilized.
Once the data are gathered, CyberGraph can recognize the named entities and identify relationships between them.

A typical search scenario is the one in which a user seeks to retrieve all the relative cybersecurity information - e.g. *CNA, CVE, CWE, CAPEC* - starting from a specific vulnerability. Firstly, there will be presented all the steps for obtaining such data without the proposed knowledge graph, which actually closely represent what a user must go through nowadays, and later it will be compared to the process of obtaining the same information through the utilization of the developed graph data source.

Let's take for example the following vulnerability: `CVE-2021-44201`. If a user wants to know everything related to that specific *CVE* entry has to:

- Visit the *NIST National Vulnerability Database* website and search for the corresponding *CVE* entry (e.g. https://nvd.nist.gov/vuln/detail/CVE-2021-44201);
- From that, the user has to analyze the presented content and extrapolate:
  - Which are the related *CWE* weaknesses. Thankfully, for each *CWE* entry the platform makes available an hyperlink to the corresponding *MITRE* description web page;
  - Who was the *CVE ID* assigner. In this case, there will be available only the *CNA* name without any hyperlink pointing to the relative description web page;
- Then, the user has to visit a different website and manually search for the related *CNA* authority in order to gain more information about it (e.g. https://www.cve.org/PartnerInformation/ListofPartners/partner/Acronis);
- Next, for each *CWE* weaknesses found, the user has to follow the corresponding external hyperlink and identify the related *CAPEC* attack pattern entries (e.g. https://cwe.mitre.org/data/definitions/79.html);
- Finally, for each *CAPEC* entry found, the user has to follow the corresponding external hyperlink;

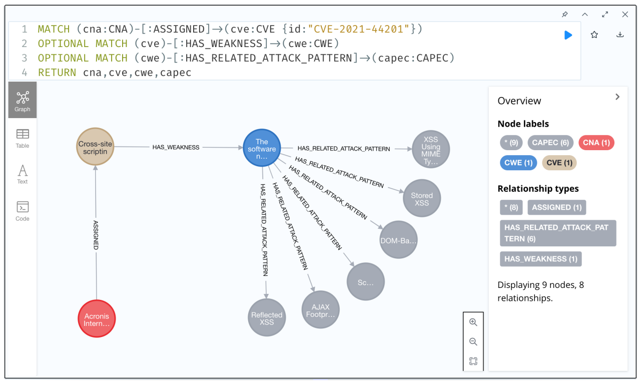
As you can observe, the required steps for a user to go though in order to get the full picture of all the cross-referenced data related to a single vulnerability are quite time consuming and obligates to visit multiple distinct websites.
Now, here's instead the *Neo4j Cypher* query to retrieve the same data using the proposed cybersecurity knowledge graph:

```
MATCH (cna:CNA)-[:ASSIGNED]->(cve:CVE {id:"CVE-2021-44201"})
OPTIONAL MATCH (cve)-[:HAS_WEAKNESS]->(cwe:CWE)
OPTIONAL MATCH (cwe)-[:HAS_RELATED_ATTACK_PATTERN]->(capec
 :CAPEC)
RETURN cna,cve,cwe,capec
```

As depicted in *Figure 4*, the *Neo4j Desktop* application provides a user-friendly visual - among other types - query output with only

[19] https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7697.pdf

displayed the desired information. Hovering or single-clicking on a node will show its relative property key values on the right-hand side detail box. Furthermore, double-clicking on a node will instead reveal additional related nodes, such as *Reference*, *Metric*, etc. ones.



```
MATCH (cna:CNA)-[:ASSIGNED]→(cve:CVE {id:"CVE-2021-44201"})
OPTIONAL MATCH (cve)-[:HAS_WEAKNESS]→(cwe:CWE)
OPTIONAL MATCH (cwe)-[:HAS_RELATED_ATTACK_PATTERN]→(capec:CAPEC)
RETURN cna,cve,cwe,capec
```

Overview

Node labels: * (9), CAPEC (6), CNA (1), CWE (1), CVE (1)

Relationship types: * (8), ASSIGNED (1), HAS_RELATED_ATTACK_PATTERN (6), HAS_WEAKNESS (1)

Displaying 9 nodes, 8 relationships.

Figure 4: Example of *Neo4j Desktop* visual query output.

