LPIOT2020-2021: M3 (Neo4J)

Adrian-Paul Carrières

Sujet

Le sujet qui a été choisi tombe dans le domaine de la cybersécurité. Pour être précis, il s'agissait d'utiliser Neo4J pour stocker des CVE, acronyme signifiant Common Vulnerabilities and Exposures. Cela désigne une liste publique de failles de sécurité informatique, que l'on identifie via son nom composé de la sorte suivante : CVE-année-identifiant. Ainsi, une des failles de Windows 10 en 2021 porte le nom de CVE-2021-26411.

Chaque entrée comporte donc des informations sur la sévérité de la faille (sous plusieurs versions), sur les produits affectés, quel est le vendeur qui a inventé ce produit, une description et ainsi de suite.

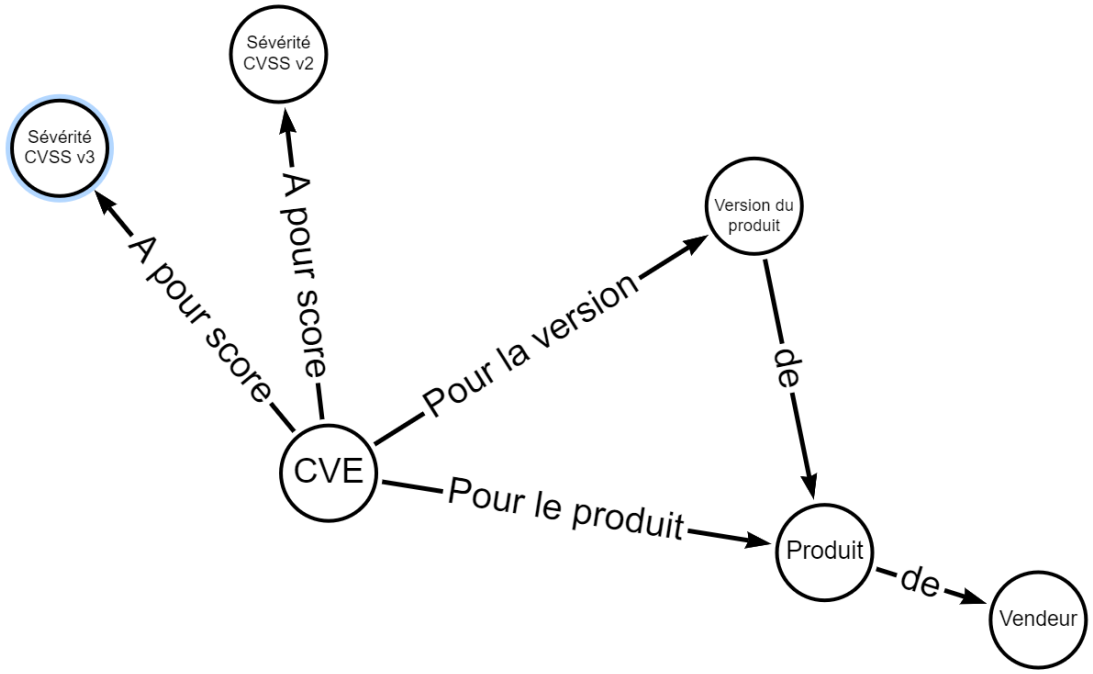
J'ai décidé d'utiliser la base de données de NIST (https://nvd.nist.gov/vuln/data-feeds) qui comporte plus de détails que les CVEs de base.

La finalité du projet était de proposer un simple outil permettant de retrouver une liste de faille pour un produit en particulier.

Un fichier de test ne comportant qu'un seul CVE ainsi que le fichier de l'année 2021 seront disponibles directement sous le format originel et celui nettoyé à la racine du projet, accompagnés du script Cypher permettant l'import des données.

Modélisation

La modélisation n'a rien de particulièrement compliquée :



C'est certes une modélisation quelque peu limitée, mais je l'ai jugée suffisante compte tenu de la finalité recherchée.

