Graham Dungan
March 26, 2025
UIN: 332001764

CSCE 413: Software Security
Class 27: SAST

Important Files

The following is a list of important files alongside this report and their purpose;

- `vuln.c` - A simple C program vulnerable to buffer overflow
- `codeql-results.csv` - A resultant csv from running a local CodeQL scan on `vuln.c`
- `codeql.yml` - A GitHub Actions workflows CodeQL file.

Local CodeQL Docker

As included with this file, we will be running a CodeQL analysis on the file `vuln.c`. `vuln.c` has been used on previous assignments as an example of a classical buffer overflow due to the unchecked use of `strcpy`.

Installing the CodeQL Image

For this assignment, I have used the Microsoft CodeQL Container for the docker image. This was installed with the command,

```
docker pull mcr.microsoft.com/cstsectools/codeql-container:latest
```

Creating a CodeQL Database

In order to run CodeQL I first created a CodeQL database. In creating a database for `vuln.c`, I have transformed my code into a detailed representation that better describes the program's operations, logic, and syntax. This intermediate representation allows us to perform multiple operations/queries on the target file instead of processing the code every time the script is inspected. The command is as follows;

```
sudo docker run --rm --entrypoint /bin/sh \
-v "$(pwd):/opt/src" \
mcr.microsoft.com/cstsectools/codeql-container:latest \
-c "codeql database create --language=cpp --source-root=/opt/src
    --command='gcc /opt/src/vuln.c -o /opt/results/vuln.bin' /opt/results/vuln-db"
```

For a brief breakdown of this command;

- `sudo docker run --rm --entrypoint /bin/sh \` runs docker with an overridden entry point, as I was having permission issues with `startup.py`
- `-v "$(pwd):/opt/src` mounts the current working directory to `/opt/src`
- `mcr.microsoft.com/cstsectools/codeql-container:latest` is the docker image we will be using
- `-c "codeql database create --language=cpp --source-root=/opt/src...` calls codeql to create a database, specifying the language, root location, and the file `vuln.c`

Running CodeQL

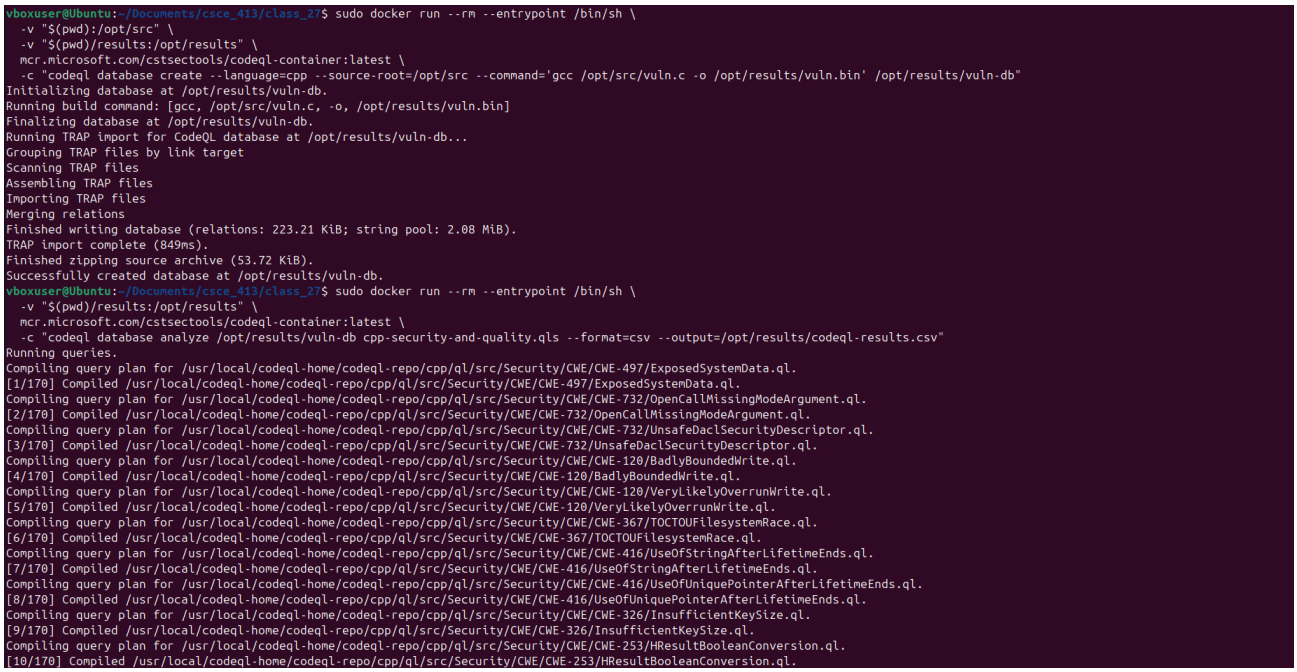
Now that a database for the file has been created, we can run an analysis on the file using the following command;

```
sudo docker run --rm --entrypoint /bin/sh \
-v "$(pwd)/results:/opt/results" \
mcr.microsoft.com/cstsectools/codeql-container:latest \
-c "codeql database analyze /opt/results/vuln-db cpp-security-and-quality.qls
    --format=csv --output=/opt/results/codeql-results.csv"
```