As perceived, the method employing the cybersecurity knowledge graph yields a superior overall user-experience defined in terms of searchability, reduced execution time and data visualization compared to the current required actions that a user must perform in order to obtain the same information. Clearly, these presented benefits are further accentuated in scenarios in which a user desires to search for *multiple* distinct vulnerabilities/entities and even more so when the tasks would be too tedious and time consuming to manually execute them (e.g. '*how many CVEs currently have a published patch?*', '*Which are the top 10 CNA authorities that have assigned the most CVE entries?*', etc.).

Given a vulnerability in a particular software product, the knowledge graph could be used to identify other products that are also vulnerable to the same or similar attacks; the graph could also be used to identify security controls that could be implemented to mitigate the risk of exploitation: this information could be used to prioritize vulnerability remediation efforts and to develop more effective security policies.

5 Conclusions and Future Work

CyberGraph offers a web interface , allowing the user to simply query the cybersecurity knowledge graph for information without writing Cipher queries.

CyberGraph is able to build a cybersecurity knowledge graph that takes around 550MB and it is composed by:

- More than 735000 nodes;
- More than 1.3 million relationships;

These data and statistics on the graph can be obtained by running the *Cypher* code `:sysinfo`. Along with CyberGraph we developed a series of corollary utility functions - written in *Python* - for automatically managing all the aspects of the graph, such as handling the data retrieval phase, the initial ingestion inside the knowledge graph and a series of operations for incremental updating, all crucial features to maintain the accuracy of the *Neo4j* database information.

The *cybersecurity knowledge graph* built by CyberGraph unifies all the major freely available information on attack patterns, weaknesses, vulnerabilities, numbering authorities and exposed platforms from multiple and disparate sources into one single unified and coherent dataset.

We plan to further integrate information from the National Vulnerability Database (NVD), and vulnerability reports from security researchers and software vendors.

All the comprehensive code implementation for the showcased cybersecurity knowledge graph are openly accessible[20].
In addition, to further enhance the capabilities of the entire ecosystem, the future works should be focused on expanding the functionalities and features offered. For instance, we suggest the integration of the recently presented *Exploit Prediction Scoring System*[21] (*EPSS*) [14] into the graph schema for the purpose of enhancing the patching vulnerability prioritization, since this type of score estimates the probability of observing any exploitation attempts against a *CVE* entry in the following 30 days.

It is worth noticing that the *EPSS* and *CVSS* scores are complementary with each other - and ***not*** mutually exclusive - since the former estimates the probability that a vulnerability will be exploited in the near future - in the next 30-day period window - and it is fully based on on-field data-driven analysis, meanwhile the latter express the overall "*severity*" of a vulnerability and it is computed on the immutable values of the relative scoring vector. Hence, the introduction of the *EPSS* score within the graph schema, alongside with the already present *CVSS* metric, can serve as an effective means for providing to the end-users a straightforward decision support system for vulnerabilities patching prioritisation.

Moreover, an additional improvement could be focused on expanding the *CAPEC* available graph information by supplementing them with the ATT&CK[22] (or *Adversarial Tactics, Techniques and Common Knowledge* [28]) dataset, as already proven in the published work accomplished by Ampel et al. [1] who successfully interlinked these two knowledge bases. Similarly, it would be also interesting to incorporate custom attack models for specific domains [8, 32].

Finally, the application of *Natural Language Understanding* (*NLU*) techniques (a subset of the *Natural Language Processing*, *NLP*) - either using pre-established *SaaS* tools, such as *IBM Watson*[23], or through open-source libraries, e.g. *AMR*[24] [2] - can bring immense benefits to the developed graph by enabling the analysis of the contained unstructured raw data, such as those from the descriptions of CVEs, to extract additional knowledge. For instance, an application could be focused on semantic extraction in order to tag the available *CVE* entries by their business categories, for the purpose of creating sub-clusters of knowledge targeting specific industries. An additional work could be the extraction of contexts, hence relations, for the newly released CVEs. This could be useful due to the fact that once a new vulnerability is released might not contains all the necessary relationships with its corresponding *CWE* entries, thus inferring them could compensate that up until these information are added in the future. The knowledge graph could also be used to develop automated tools for vulnerability assessment and remediation. These tools could use the graph to: scan systems for vulnerabilities, identify patches or workarounds for known vulnerabilities, automate the deployment of patches or workarounds. This could help to reduce the time it takes to fix vulnerabilities and to improve the overall security of organizations. Finally, the knowledge graph could be used to generate insights

