|  |  |
| --- | --- |
| Lab report 3  OpenSSF Scorecard | Abstract  This lab covers basics of how the “OpenSSF Scorecard” works using coursework which provides a certificate upon completion, trial, and error, and the benefits it provides for our respective GitHub.  Nash Morrison  Computer Security: CEN 3078 |

**Task 1: OpenSSF Scorecard Course [Certification]**

Our first task involves completing the “Securing Projects with OpenSSF Scorecard” (LFEL1006) course:



This course teaches the user how to properly set up your scorecard within your Github repository. The following pathway will lead to connecting your specific scorecard in your Github settings:

This creates a [.yaml] file that will scan our repository for security risks that need correcting. The system works by categorizing the threats into three main categories, [High], [Medium], and [Low]. The ones we will be focusing on today are two [High Level] threats. To better understand how these systems work, I also corrected an extra [High Level] and two [Medium Level] threats so that I will have a better understanding of future repositories.

**Task 2: Two Badges**

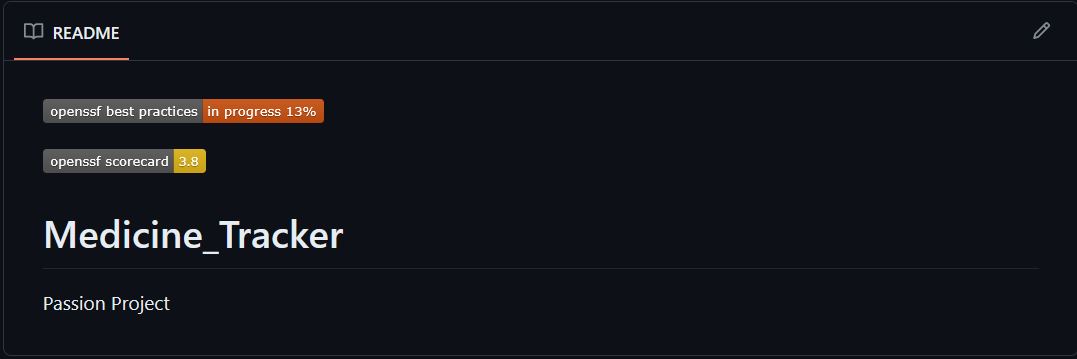
***Repository:*** [***https://github.com/NWMorrison/Medicine\_Tracker***](https://github.com/NWMorrison/Medicine_Tracker)

For this lab we are dealing with two separate badges that we must show in our repository which is a passion project I have been working on titled, “Medicine\_Tracker.” Once the [.yaml] file is committed, we must add two lines of code within our [README.md] file so that we can properly show our scorecards.

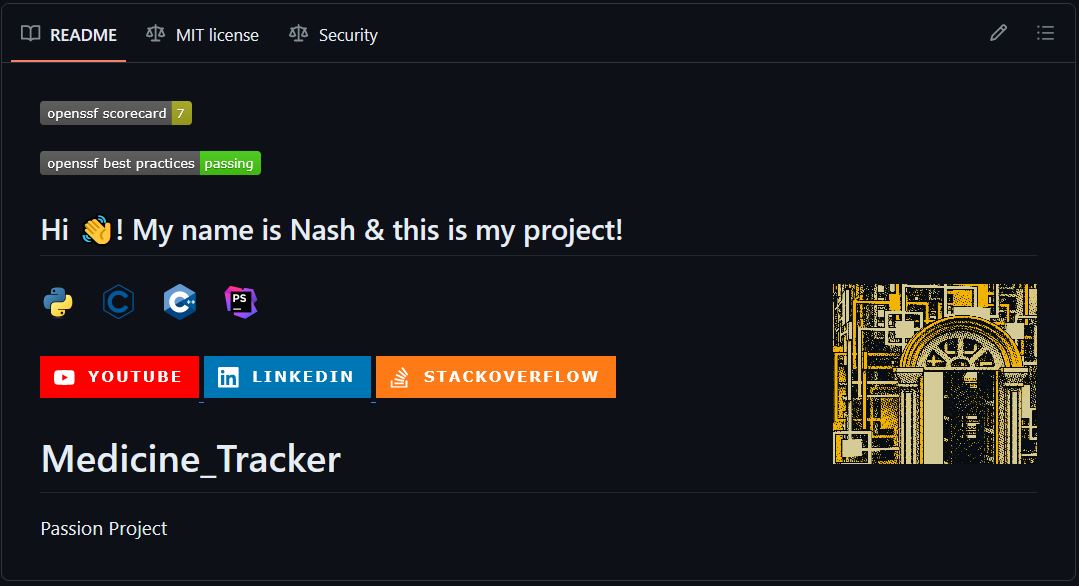
Our first badge, “OpenSSF Scorecard” which coincides with our [.yaml] file uses the following code snippet:

Our second badge, “OpenSSF Best Practices” uses the following code snippet:

This provided my repository with the following two badges with their initial scores:



As shown above, my “OpenSSF Scorecard” is a score out of a possible ten points, so improvements on our security is priority number one if we want to improve the score. My “Best Practices” score has three different rankings, a [Passing], [Silver], and [Gold]. For this lab there must be at least a “Passing” rank and an improvement upon our 3.8 score. After fixing three [High] and two [Medium] alerts, and creating a more welcoming repository, this is my result for my scores:

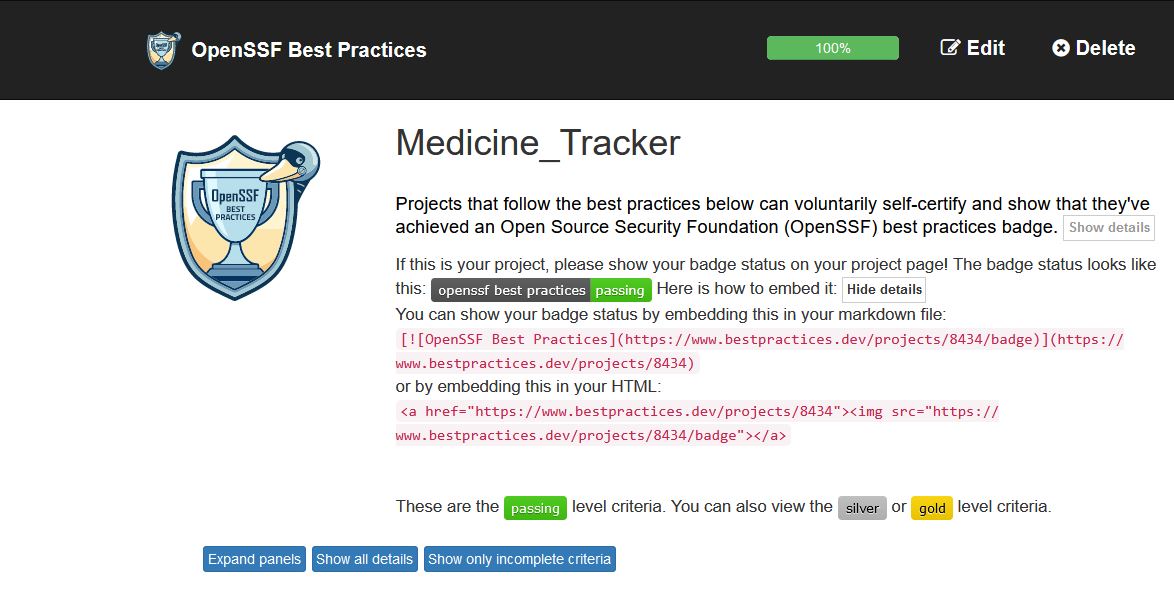


I was able to almost double my score with only minor modifications to my code.

This lab also got me into learning how to install Ruby, Jekyll, and bundle so that I can host my own website through Github. It is still a **work in progress**, but figured I would provide the link to show that I went beyond the expectations for this lab.