Similarly, a brief breakdown of this command (barring the previously discussed lines);

- `-c "codeql database analyze /opt/results/vuln-db cpp-security-and-quality.qls` uses codeql to analyze the previously constructed database, specifying the CodeQL security suite `cpp-security-and-quality.qls`.
- `--format=csv --output=/opt/results/codeql-results.csv` specifies the output to be a csv in the previously mounted directory.

An image of executing these commands follows,



Results

As included with this report, `codeql-results.csv` contains details of the analysis on `vuln.c`, which will be displayed here;

```
"Unbounded write","Buffer write operations that do not control the length of
data written may overflow.","error","This ’call to strcpy’ with input from
[[""a command-line argument""—""relative:///vuln.c:10:26:10:29""]]
may overflow the destination.","/vuln.c","6","2","6","7"
```

As expected, CodeQL has found a potential buffer overflow due to the use of `strcpy`. What follows is a screenshot of the output of the analysis as well as `codeql-results.csv`

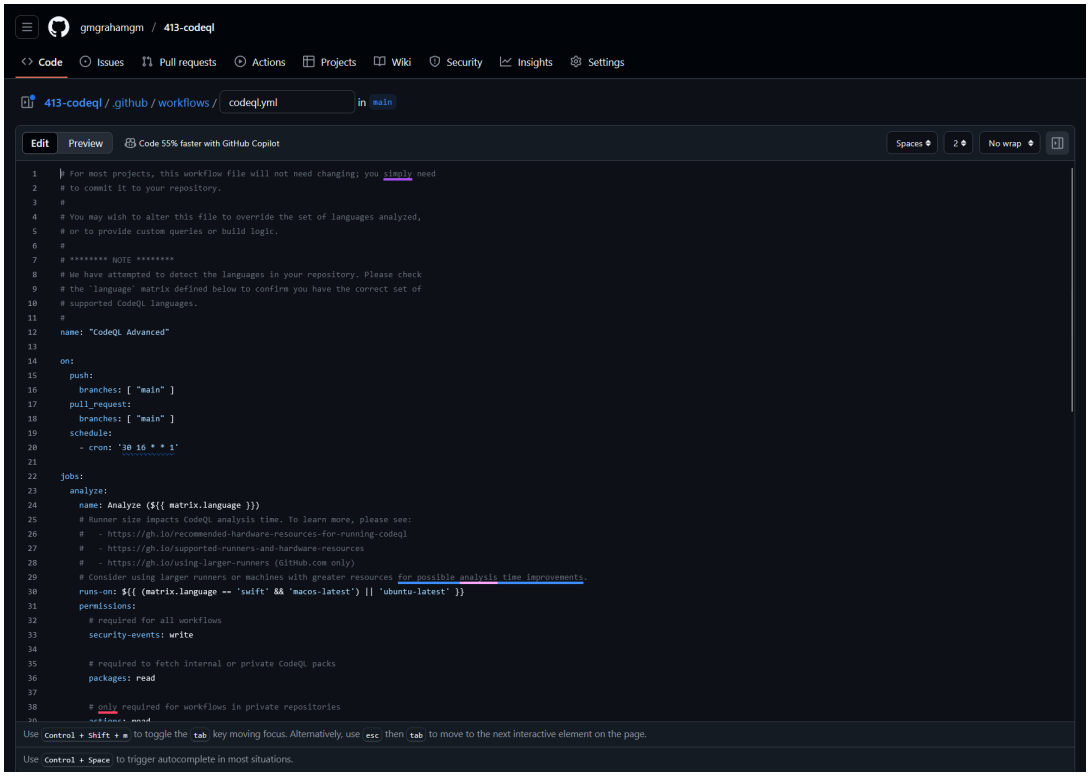
```
Starting evaluation of codeql/cpp-queries/jsf/4.24 Control Flow Structures/AV Rule 196.ql.
[167/170 eval 77ms] Evaluation done; writing results to codeql/cpp-queries/jsf/4.21 Operators/AV Rule 166.bqrs.
Starting evaluation of codeql/cpp-queries/jsf/4.24 Control Flow Structures/AV Rule 197.ql.
Starting evaluation of codeql/cpp-queries/jsf/4.24 Control Flow Structures/AV Rule 201.ql.
[168/170 eval 43ms] Evaluation done; writing results to codeql/cpp-queries/jsf/4.24 Control Flow Structures/AV Rule 196.bqrs.
[169/170 eval 23ms] Evaluation done; writing results to codeql/cpp-queries/jsf/4.24 Control Flow Structures/AV Rule 197.bqrs.
[170/170 eval 407ms] Evaluation done; writing results to codeql/cpp-queries/jsf/4.24 Control Flow Structures/AV Rule 201.bqrs.
Shutting down query evaluator.
Interpreting results.
Analysis produced the following diagnostic data:

|          Diagnostic          | Summary  |
