Refonte de la rétroaction d’un programme de détection de vulnérabilité

Rapport final présenté à

Professeur Raphaël Khoury - Superviseur,

Professeur Karim El Guemhioui - Coordonnateur

par

Abdel-Gany Jr Odedele (ODEA82070001)

dans le cadre du cours

INF4173 Projet Synthèse

Département d’informatique et d’ingénierie

Université du Québec en Outaouais

21 avril 2023

Table of Contents

Résumé

Prévenir des vulnérabilités dans du code est une tâche en constante fluctuation. De nouveaux points d'attaque sont constamment découverts. Rectifier les vecteurs de pénétration est accomplie à l'aide de collaboration sous la forme d'une liste de faiblesse détectée. Cette liste communément appelée *Common Vulnerabilities and Exposure* ou *CVE*, aide les professionnels à coordonner leurs efforts dans le but de corriger les erreurs et rendre leurs systèmes plus sécuritaires.

Une trentaine de ces CVE ont été analysés par des étudiants durant leur projet de synthèse. Ils ont conduit un banc de test de code vulnérable à une classe de vulnérabilités particulière : les erreurs d'utilisation des librairies cryptographiques. Pour accomplir leur analyse, ils ont utilisé des outils de test statique. Tout ce processus a été documenté dans leur rapport [1].

Le test statique de sécurité des applications est un outil de sécurité des applications fréquemment utilisées, qui analyse le code source, les binaires ou les octets d'une application. Cet outil de test en boîte blanche identifie la cause profonde des vulnérabilités et aide à remédier aux failles de sécurité présentes dans le code. Ces solutions analysent une application de l'« intérieur » et n'ont pas besoin d'un système en cours d'exécution pour effectuer la recherche.

Dans leur rapport, les auteurs ont fait l'observation qu'une bonne portion des vulnérabilités retournées par leurs outils d'analyse statique manquaient de spécificité, ou ne prennent pas en compte certaines erreurs se trouvant dans la signature des fonctions, entre autres. Pour ces raisons, ils ont obtenu « un taux de succès de 3.33%. »[1]

L'objectif de ce projet est donc d'améliorer les messages d'erreurs obtenues des logiciels d'analyse de vulnérabilité de code, faire la modification des règles d'analyse et potentiellement augmenter le pourcentage de succès de la détection. Cet objectif est basé sur la recommandation contenue dans le document du projet précédent. Cette tâche sera accomplie à l'aide de la recherche de mots-clés et patrons qui prédisent la présence de vulnérabilités spécifiques. Cette récolte sera ensuite appliquée aux outils pour une analyse plus complète.

À l'aide de Visual Code Grepper (VCG), un outil d'analyse statique très limité, il est possible d'obtenir un taux de succès de 30% ou 9 sur les 30 CVE analysés durant la recherche de Jérémy Bolduc et Jason Lafrenière Nickopoulos. Ceci est une augmentation de 26.66%.

Une portion manquante à cette recherche due à certaines contraintes est l'évaluation de la qualité de ces détections à l'aide d'une enquête sur une population de programmeur.

Introduction

L'objectif de ce projet est d'améliorer les messages d'erreurs obtenues d'un logiciel d'analyse de vulnérabilité de code, faire la modification des règles d'analyses et potentiellement augmenter le pourcentage de succès de la détection. Cet objectif est basé sur la recommandation contenue dans le document du projet précédent. Cette tâche sera accomplie à l'aide de la recherche de mots-clés et patrons qui prédissent la présence de vulnérabilités spécifiques. Cette récolte sera ensuite appliquée à l'outil pour une analyse plus complète. Ce rapport présente tout le travail fait dans le but d'accomplir l'objectif décrit dans le paragraphe précédent.

Description des étapes accomplies

Familiarisation avec le travail accompli dans le projet précédent

Cette étape a inclus la lecture du rapport du projet de l'hiver précédent [1] sur lequel celui-ci est bâti, la lecture des sources primaires qui y sont citées et l'obtention de tout le code qui a été analysé dans le projet antérieur.