Website: <https://nwmorrison.github.io/>

On the “OpenSSF Best Practices” website, it has a checklist of six different areas that covers a wide range of topics including [Basics], [Change Control], [Reporting], [Quality], [Security], [Analysis]. To achieve a passing score for this badge, you must go through and meet or check off every requirement listed in each category. They range from how many individuals have access to your repository, can users provide feedback for vulnerabilities, having a license, having a security policy, a version number, and a wide variety of other criteria that makes sure that your code is secure from outside influences that could damage or otherwise harm your software.



I also wanted to provide confirmation that my repository does have its passing rank. I noticed that sometimes it would show up with my initial progression score. I did research and found that it is a cache issue. Sometimes it shows up properly, other times it does not.

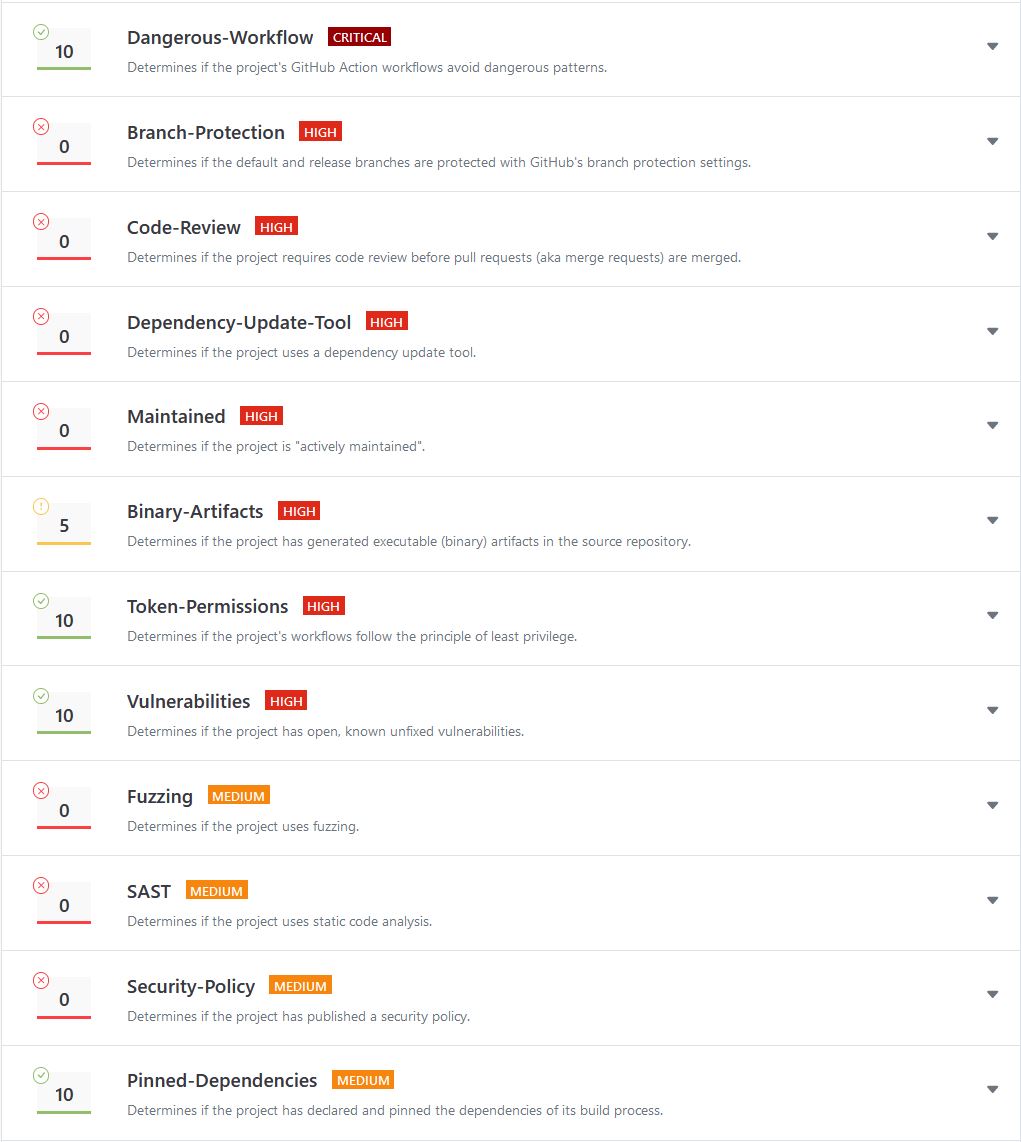
Source: <https://github.com/coreinfrastructure/best-practices-badge/issues/2070>