+------------------------------+----------+
| Successfully extracted files | 1 result |

Analysis produced the following metric data:

|                        Metric                         | Value |
+-------------------------------------------------------+-------+
| Total lines of user written C/C++ code in the database |    12 |
```

```
vboxuser@Ubuntu:~/Documents/csce_413/class_27$ ls results/
codeql-results.csv  vuln.bin  vuln-db
vboxuser@Ubuntu:~/Documents/csce_413/class_27$ cat results/codeql-results.csv
"Unbounded write","Buffer write operations that do not control the length of data written may overflow.","error","This 'call to strcpy' with input from [[""a command-line argument""|""
relative:///vuln.c:10:26:10:29""]] may overflow the destination.","/vuln.c","6","2","6","7"
```

Remote CodeQL GitHub Action

Setting Up GitHub Actions

For this section of the assignment, I will create a simple public GitHub repository named `413-codeql`. GitHub offers the ability to create a CodeQL GitHub Actions workflow file via a template,



This template generally covers all the necessities in terms of code scanning. However, I have made modifications to specify the language to be C/Cpp so that the scanner can operate in all branches, remove the scheduling with crontab, and specify build instructions. For the sake of brevity, the `codeql.yml` GitHub actions file have been included with this report.

Uploading Vulnerable Files

Now that the repository has been created with code scanning, all that is needed is to upload the vulnerable file `vuln.c`. Before `vuln.c` is uploaded, however, I will make a brief modification to make it more vulnerable, such that CodeQL can easily detect it. After some extensive testing, `vuln.c`'s buffer overflow (caused by `strcpy`) is not immediately detectable by CodeQL on GitHub as the data passed into `strcpy` is never used in a security-sensitive manner. By adding the statement `printf(args[1]);`, we add yet another vulnerability to the script.