Le but étant donc depuis une application (ici Web) de récupérer la liste des CVE par version d'un produit spécifique. J'y ai cela dit ajouté la sévérité pour l'utilisation des requêtes, mais elle aurait pu tout aussi bien être affichée dans l'application Web à côté de chaque CVE.

Compétences

En compétences pures de Neo4J je n'ai pas particulièrement progressé par rapport aux différents TP. La différence majeure tenant de l'utilisant d'apoc pour pouvoir gérer des JSON ainsi que de quelques fonctions comme le UNWIND que nous n'avons de souvenir pas utilisé.

Là où j'ai plus appris rentre dans la recherche d'information et surtout le nettoyage des données.

Nettoyage des données

J'ai découvert avec un peu de surprise que les données des produits et du vendeur étaient bien plus difficiles d'accès qu'auparavant.

Ainsi, ce qui était un simple :

```
"affects" : {
        "vendor" : {
          "vendor_data" : [...]
        }
      }
```

est depuis devenu :

```
"configurations": {
                "CVE_data_version": "4.0",
                "nodes": [
                    {
                        "operator": "AND",
                        "children": [
```

```
                        {
                            "operator": "OR",
                            "children": [],
                            "cpe_match": [
                                {
                                    "vulnerable": true,
                                    "cpe23Uri": "...",
                                    "versionEndExcluding": "604",
                                    "cpe_name": []
                                }
                            ]
                        },
                        {
                            "operator": "OR",
                            "children": [],
                            "cpe_match": [
                                {
                                    "vulnerable": false,
                                    "cpe23Uri": "...",
                                    "cpe_name": []
                                }
                            ]
                        }
                    ],
                    "cpe_match": []
                }
            ]
        }
```

J'ai donc cherché à simplifier beaucoup plus mes fichiers pour me simplifier ensuite la vie dans mes requêtes Cypher. En effet, j'ai assez vite abandonné l'idée de traiter un niveau d'imbrication aussi variable avec de la récursion dans Cypher.

Mes fichiers finaux suivent ce template :

```
[
    {
        "description": "...",
        "id": "CVE-2021-0109",
        "products": [
            {
                "product": "compute_stick_stk1a32sc",
                "vendor": "intel",
                "version": "-"
            },
            {
                "product": "compute_stick_stk1a32sc_firmware",
                "vendor": "intel",
                "version": "*"
            }
        ],
        "severity_v2": "MEDIUM",
        "severity_v3": "HIGH"
    }
]
```

Je me suis alors tourné vers un langage de programmation fonctionnelle, Elixir, pour manipuler chaque fichier en parallèle, afin d'accélérer son traitement automatique.

Une fois mes requêtes Cypher prêtes, j'ai enfin pu expérimenter l'utilisation d'un driver et d'une très légère librairie pour connecter mon futur client web à Neo4J

Client Web

J'ai décidé de continuer sur ma lancée avec Elixir en utilisant son framework web Phoenix et une de ses "nouvelles" fonctionnalités qu'est LiveView. Cela permet entre autre de réaliser des sites web très réactifs sans vraiment se soucier d'un frontend à la React ou Angular. Pas d'Ajax non plus, en fait tout se déroule via des websockets et du HTML généré côté serveur.

Vous pouvez en apprendre plus ici : https://phoenixframework.org/

Requêtes

On commence par créer nos contraintes d'unicité, qui vont aussi nous permettre de générer des index dessus.

```
CREATE CONSTRAINT ON (cve:CVE) ASSERT cve.id IS UNIQUE;
CREATE CONSTRAINT ON (vendor:VENDOR) ASSERT vendor.name IS UNIQUE;
CREATE CONSTRAINT ON (severity_v2:SEVERITY_V2) ASSERT severity_v2.severity IS UNIQUE;
CREATE CONSTRAINT ON (severity_v3:SEVERITY_V3) ASSERT severity_v3.severity IS UNIQUE;
CREATE CONSTRAINT ON (product:PRODUCT) ASSERT product.name IS UNIQUE;
CREATE CONSTRAINT ON (version:PRODUCT_VERSION) ASSERT version.name IS UNIQUE;
```