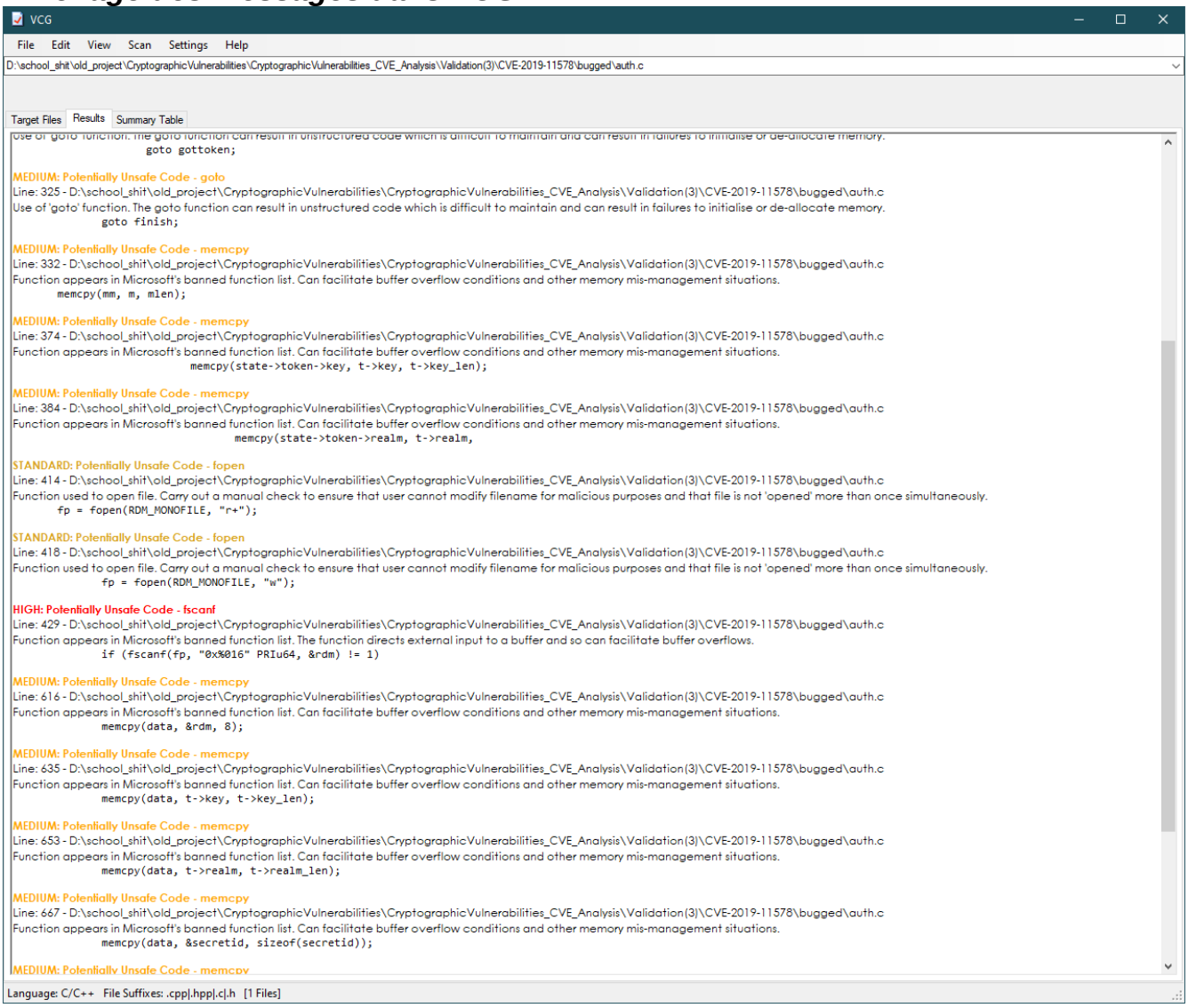
L'outil d'analyse de vulnérabilité de code et analyse des mots-clés, du code désigné vulnérable

Cette étape a nécessité l'installation de certains des outils d'analyse utilisés dans le projet précédent [1]. La recherche de la méthode de modification des résultats des outils a été nécessaire dans cette portion du projet. Le filtrage du code obtenu durant la recherche antérieure a aussi été accompli.

L'outil qui a été choisi est VisualCodeGrepper (VCG) [2] parce qu'il est un outil assez simple pour pouvoir représenter le pire cas de l'analyse statique. D'après une analyse du code source du programme, il ne supporte pas de langages de programmation autre C/C++, Java, C#, VB et PL/SQL. On ne peut pas lui donner d'autres langages de programmation parce qu'il fait des vérifications initiales spécifiques à la syntaxe de la langue sélectionnée. Une autre limitation est le manque de support pour les expressions régulières qui empêche la recherche pour certains types d'erreurs cryptographiques tels que des valeurs hard-codés.

En se basant sur les limitations de VCG [2], les failles associées à la cryptographie présentes dans le projet antécédent qui peuvent-être détecter par l'outil ont été simple à déceler.

Affichage des messages dans VCG



Formatage des règles d'analyse de VCG

```
mot-clé[=>][[Niveau d'importance]][Message affiché]
```

Niveau	Signification
1	Critique
2	Sévère
3	Moyen
0	Normal

Les types de CVE

Le type de CVE est défini de la même manière que l'ancien projet. Les types définis dans le *framework Tafelsalz* [3].

1. *Initialization*
    a. *Predictable Sequences*
    b. *Re-use*

    c. *Weak Values*
    d. *Source*
2. *Insecure Defaults*
3. *Weak Algorithms*
4. *Validation*
5. *Persistence of Secrets*
6. *Usage Complexity*
7. *Knowledge Base*
8. *Identity Management*
9. *Password Hashing*
10. *Other*

Les CVEs de l'ancien projet détectable avec VCG

À l'aide d'outil d'analyse statique, il est possible de détecter 17 sur les 30 vulnérabilités analysées dans le projet de l'hiver passé [1].