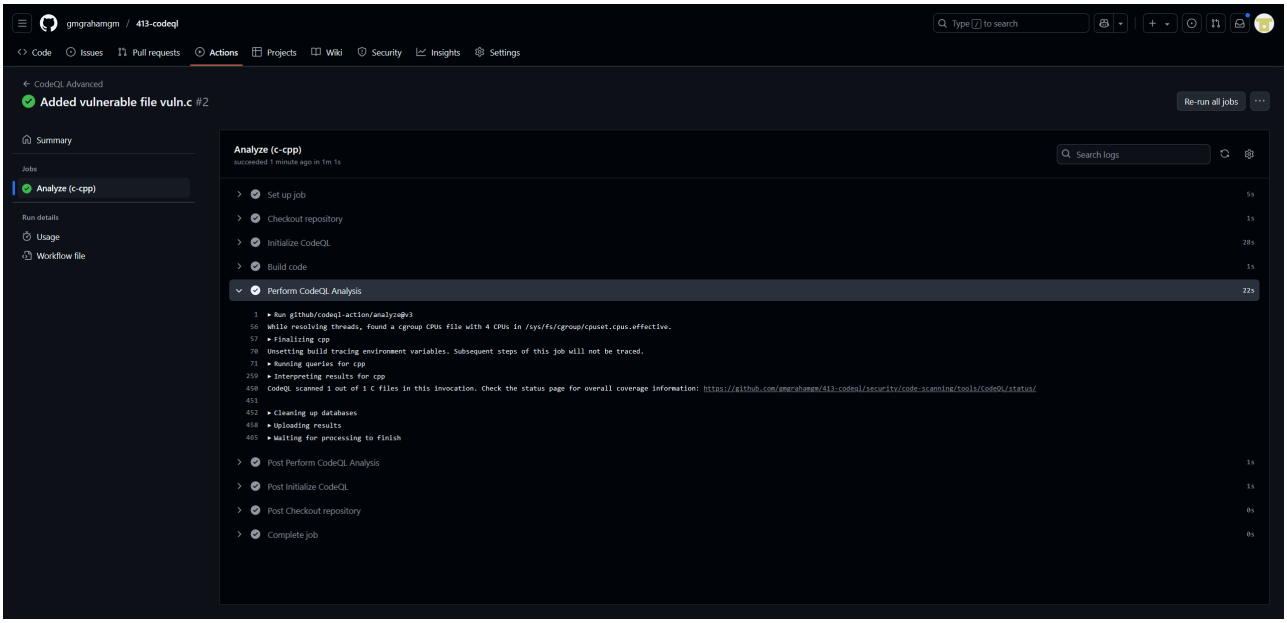
```c
#include <stdio.h>
#include <string.h>

int copy_string(char* str) {
        char buf[128];
        strcpy(buf, str);
        return 1;
}

int main(int argc, char *argv[]) {
        copy_string(argv[1]);
        printf("Successfuly copied the string ");
        printf(argv[1]); // New line
        return 0;
}
```

Now that `vuln.c` has been modified to be more easily detectable by CodeQL, we can push it to the repository,

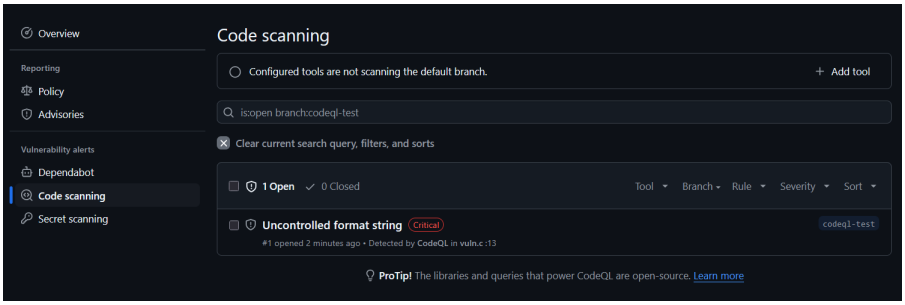
```
vboxuser@Ubuntu:~/Documents$ git clone git@github.com:gmgrahamgm/413-codeql.git
Cloning into '413-codeql'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (8/8), done.
vboxuser@Ubuntu:~/Documents$ cd 413-codeql/
vboxuser@Ubuntu:~/Documents/413-codeql$ git checkout codeql-test
branch 'codeql-test' set up to track 'origin/codeql-test'.
Switched to a new branch 'codeql-test'
vboxuser@Ubuntu:~/Documents/413-codeql$ cp ../csce_413/class_18/vuln.c ./
vboxuser@Ubuntu:~/Documents/413-codeql$ nano vuln.c
vboxuser@Ubuntu:~/Documents/413-codeql$ git add vuln.c
vboxuser@Ubuntu:~/Documents/413-codeql$ git status
On branch codeql-test
Your branch is up to date with 'origin/codeql-test'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   vuln.c

vboxuser@Ubuntu:~/Documents/413-codeql$ git commit -m "Added vulnerable file vuln.c"

[codeql-test 89c6316] Added vulnerable file vuln.c
 1 file changed, 15 insertions(+)
 create mode 100644 vuln.c
vboxuser@Ubuntu:~/Documents/413-codeql$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 6 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 479 bytes | 479.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:gmgrahamgm/413-codeql.git
   42dd865..89c6316  codeql-test -> codeql-test
vboxuser@Ubuntu:~/Documents/413-codeql$
```