[20]https://github.com/UniVE-SSV/CyberGraph
[21]https://www.first.org/epss/model
[22]https://attack.mitre.org/
[23]https://www.ibm.com/watson
[24]https://amr.isi.edu/language.html

into emerging trends in vulnerability exploitation, by identifying patterns in the types of vulnerabilities that are being exploited, analysing the sources of vulnerabilities, tracking the evolution of exploit techniques. Overall, Cybergraph is a valuable tool helping to improve vulnerability management and reduce the risk of exploitation.

Acknowledgments

This work was supported by project SERICS (PE00000014) under the National Recovery and Resilience Plan of the Italian Ministry of University and Research (MUR), funded by the European Union - NextGenerationEU.

References

[1] Benjamin Ampel, Sagar Samtani, Steven Ullman, and Hsinchun Chen. 2021. Linking common vulnerabilities and exposures to the mitre att&ck framework: A self-distillation approach. *arXiv preprint arXiv:2108.01696* (2021).

[2] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*. 178–186.

[3] Sean Barnum. 2012. Standardizing cyber threat intelligence information with the structured threat information expression (stix). *Mitre Corporation* 11 (2012), 1–22.

[4] Cataldo Basile, Daniele Canavese, Leonardo Regano, Paolo Falcarin, and Bjorn De Sutter. 2019. A meta-model for software protections and reverse engineering attacks. *Journal of Systems and Software* 150 (2019), 3–21. https://doi.org/10.1016/j.jss.2018.12.025

[5] Cagatay Catal, Alper Ozcan, Emrah Donmez, and Ahmet Kasif. 2023. Analysis of cyber security knowledge gaps based on cyber security body of knowledge. *Education and Information Technologies* 28, 2 (2023), 1809–1831.

[6] M. Ceccato, P. Tonella, C. Basile, B. Coppens, B. De Sutter, P. Falcarin, and M. Torchiano. 2017. How Professional Hackers Understand Protected Code while Performing Attack Tasks. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 154–164. https://doi.org/10.1109/ICPC.2017.2

[7] Mariano Ceccato, Paolo Tonella, Cataldo Basile, Paolo Falcarin, Marco Torchiano, Bart Coppens, and Bjorn De Sutter. 2019. Understanding the behaviour of hackers while performing attack tasks in a professional setting and in a public challenge. *Empirical Software Engineering* 24 (2019), 240–286.

[8] Thomas M. Chen, Juan Carlos Sánchez-Aarnoutse, and John F. Buford. 2011. Petri Net Modeling of Cyber-Physical Attacks on Smart Grid. *IEEE Trans. Smart Grid* 2, 4 (2011), 741–749. https://doi.org/10.1109/TSG.2011.2160000

[9] Stefan Fenz and Andreas Ekelhart. 2009. Formalizing information security knowledge. In *Proceedings of the 4th international Symposium on information, Computer, and Communications Security*. 183–194.

[10] Zhuobing Han, Xiaohong Li, Hongtao Liu, Zhenchang Xing, and Zhiyong Feng. 2018. Deepweak: Reasoning common software weaknesses via knowledge graph embedding. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 456–466.

[11] International Business Machines Corporation (IBM). 2022. *Cost of a Data Breach - 2022 Report*. arXiv:https://www.ibm.com/downloads/cas/3R8N1DZJ https://www.ibm.com/reports/data-breach

[12] SonicWall Inc. 2022. *2022 Cyber Threat Report*. https://www.sonicwall.com/medialibrary/en/white-paper/2022-sonicwall-cyber-threat-report.pdf

[13] Verizon Communications Inc. 2022. *2022 Data Breach Investigations Report*. arXiv:https://www.verizon.com/business/resources/T6a/reports/dbir/2022-data-breach-investigations-report-dbir.pdf https://www.verizon.com/business/resources/reports/dbir/

[14] Jay Jacobs, Sasha Romanosky, Idris Adjerid, and Wade Baker. 2020. Improving vulnerability remediation through better exploit prediction. *Journal of Cybersecurity* 6, 1 (2020), tyaa015.