De ces détections, 9 peuvent être accomplies avec VCG [2].

La majorité des vulnérabilités en lien avec des séquences prédictibles sont détectables, mais le total est de 5/12 à cause du nombre limité de langages supportés par VCG [2]. Le nombre monte à 11/12 si cette restriction n'est pas prise en compte.

- Predictable sequences (5/12)
    - CVE-2014-5386: https://www.vulncode-db.com/CVE-2014-5386
        - C++ (On peut détecter le mot-clé rand()
    - CVE-2015-8867: https://www.cvedetails.com/cve/CVE-2015-8867/
        - C (On peut détecter le mot-clé RAND_pseudo_bytes)
    - CVE-2018-12520: https://www.cvedetails.com/cve/CVE-2018-12520/
        - C++ (On peut détecter le mot-clé rand()
    - CVE-2019-11808: https://www.cvedetails.com/cve/CVE-2019-11808/
        - Java On peut détecter le mot-clé ThreadLocalRandom
    - CVE-2020-12735: https://github.com/domainmod/domainmod/issues/122
        - PHP (On peut détecter le mot-clé md5()
- Re-use (1/2)
    - CVE-2022-1434: https://www.cvedetails.com/cve/CVE-2022-1434/
        - C (On peut détecter le mot-clé MD5)

- Weak values (2/2)
  - CVE-2019-10908: https://github.com/airsonic/airsonic/commit/61c842923a6d60d4aedd126445a8437b53b752c8
    - Java (On peut détecter le mot-clé RandomStringUtils et le mot-clé java.util.Random)
  - CVE-2022-1235 : https://github.com/livehelperchat/livehelperchat/commit/6538d6df3d8a60fee254170b08dd76a161f7bfdc
    - PHP (On peut détecter le mot-clé md5 et le mot-clé mt_rand()
- Insecure defaults (1/2)
  - CVE-2016-1000352&1000344 : https://www.cvedetails.com/cve/CVE-2016-1000352/ https://www.cvedetails.com/cve/CVE-2016-1000344/
    - Java (On peut détecter le mot-clé AESEngine et le mot-clé DefaultMultiBlockCipher et prévenir l'utilisateur à propos de la mode ECB de ces fonctions)

Langage	Nombre de vulnérabilités
C	2
C++	2
Java	3
PHP	2
Total	9

Type de vulnérabilité	Nombre de vulnérabilités
Predictable sequences	5
Re-use	1
Weak values	2
Insecure defaults	1
Total	**9**

Les CVEs de l'ancien projet détectable par l'analyse statique sans les limitations de VCG

Sans les limitations de VCG le nombre de cas évaluables par analyse statique augmente de 8.

- Predictable sequence (6/12)
  - CVE-2011-0766: https://www.vulncode-db.com/CVE-2011-0766
    - Erlang && C (On peut détecter le mot-clé random et le mot-clé irandom)

    - CVE-2012-2417: https://www.vulncode-db.com/CVE-2012-2417
        - Python (On peut détecter le mot-clé bignum(getPrime()
    - CVE-2013-1445: https://www.vulncode-db.com/CVE-2013-1445
        - Python (On peut détecter le mot-clé Crypto.Random.atfork())
    - CVE-2020-28924: https://github.com/rclone/rclone/issues/4783
        - GO (On peut détecter le mot-clé import of math/rand)
    - CVE-2021-3538: https://www.cvedetails.com/cve/CVE-2021-3538/
        - GO (On peut détecter le mot-clé g.rand.Read()
    - CVE-2022-36045: https://www.cvedetails.com/cve/CVE-2022-36045/
        - Javascript (Warning: can be flagged with a warning about the insecurity of Math.random)
- Insecure Defaults (1/2)
    - CVE-2012-3458: https://www.vulncode-db.com/CVE-2012-3458
        - Python (On peut détecter le mot-clé et offrir un avertissement sur le bon usage du module)
- Other (1/4)
    - CVE-2016-10530: https://www.cvedetails.com/cve/CVE-2016-10530/
        - Javascript (On peut détecter le mot-clé http dans le fichier)

Langage	Nombre de vulnérabilités
C	1
Javascript	2
Python	3
Go	2
Total	**8**

Type de vulnérabilité	Nombre de vulnérabilités
Predictable sequences	6
Insecure defaults	1
Autres	1
Total	**8**