CodeQL Analysis

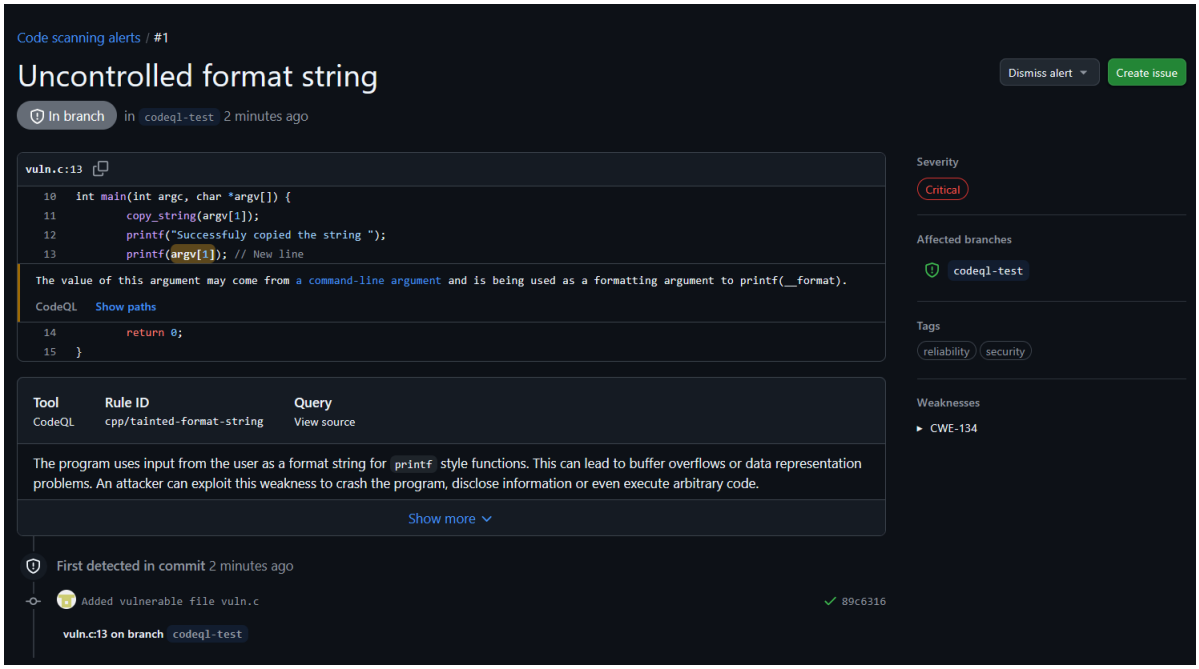
Once the code has been pushed to the repository, it is now possible to view the GitHub action scan in real-time.



Once the code analysis has finished, we can view the outcome of the scan in the security tab.



In viewing this security notification, we can find the type of vulnerability, the file that caused it, and why the vulnerability is considered dangerous.



As expected, the offending file was the `vuln.c` due to the recently added `printf()` call. We have successfully set up CodeQL in GitHub Actions to detect vulnerable code.