On va ensuite "énumérer" notre tableau contenant le nom des différents fichiers json, puis les charger en mémoire en utilisant apoc :

```
UNWIND ['output-nvdcve-1.1-2021.json'] as filename

CALL apoc.load.json(filename) YIELD value AS entry
```

On entrera ensuite assez simplement le CVE en lui même :

```
MERGE (cve:CVE {id: entry.id})
    SET cve.description = entry.description

MERGE (severity_v2:SEVERITY_V2 {severity: coalesce(entry.severity_v2, "Unknown")})
MERGE (severity_v3:SEVERITY_V3 {severity: coalesce(entry.severity_v3, "Unknown")})

MERGE (cve) -[:SCORED]-> (severity_v2)
MERGE (cve) -[:SCORED]-> (severity_v3)
```

La petite surprise qui m'attendait était que lorsqu'une faille est finalement marquée comme étant une erreur et non pas un problème de sécurité, le CVE se voit retiré certains de ses attributs mais reste conservé dans les bases de données.

C'est en soit une très bonne chose, mais ce n'est pas une pratique que j'ai eu l'occasion de voir souvent. C'est pour quoi on remplacera la non existence de ces valeurs par un Unknown.

```
FOREACH (entry_product IN entry.products |
    MERGE (vendor:VENDOR {name: entry_product.vendor})
    MERGE (product:PRODUCT {name: entry_product.product})
    MERGE (version:PRODUCT_VERSION {name: entry_product.vendor + '_' + entry_product.product + '_' + entry_product.version})

    MERGE (cve) -[:FOR_PRODUCT]-> (product)
    MERGE (product) -[:FROM]-> (vendor)
    MERGE (cve)-[:FOR_VERSION]-> (version)
    MERGE (version)-[:OF]-> (product)
)
```

On finit ensuite par enregistrer les différents produits touchés par le CVE, ainsi que leurs vendeurs et leurs versions, via un for each sur le tableau des produits affectés.

Requêtes

Quels sont les fournisseurs ayant le plus de CVE

La requête est la suivante :

```
MATCH (v:VENDOR)<--(p:PRODUCT)<--(c:CVE)
RETURN count(c) AS nb_cve, v.name  ORDER BY  nb_cve DESC
```

Cela nous donne un résultat tel que celui-ci :

	nb_cve	v.name
1	6638	"qualcomm"
2	2725	"cisco"
3	1970	"microsoft"
4	1672	"asus"
5	1143	"netgear"
6	531	"juniper"

Fournisseur ayant le plus de cve par produit

La requête est la suivante :

```
MATCH (v:VENDOR)<-[:FROM]-(p:PRODUCT)<-[r:FOR_PRODUCT]-(c:CVE)
RETURN v.name, p.name, count(r) as nb  ORDER BY nb DESC
```

	v.name	p.name	nb
1	"fedoraproject"	"fedora"	244
2	"microsoft"	"windows_10"	240
3	"microsoft"	"windows_server_2016"	238
4	"microsoft"	"windows_server_2019"	217
5	"microsoft"	"windows"	198
6	"google"	"android"	192

Groupé par sévérité

```
MATCH (v:VENDOR)<-[:FROM]-(p:PRODUCT)<--(c:CVE)-[r:SCORED]->(s:SEVERITY_V3)
RETURN v.name, collect(p.name), s.severity, count(r) as nb  ORDER BY nb DESC
```

Ici on va utiliser la fonction d'aggrégation collect pour rassembler tous les produits dans la même colonne.

Cela nous affichera quelque chose dans ce genre là :