Les CVEs de l'ancien projet non détectable par l'analyse statique

- Re-use (0/2)
  - CVE-2019-15075: https://www.cvedetails.com/cve/CVE-2019-15075/
    - PHP Les valeurs statiques varient trop pour être détecté de manière consistante
- Validation (0/3) : Aucune vulnérabilité de type validation ne peut être détectée parce qu'elles nécessitent la détection de code manquant
  - CVE-2016-2053: https://www.vulncode-db.com/CVE-2016-2053
    - C
  - CVE-2019-11578: https://www.cvedetails.com/cve/CVE-2019-11578/
    - C
  - CVE-2021-32738: https://github.com/stellar/js-stellar-sdk/compare/v8.2.2...v8.2.3
    - Typescript (Javascript)
- Usage complexity (0/5) Ce type de problème ne peut pas être évalué par une approche statique parce qu'il est relié à des détails d'implantations qui n'ont pas été pris en compte.
  - CVE-2017-7526: https://www.cvedetails.com/cve/CVE-2017-7526/
    - C
  - CVE-2018-16870: https://www.cvedetails.com/cve/CVE-2018-16870/
    - C
  - CVE-2018-19653: https://www.cvedetails.com/cve/CVE-2018-19653/
    - Go
  - CVE-2019-9155: https://www.cvedetails.com/cve/CVE-2019-9155/
    - Javascript
  - CVE-2020-26263: https://www.cvedetails.com/cve/CVE-2020-26263/
    - Python
- Other (0/4)

    - CVE-2013-2548: https://www.vulncode-db.com/CVE-2013-2548
        - C
    - CVE-2014-3570: https://www.vulncode-db.com/CVE-2014-3570
        - C Impossible pour la même raison que Ies vulnérabilités de type complexité d'usage
    - CVE-2014-8275: https://www.vulncode-db.com/CVE-2014-8275
        - C Impossible pour la même raison que Ies vulnérabilités de type vérification
- Predictable sequence (0/12)
    - CVE-2021-41117: http://m.cvedetails.com/cve/CVE-2021-41117/
        - Javascript (On peut détecter le mot-clé putByte(String.fromCharCode(), mais ce mot-clé est trop générique et causerait une panoplie de faux positifs)

Langage	Nombre de vulnérabilités
C	7
Javascript	3
Python	1
Go	1
PHP	1
Total	**13**

Type de vulnérabilité	Nombre de vulnérabilités
Predictable sequence	1
Re-use	1
Validation	3
Usage complexity	5
Other	3
Total	**13**

Compilation des mots-clés et patrons qui permettent de déceler les vulnérabilités d'utilisation des librairies cryptographiques

Toute l'information récoltée durant la recherche est utilisée pour déterminer les mots-clés et patrons utiles à la détection de vulnérabilité. Elle permet aussi de les classifiés par vulnérabilités spécifiques qu'ils exposent. La compilation de vulnérabilité provenant de librairies cryptographiques populaires a

été faite pour permettre d’appliquer la méthodologie de rédaction de messages d’erreur sur plus de cas du monde réel. Ceci facilite l’évaluation des messages parce qu’ils ne sont pas limités à des CVE spécifiques uniquement. Les librairies et leurs fonctions sont principalement dépréciées. Elles sont contenues dans un fichier Excel [4].

Données du fichier excel

Langage	Nombre de vulnérabilités
C/C++	35
Java	21
C#	5
PHP	10
Total	**71**

Type de vulnérabilité	Nombre de vulnérabilités
Predictable sequence	6
Insecure Defaults	3
Weak Algorithms	16
Deprecated	44
Password Hashing	1
Other	1
Total	**71**

Rédaction des messages spécifiques pour les erreurs que représentent les mots-clés et patrons détectés

Méthodologie de rédaction

Chaque message rédigé respecte les conventions suivantes :

- Doit-être en anglais pour être le plus semblable possible aux messages d’erreurs d’un compilateur et être accessible au plus grand nombre de personne possible.
- Doit mentionner la librairie de provenance d'une méthode s’il est plausible qu'elle fasse partie d'une autre librairie pour éviter de la confusion.
- Certains messages existent sous la forme d'avertissement à propos d'erreurs possibles lors de l'implantation de certains mots-clés.

Par la suite, le niveau de sévérité de la vulnérabilité est déterminé à l’aide d’une comparaison au score d’une vulnérabilité similaire sur National Vulnerability Database [5].

Discussion

Toutes les données ont été compilées dans un fichier Excel [4]. Ce fichier contient les librairies cryptographiques à risque des langages supportés par VCG [2], les CVEs détectables par l'outil d'analyse statique décelé dans les étapes précédentes. Chaque librairie est associée aux mots-clés de détection de leurs vulnérabilités. Les langages COBOL et SQL ne sont pas inclus parce qu'ils ne supportent pas l'inclusion de librairies. Pour faciliter la compilation des règles d'analyse de VCG [2], j'ai créé un script pour générer les fichiers à partir du fichier Excel.