[15] Yan Jia, Yulu Qi, Huaijun Shang, Rong Jiang, and Aiping Li. 2018. A practical approach to constructing a knowledge graph for cybersecurity. *Engineering* 4, 1 (2018), 53–60.

[16] Corinne L Jones, Robert A Bridges, Kelly MT Huffer, and John R Goodall. 2015. Towards a relation extraction framework for cyber-security concepts. In *Proceedings of the 10th Annual Cyber and Information Security Research Conference*. 1–4.

[17] Kun Li, Huachun Zhou, Zhe Tu, and Bohao Feng. 2020. CSKB: a cyber security knowledge base based on knowledge graph. In *Security and Privacy in Digital Economy: First International Conference, SPDE 2020, Quzhou, China, October 30–November 1, 2020, Proceedings 1*. Springer, 100–113.

[18] Xiang Li, Jinfu Chen, Zhechao Lin, Lin Zhang, Zibin Wang, Minmin Zhou, and Wanggen Xie. 2017. A mining approach to obtain the software vulnerability characteristics. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*. IEEE, 296–301.

[19] Peipei Liu, Hong Li, Zuoguang Wang, Jie Liu, Yimo Ren, and Hongsong Zhu. 2022. Multi-features based Semantic Augmentation Networks for Named Entity Recognition in Threat Intelligence. In *2022 26th International Conference on Pattern Recognition (ICPR)*. IEEE, 1557–1563.

[20] SS Jeremy Long, S Springett, and W Stranathan. 2015. Owasp dependency check. https://owasp.org/www-project-dependency-check/

[21] Leo Obrst, Penny Chase, and Richard Markeloff. 2012. Developing an Ontology of the Cyber Security Domain.. In *STIDS*. 49–56.

[22] Alessandro Oltramari, Lorrie Faith Cranor, Robert J Walls, and Patrick D McDaniel. 2014. Building an Ontology of Cyber Security.. In *STIDS*. Citeseer, 54–61.

[23] Aditya Pingle, Aritran Piplai, Sudip Mittal, Anupam Joshi, James Holt, and Richard Zak. 2019. Relext: Relation extraction using deep learning approaches for cybersecurity knowledge graph improvement. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. 879–886.

[24] The Software Security Project. 2023. Zed Attack Proxy. https://www.zaproxy.org/

[25] Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. 2021. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 2 (2021), 1–49.

[26] Taneeya Satyapanich, Francis Ferraro, and Tim Finin. 2020. Casie: Extracting cybersecurity event information from text. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 8749–8757.

[27] Guowei Shen, Wanling Wang, Qilin Mu, Yanhong Pu, Ya Qin, and Miao Yu. 2020. Data-driven cybersecurity knowledge graph construction for industrial control system security. *Wireless Communications and Mobile Computing* 2020 (2020), 1–13.

[28] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. 2018. Mitre att&ck: Design and philosophy. In *Technical report*. The MITRE Corporation.

[29] Zareen Syed, Ankur Padia, Tim Finin, Lisa Mathews, and Anupam Joshi. 2016. UCO: A unified cybersecurity ontology. In *Workshops at the thirtieth AAAI conference on artificial intelligence*.

[30] Hongbo Xiao, Zhenchang Xing, Xiaohong Li, and Hao Guo. 2019. Embedding and predicting software security entity relationships: A knowledge graph based approach. In *Neural Information Processing: 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12–15, 2019, Proceedings, Part III 26*. Springer, 50–63.

[31] Liu Yuan, Yude Bai, Zhenchang Xing, Sen Chen, Xiaohong Li, and Zhidong Deng. 2021. Predicting entity relations across different security databases by using graph attention network. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 834–843.

[32] Gaofeng Zhang, Paolo Falcarin, Elena Gómez-Martínez, Christophe Tartary, Shareeful Islam, Bjorn De Sutter, and Jerome D'Annoville. 2016. Attack Simulation based Software Protection Assessment Method for Protection Optimisation. In *Proc. Int. Conf. Cyber Security and Protection of Digital Services (Cyber Security)*. 1–8. https://doi.org/10.1109/CyberSecPODS.2016.7502352