	v.name	collect(p.name)	s.severity	nb
1	"qualcomm"	["fsm10056_firmware", "fsm10056_firmware", "fsm10056_firmware", "qca4020_firmware", "qca4020_firmware", "qpa5373", "qpa5373", "qpa5373", "qpa5373", "apq8064au_firmware", "sd8c", "sd8c", "sd8c", "sd8c", "sd8c", "sd8c", "sd8c", "pm8150l_firmware", "pm8150l_firmware", "pm8150l_firmware", "pm8150l_firmware", "pm8150l_firmware", "pm8150l_firmware", "pm8150l_firmware", "sd710", "sd710", "sd710", "sd710", "sd710", "sd710", "sdxr2_5g", "qet6110", "qet6110", "qet6110", "qet6110", "qet6110", "qet6110", "qca8072", "qca8072", "qca8072", "qcn5164_firmware", "qcn5164_firmware", "qcn5164_firmware", "qcn5164_firmware", "msm8917", "msm8917", "msm8917", "msm8917", "qca8081", "qca8081", "qca8081", "qca8081", "qca8081", "mdm9607", "pm8250_firmware", "pm8250_firmware", "pm8250_firmware", "pm8250_firmware", "pm8250_firmware", "pm8250_firmware", "qcn7605_firmware", "qcn7605_firmware", "qcn7605_firmware", "qcn7605_firmware", "qat5533_firmware", "qat5533_firmware", "qat5533_firmware", "qat5533_firmware", "qat5533_firmware", "qat5533_firmware", "qat3522", "qat3522", "qat3522", "qat3522", "qat3522", "qat3522", "qca6174a", "qca6174a", "qca6174a", "qca6174a", "qca6174a", "qca6174a", "pm640a_firmware", "pm640a_firmware", "pm640a_firmware", "pm640a_firmware", "pm640a_firmware", "pm640a_firmware", "wcn3620", "wcn3620", "wcn3620", "wcn3620", "qca6595au", "qca6595au", "qca6595au", "qca6595au", "qca6595au", "qca6595au", "qca6595au", "sd480", "sd480", "sd480", "sd480", "sd480", "sd480", "csrb31024", "csrb31024", "csrb31024", "csrb31024", "csrb31024", "rsw8577_firmware", "rsw8577_firmware", "rsw8577_firmware", "rsw8577_firmware", "rsw8577_firmware", "rsw8577_firmware", "qpa4340_firmware", "qpa4340_firmware", "qpa4340_firmware", "qpa4340_firmware", "qpa4340_firmware", "qpa4340_firmware", "qdm2305_firmware", "qdm2305_firmware", "qdm2305_firmware", "qdm2305_firmware", "qdm2305_firmware", "qdm2305_firmware", "qdm2310_firmware", "qdm2310_firmware", "qdm2310_firmware", "qdm2310_firmware", "qdm2310_firmware", "qdm2310_firmware", "qdm5677_firmware", "qdm5677_firmware", "qdm5677_firmware", "qdm5677_firmware", "qdm5677_firmware", "qdm5677_firmware", "pmr525_firmware", "pmr525_firmware", "pmr525_firmware", "pmr525_firmware", "pmr525_firmware", "pmr525_firmware", "pmx24_firmware", "pmx24_firmware", "pmx24_firmware", "pmx24_firmware", "pmx24_firmware", "pmx24_firmware"	"HIGH"	5092

Groupé par sévérité, les 2 versions

```
MATCH (v:VENDOR)<-[:FROM]-(p:PRODUCT)<--(c:CVE)-[rv3:SCORED]->(s1:SEVERITY_V3),
(c)-[rv2]->(s2:SEVERITY_V2)
RETURN v.name, collect(p.name), s1.severity, s2.severity, (count(rv2) + count(rv3)) as nb
ORDER BY nb DESC
```