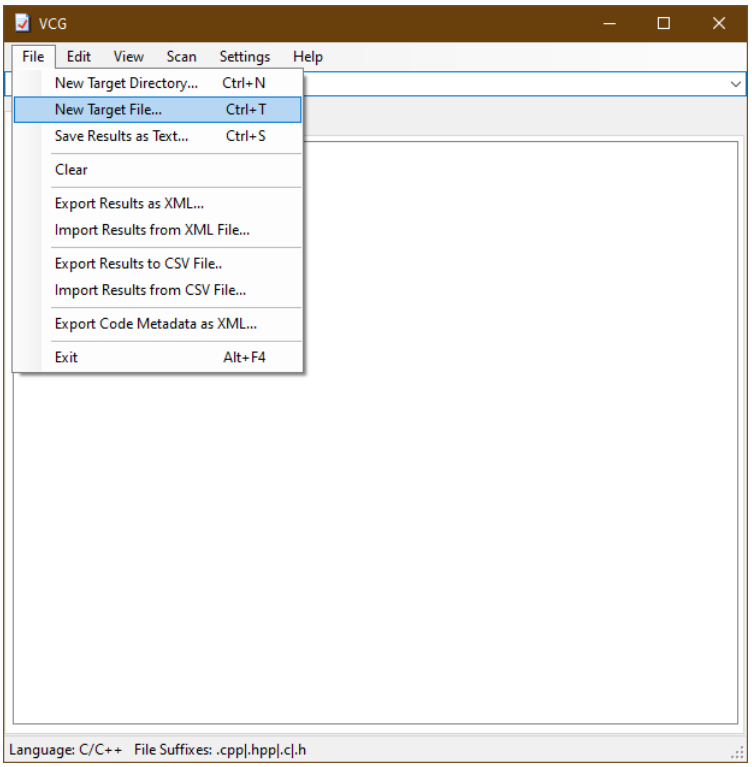
Listage du code (conf_maker.py)

```python
import sys
import os
import pandas as p
# Le script pour créer les fichier.conf à partir du fichier excel
#    Example:
#    python conf_maker.py path\excel_file directory_for_conf_files
#
#    Dependencies:
#    Pandas
def excel_to_conf():
    if len(sys.argv) == 3:
        if(not os.path.isfile(sys.argv[1]) and sys.argv[1].endswith((".xls",".xlt",".xlsx",".xltx"))):
            print("The given file isn't an excel spreadsheet")
            return
        if(not os.path.isdir(sys.argv[2])):
            print("The destination path doesn't exist")
            return
        tabs = p.ExcelFile(sys.argv[1]).sheet_names
        for i in range(len(tabs)):
            df = p.read_excel(sys.argv[1],i)
            if(len(df.axes[0]) > 1 and len(df.axes[1]) > 5):
                try:
                    with open(sys.argv[2]+"\\"+tabs[i]+".conf",'x') as f:
                        for r in range(1,len(df.axes[0])):
                            # column 4 to 6
                            # name[=>][[N]][description]
                            buffer = str(df.iat[r,4]).lstrip() +"=>[" + (str(int(df.iat[r,5])) if df.iat[r,5] == df.iat[r,5] else "0") + "]" + str(df.iat[r,6]).lstrip() +"\n"
                            if(buffer[0:3] != "nan"):
                                f.write(buffer)
                        print(tabs[i]+".conf: creation completed")
                except:
                    print("The file: "+tabs[i]+".conf"+" already exists")
            else:
                print(tabs[i]+".conf: The spreadsheet has no relevant data to parse")

    else :
        print("Incorrect number arguments")
        print("Arg 1:excel file Arg 2:conf file")

excel_to_conf()
```