	v.name	collect(p.name)	s1.severity	s2.severity	nb
1	"qualcomm"	["fsm10056_firmware", "fsm10056_firmware", "fsm10056_firmware", "qca4020_firmware", "qca4020_firmware", "qpa5373", "qpa5373", "qpa5373", "apq8064au_firmware", "sd8c", "sd8c", "sd8c", "sd8c", "sd8c", "sd8c", "sd8c", "pm8150l_firmware", "pm8150l_firmware", "pm8150l_firmware", "pm8150l_firmware", "pm8150l_firmware", "pm8150l_firmware", "sd710", "sd710", "sd710", "sd710", "sd710", "sdxr2_5g", "qet6110", "qet6110", "qet6110", "qet6110", "qet6110", "qet6110", "qca8072", "qca8072", "qca8072", "qcn5164_firmware", "qcn5164_firmware", "qcn5164_firmware", "qcn5164_firmware", "msm8917", "msm8917", "msm8917", "qca8081", "qca8081", "qca8081", "qca8081", "qca8081", "mdm9607", "pm8250_firmware", "pm8250_firmware", "pm8250_firmware", "pm8250_firmware", "pm8250_firmware", "pm8250_firmware", "qcn7605_firmware", "qcn7605_firmware", "qcn7605_firmware", "qcn7605_firmware", "qat5533_firmware", "qat5533_firmware", "qat5533_firmware", "qat5533_firmware", "qat5533_firmware", "qat5533_firmware", "qat3522", "qat3522", "qat3522", "qat3522", "qca6174a", "qca6174a", "qca6174a", "qca6174a", "qca6174a", "pm640a_firmware", "pm640a_firmware", "pm640a_firmware", "pm640a_firmware", "pm640a_firmware", "wcn3620", "wcn3620", "wcn3620", "qca6595au", "qca6595au", "qca6595au", "qca6595au", "qca6595au", "qca6595au", "sd480", "sd480", "sd480", "sd480", "sd480", "csrb31024", "csrb31024", "csrb31024", "csrb31024", "rsw8577_firmware", "rsw8577_firmware", "rsw8577_firmware", "rsw8577_firmware", "rsw8577_firmware", "rsw8577_firmware", "qpa4340_firmware", "qpa4340_firmware", "qpa4340_firmware", "qpa4340_firmware", "qpa4340_firmware", "qpa4340_firmware", "qdm2305_firmware", "qdm2305_firmware", "qdm2305_firmware", "qdm2305_firmware", "qdm2305_firmware", "qdm2305_firmware", "qdm2310_firmware", "qdm2310_firmware", "qdm2310_firmware", "qdm2310_firmware", "qdm2310_firmware", "qdm2310_firmware", "qdm5677_firmware", "qdm5677_firmware", "qdm5677_firmware", "qdm5677_firmware", "qdm5677_firmware", "qdm5677_firmware", "pmr525_firmware", "pmr525_firmware", "pmr525_firmware", "pmr525_firmware", "pmr525_firmware", "pmr525_firmware", "pmx24_firmware", "pmx24_firmware", "pmx24_firmware", "pmx24_firmware", "pm6150a", "pm6150a", "pm6150a", "pm6150a", "sd835_firmware", "sd835_firmware", "sd835_firmware"	"HIGH"	"HIGH"	8528

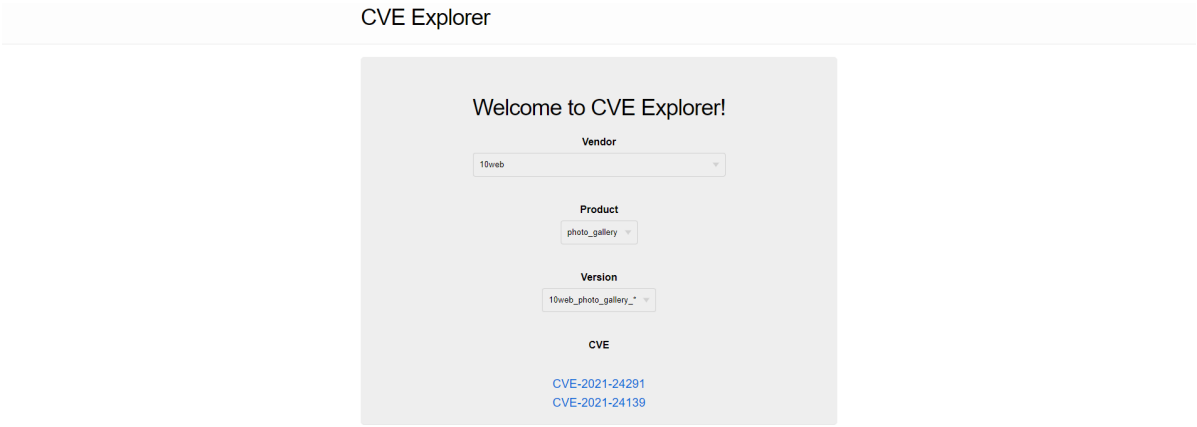
La répartition par sévérité

Ici, de même, elle sera ordonnée par les deux sévérités.

```
MATCH (c:CVE)-->(sv2:SEVERITY_V2) MATCH (c)-->(sv3:SEVERITY_V3)
RETURN collect(c.id) as CVE_List, count(c.id) AS number_of_cve, sv2.severity, sv3.severity
ORDER BY sv3.severity, sv2.severity
```