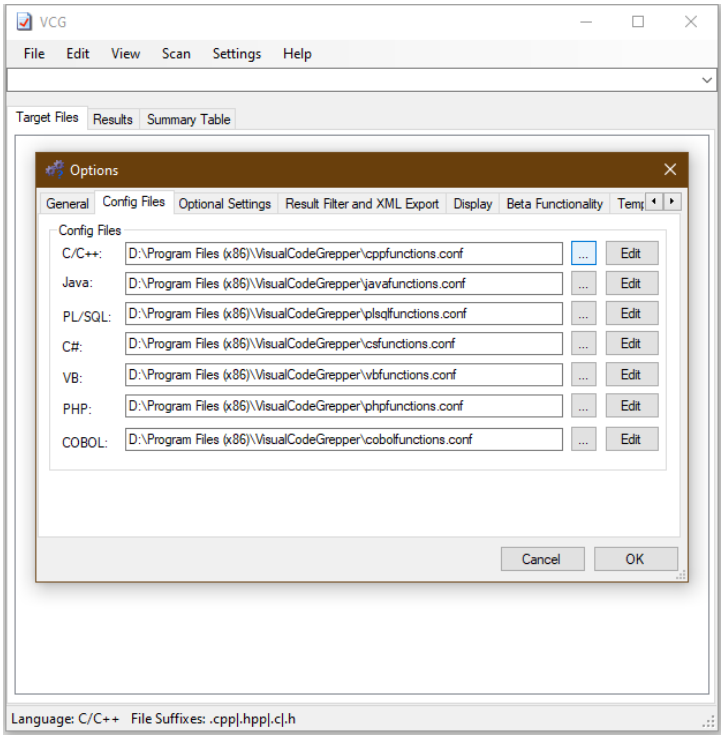
Dépendances : openpyxl et pandas

Appliquer les fichiers

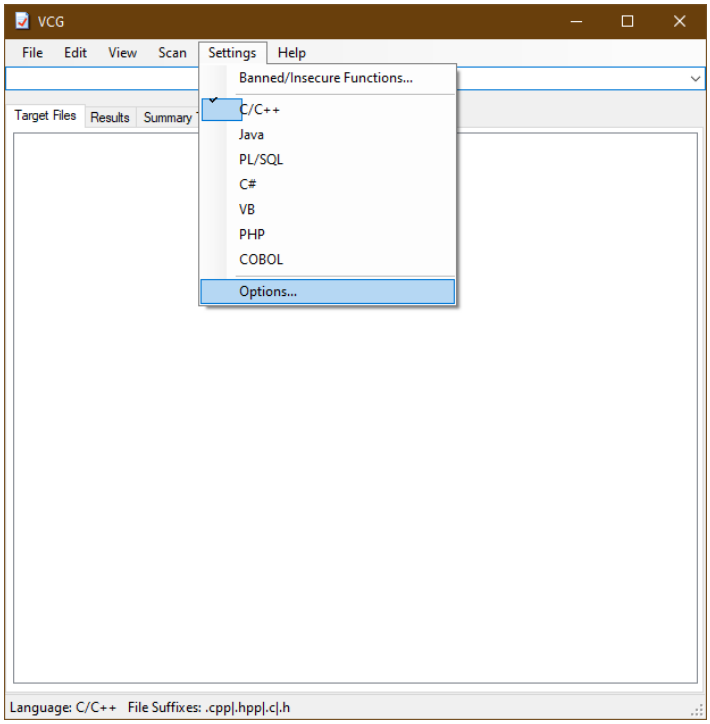
1. Accéder aux options dans VCG



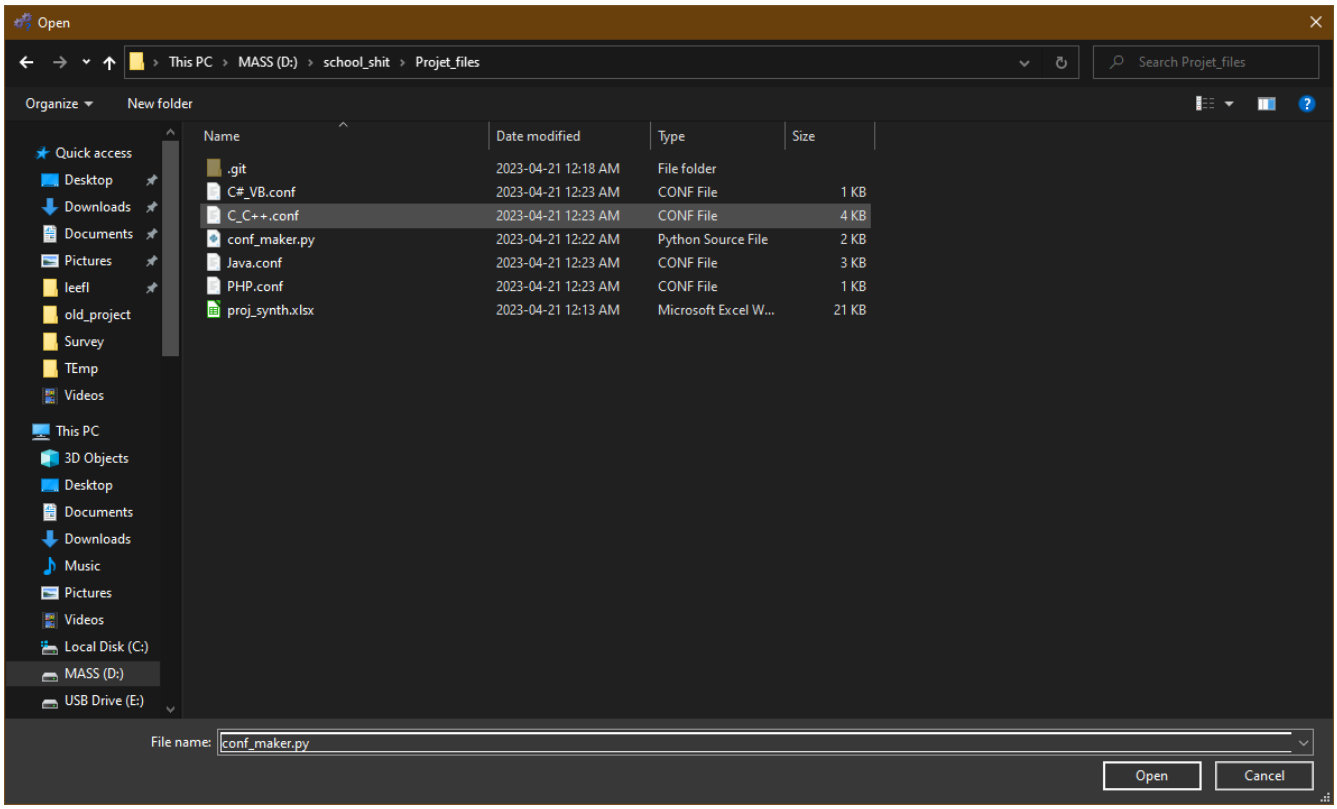
2. Choisir le fichier à remplacer ou modifier



3. Choisir le nouveau fichier



4. VCG peut ensuite être exécuté sur un fichier supporté



5. Les résultats apparaissent dans la section Results comme illustrés dans la section *Affichage des messages dans VCG*

Comparaison avec les messages de l'ancien projet

À la suite de l'ajout de toutes ces règles de détection, il est possible d'observer les changements dans les messages.