	CVE_List	number_of_cve	sv2.severity	sv3.severity
1	["CVE-2021-31875", "CVE-2021-21321", "CVE-2021-26893", "CVE-2021-27573", "CVE-2021-25669", "CVE-2021-29937", "CVE-2021-26689", "CVE-2021-1295", "CVE-2021-24222", "CVE-2021-1994", "CVE-2021-3325", "CVE-2021-31914", "CVE-2021-26895", "CVE-2021-21018", "CVE-2021-28031", "CVE-2021-20618", "CVE-2021-26201", "CVE-2021-23369", "CVE-2021-27258", "CVE-2021-22987", "CVE-2021-1871", "CVE-2021-24171", "CVE-2021-27113", "CVE-2021-0396", "CVE-2021-20588", "CVE-2021-25289", "CVE-2021-26956", "CVE-2021-30176", "CVE-2021-30234", "CVE-2021-3021", "CVE-2021-0266", "CVE-2021-21978", "CVE-2021-1796", "CVE-2021-27177", "CVE-2021-1627", "CVE-2021-2075", "CVE-2021-27372", "CVE-2021-20998", "CVE-2021-27103", "CVE-2021-21242", "CVE-2021-26918", "CVE-2021-28300", "CVE-2021-31414", "CVE-2021-26809", "CVE-2021-3401", "CVE-2021-27329", "CVE-2021-27357", "CVE-2021-1626", "CVE-2021-28793", "CVE-2021-1722", "CVE-2021-27164", "CVE-2021-29465", "CVE-2021-0248", "CVE-2021-26295", "CVE-2021-25907", "CVE-2021-1910", "CVE-2021-20078", "CVE-2021-21322", "CVE-2021-3286", "CVE-2021-25928", "CVE-2021-21386", "CVE-2021-23370", "CVE-2021-20623", "CVE-2021-27101", "CVE-2021-32099", "CVE-2021-3148", "CVE-2021-22505", "CVE-2021-1291", "CVE-2021-27155", "CVE-2021-27161", "CVE-2021-3331", "CVE-2021-26955", "CVE-2021-1140", "CVE-2021-25346", "CVE-2021-28305", "CVE-2021-27707", "CVE-2021-3190", "CVE-2021-31737", "CVE-2021-25847", "CVE-2021-29145", "CVE-2021-26822", "CVE-2021-29940", "CVE-2021-25900", "CVE-2021-25758", "CVE-2021-27817", "CVE-2021-3199", "CVE-2021-23352", "CVE-2021-26703", "CVE-2021-21505", "CVE-2021-27215", "CVE-2021-27171", "CVE-2021-21477", "CVE-2021-24078", "CVE-2021-28132", "CVE-2021-23374", "CVE-2021-21016", "CVE-2021-28671", "CVE-2021-27114", "CVE-2021-22714", "CVE-2021-21335", "CVE-2021-2256", "CVE-2021-21513", "CVE-2021-31726", "CVE-2021-3144", "CVE-2021-28123", "CVE-2021-30231", "CVE-2021-27730", "CVE-2021-3188", "CVE-2021-20716", "CVE-2021-1292", "CVE-2021-31757", "CVE-2021-0316", "CVE-2021-3033", "CVE-2021-21347", "CVE-2021-27710", "CVE-2021-23330", "CVE-2021-27514", "CVE-2021-3118", "CVE-2021-27691", "CVE-2021-23344", "CVE-2021-28668", "CVE-2021-21344", "CVE-2021-27158", "CVE-2021-23378", "CVE-2021-20711", "CVE-2021-30168", "CVE-2021-26797", "CVE-2021-28879", "CVE-2021-3210", "CVE-2021-1498", "CVE-2021-3110", "CVE-2021-28381", "CVE-2021-26305", "CVE-2021-3177", "CVE-2021-20617", "CVE-2021-26957", "CVE-2021-27314", "CVE-2021-3406", "CVE-2021-30000", "CVE-2021-28797", "CVE-2021-27156", "CVE-2021-30072", "CVE-2021-3122", "CVE-2021-26897", "CVE-2021-21307", "CVE-2021-22679", "CVE-2021-23899", "CVE-2021-25149", "CVE-2021-29369", "CVE-2021-3346", "CVE-2021-1870", "CVE-2021-23375", "CVE-2021-27135", "CVE-2021-31755", "CVE-2021-22667", "CVE-2021-1388", "CVE-2021-27236", "CVE-2021-25833", "CVE-2021-27130", "CVE-2021-27877", "CVE-2021-2029", "CVE-2021-27706", "CVE-2021-30457", "CVE-2021-26840", "CVE-2021-30473", "CVE-2021-22850", "CVE-2021-25832", "CVE-2021-20655", "CVE-2021-1468", "CVE-2021-30464", "CVE-2021-27440", "CVE-	485	"HIGH"	"CRITICAL"