Les erreurs qui diffèrent de leur CVE ne seront pas présentées parce qu'elles sont déjà mentionnées dans le projet de l'hiver passé [1].

Liste des messages d'erreurs de vrais positifs

<table>
<tr><th>CVE</th><th>Messages d'erreurs</th></tr>
<tr><td>CVE-2022-1235</td><td>Ancien message :

MEDIUM: Potentially Unsafe Code - md5
Line: 124 - C:\Users\Jeremy\Downloads\lhc+web_modules_lhinstall_install.php
MD5 Hashing algorithm.
$cfgSite->setSetting('site', 'secrethash',
(!empty(getenv('LHC_SECRET_HASH')) ? getenv('LHC_SECRET_HASH') : substr(md5(time() . ":" .
mt_rand()),0,10)));

STANDARD: Potentially Unsafe Code - Use of Deterministic Pseudo-Random Values
Line: 124 - C:\Users\Jeremy\Downloads\lhc+web_modules_lhinstall_install.php
The code appears to use the mt_rand function. The resulting values, while appearing random to a casual observer, are predictable and may be enumerated by a skilled and determined attacker, although this is partly mitigated by a seed that does not appear to be time-based.
$cfgSite->setSetting('site', 'secrethash',
(!empty(getenv('LHC_SECRET_HASH')) ? getenv('LHC_SECRET_HASH') : substr(md5(time() . ":" .
mt_rand()),0,10)));

Nouveau message

MEDIUM: Potentially Unsafe Code - md5
Line: 74 - D:\school_shit\old_project\CryptographicVulnerabilities
\CryptographicVulnerabilities_CVE_Analysis\Initialisation(16)\WeakValues(2)\CVE-2022-1235
\bugged\lhc+web_cli_lib_install.php
MD5 Hashing algorithm.
$cfgSite->setSetting('site', 'secrethash', substr(md5(time() . ":" .
mt_rand()),0,10));

STANDARD: Potentially Unsafe Code - Use of Deterministic Pseudo-Random Values
Line: 74 - D:\school_shit\old_project\CryptographicVulnerabilities
\CryptographicVulnerabilities_CVE_Analysis\Initialisation(16)\WeakValues(2)\CVE-2022-1235
\bugged\lhc+web_cli_lib_install.php
The code appears to use the mt_rand function. The resulting values, while appearing random to a casual observer, are predictable and may be enumerated by a skilled and determined attacker, although this is partly mitigated by a seed that does not appear to be time-based.
$cfgSite->setSetting('site', 'secrethash', substr(md5(time() . ":" .
mt_rand()),0,10));</td></tr>
<tr><td>CVE-2014-5386</td><td>Ancien message:
N/A
Nouveau message:</td></tr>
</table>

	MEDIUM: Potentially Unsafe Code - rand Line: 379 - D:\school_shit\old_project\CryptographicVulnerabilities\CryptographicVulnerabilities_CVE_Analysis\Initialisation(16)\PredictableSequences(12)\CVE-2014-5386(FromPythonCode)\buggy\buggy_ext_mcrypt.cpp rand() is not cryptographically secure `iv[--size] = (char)(255.0 * rand() / RAND_MAX);`
CVE-2018-12520	Ancien message: N/A Nouveau message: MEDIUM: Potentially Unsafe Code - RAND_pseudo_bytes(Line: 5101 - D:\school_shit\old_project\CryptographicVulnerabilities\CryptographicVulnerabilities_CVE_Analysis\Initialisation(16)\PredictableSequences(12)\CVE-2015-8867\bugged\openssl.c RAND_pseudo_bytes() is not cryptographically secure, use RAND_bytes() instead `if ((strong_result = RAND_pseudo_bytes(buffer, buffer_length)) < 0) {`
CVE-2019-11808	Ancien message: N/A Nouveau message: MEDIUM: Potentially Unsafe Code - rand Line: 144 - D:\school_shit\old_project\CryptographicVulnerabilities\CryptographicVulnerabilities_CVE_Analysis\Initialisation(16)\PredictableSequences(12)\CVE-2018-12520\bugged\HTTPserver.cpp rand() is not cryptographically secure `snprintf(random, sizeof(random), "%d", rand());`

CVE-2020-12735	Ancien message:
	N/A
	Nouveau message:
	HIGH: ThreadLocalRandom
	The class is not cryptographically secure
	Line: 23 - Filename: D:\school_shit\old_project\CryptographicVulnerabilities\CryptographicVulnerabilities_CVE_Analysis\Initialisation(16)\PredictableSequences(12)\CVE-2019-11808\bugged\DefaultSessionIdGenerator.java
	`import java.util.concurrent.ThreadLocalRandom;`
	