Client Web



Installation

Il vous faudra (bien sûr) Neo4J : https://neo4j.com/download-neo4j-now/

Vous aurez ensuite besoin d'Elixir : https://elixir-lang.org/install.html

Puis de Phoenix : https://hexdocs.pm/phoenix/installation.html#content

et de NodeJS pour la partie assets : https://nodejs.org/en/download/

Il faudra ensuite soit dézipper soit cloner le repository du projet : https://github.com/AdrianPaulCarrieres/lpiot2020-neo4j-cve-adrianpaulcarrieres

Utilisation

Converter

Télécharger un ou plusieurs fichiers venant de : https://nvd.nist.gov/vuln/data-feeds

Nous pourrons notamment utiliser : https://nvd.nist.gov/feeds/json/cve/1.1/nvdcve-1.1-2021.json.zip

Extraire l'archive et placer le ou les fichiers json à la racine du projet.

Il faudra ensuite ouvrir le fichier converter\lib\converter\application.ex et modifier la ligne suivante 20 pour ajouter ou supprimer les fichiers à convertir.

On aura besoin d'installer les librairies depuis la racine du dossier convert en tapant :

```
mix deps.get
```

Sauvegarder le fichier et lancer le tout en tapant (toujours depuis la racine du dossier converter)

Cela devrait compiler les librairies et l'application puis la lancer en mode interactive.

Pour Windows dans Powershell :

```
iex.bat -S mix
```

Pour le cmd et sous Linux :

```
iex -S mix
```

Il faudra ensuite attendre (de quelques secondes à j'imagine plusieurs minutes en fonction du nombre de fichiers et de la puissance de la machine) pour voir apparaître les lignes suivantes :

```
:ok
:ok
:ok
```

Il devrait y avoir autant de :ok que de fichiers lus, mais il n'y a pas vraiment à les compter, les :ok n'apparaissant qu'une fois tous les traitements terminés.

On devrait voir apparaître les fichiers output à la racine également.

L'étape suivante est donc de déplacer ces fichiers dans le dossier import de Neo4J et de le lancer le script (morceau par morceau) donné précédemment (ou trouvable à la racine du repository).

CVE Explorer

On modifie en premier lieu le fichier de configuration pour accéder à la base Neo4J :

cve_explorer\config\config.exs à la ligne 27 pour l'adresse et le port bolt et la ligne 28 pour l'authentification à la base de données si elle a lieu (l'user par défaut de Neo4J desktop étant "neo4j").

Depuis la racine du dossier cve_explorer :

```
mix deps.get
cd assets && npm install
cd ..
mix phx.server
```

Cela installera les dépendances et les assets puis lancera le serveur.

On pourra accéder à l'application depuis localhost:4000

Vidéo de présentation

Vous pourrez trouver une vidéo résumant et surtout présentant mon projet à l'adresse suivante :

https://youtu.be/Ru6vFNNuyog