	HIGH: ThreadLocalRandom
	The class is not cryptographically secure
	Line: 28 - Filename: D:\school_shit\old_project\CryptographicVulnerabilities\CryptographicVulnerabilities_CVE_Analysis\Initialisation(16)\PredictableSequences(12)\CVE-2019-11808\bugged\DefaultSessionIdGenerator.java
	`ThreadLocalRandom random = ThreadLocalRandom.current();`

CVE-2022-1434	Ancien message: N/A Nouveau message: MEDIUM: Potentially Unsafe Code - md5 Line: 69 - D:\school_shit\old_project\CryptographicVulnerabilities\CryptographicVulnerabilities_CVE_Analysis\Initialisation(16)\PredictableSequences(12)\CVE-2020-12735\bugged\reset.php MD5 Hashing algorithm. `$new_password = substr(md5(time()), 0, 8);`
CVE-2019-10908	Ancien message: N/A Nouveau message:

	HIGH: Potentially Unsafe Code - RandomStringUtils Line: 7 - D:\school_shit\old_project\CryptographicVulnerabilities\CryptographicVulnerabilities_CVE_Analysis\Initialisation(16)\WeakValues(2)\CVE-2019-10908(FromPythonCode)\buggy\buggy_RecoverController.java This class uses java.util.Random under the hood which can be bruteforced, use java.security.SecureRandom. `import org.apache.commons.lang.RandomStringUtils;`

CVE-2016-1000352&1000344	Ancien message:
	N/A
	Nouveau message:
	HIGH: AESEngine
	This class defaults to ECB mode, which is unsafe.
	Line: 563 - Filename: D:\school_shit\old_project\CryptographicVulnerabilities\CryptographicVulnerabilities_CVE_Analysis\InsecureDefaults(2)\CVE-2016-1000344&1000352\bugged\ec_IESCipher.java
	`super(new CBCBlockCipher(new AESEngine()), 16);`
	
	HIGH: AESEngine
	This class defaults to ECB mode, which is unsafe.
	Line: 437 - Filename: D:\school_shit\old_project\CryptographicVulnerabilities\CryptographicVulnerabilities_CVE_Analysis\InsecureDefaults(2)\CVE-2016-1000344&1000352\bugged\AESEngine.java
	`public AESEngine()`
	
	HIGH: AESEngine
	This class defaults to ECB mode, which is unsafe.
	Line: 27 - Filename: D:\school_shit\old_project\CryptographicVulnerabilities\CryptographicVulnerabilities_CVE_Analysis\InsecureDefaults(2)\CVE-2016-1000344&1000352\bugged\dh_IESCipher.java
	`import org.bouncycastle.crypto.engines.AESEngine;`

Difficultés rencontrées

Les difficultés rencontrées ont été reliées à trouver les références de librairie cryptographique en grand nombre et finir la création d'un questionnaire à cause de limitations du site sélectionné comme hôte.

L'évaluation des résultats à l'aide de Limesurvey[6] était prévue, mais trop de problèmes ce sont présentés pour que le plan est pu être mené à terme. Cette tâche sera accomplie cet été.

Limitations

Les limitations de l'analyse s'illustrent principalement par la quantité de faux positif possible en résultat de l'analyse.

Conclusion

En conclusion, l'objectif d'améliorer la détection des a été accompli. À l'aide de Visual Code Grepper (VCG) il est possible d'obtenir un taux de succès de 30% ou 9 sur les 30 CVE analysés durant la recherche précédente [1]. Ceci est une augmentation de 26.66%. Il serait très pratique de faire l'analyse des messages d'erreur générés à partir de la méthodologie décrite plutôt dans ce document.
L'évaluation de la qualité de ces détections aurait été à l'aide d'une enquête sur une population de programmeur.

Bibliographie

[1] J. Bolduc and J. Lafrenière Nickopoulos, *Analyse des vulnérabilités cryptographiques catégorisées par des logiciels d'analyse*, Dec, 19, 2022 [Online]. Available: https://github.com/RaphaelKhoury/CryptographicVulnerabilities/blob/main/RapportFinal.docx.

[2] nccgroup. VCG. https://github.com/nccgroup/VCG (accessed Mar, 3, 2023).

[3] Blochberger, M., Petersen, T., & Federrath, H. (2019). Mitigating cryptographic mistakes by design. *Mensch und Computer 2019-Workshopband*.

[4] A. Odedele, Tableau des vulnérabilités, Mar, 3, 2023 [online]. Available: https://github.com/2longAGO/Projet_files/blob/main/proj_synth.xlsx

[5] NIST, National Vulnerability Database. [online]. Available: https://nvd.nist.gov

[6] A. Odedele, Questionnaire tentatif, Avril, 17, 2023 [online]. Available: https://github.com/2longAGO/Projet_files/blob/main/Questionnaires%20-%20Test.pdf