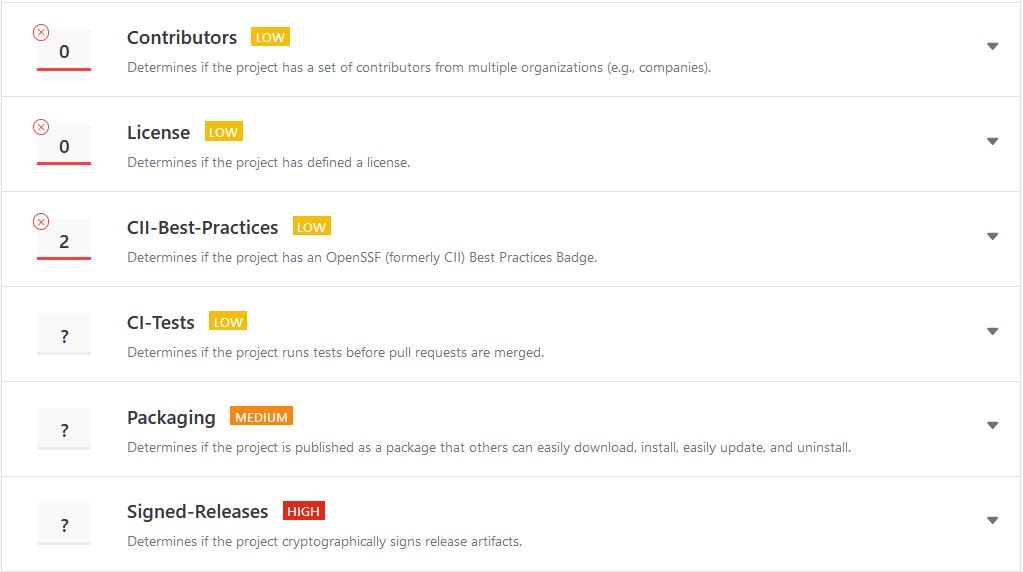
**Task 3: Improving On Scorecard**

As previously stated above, I decided to fix more than the given requirements. Initially this was our score:



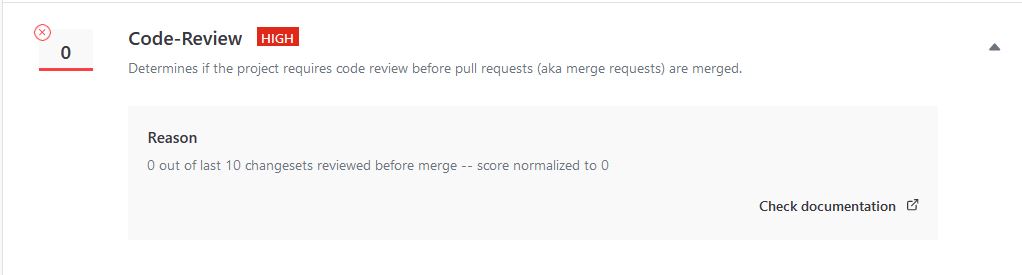
This score, found on the “OpenSSF Scorecard Report” website is where we added our [.yaml] file in our Github repository as previously discussed. Here we can see every security threat to our code that needs analysis:



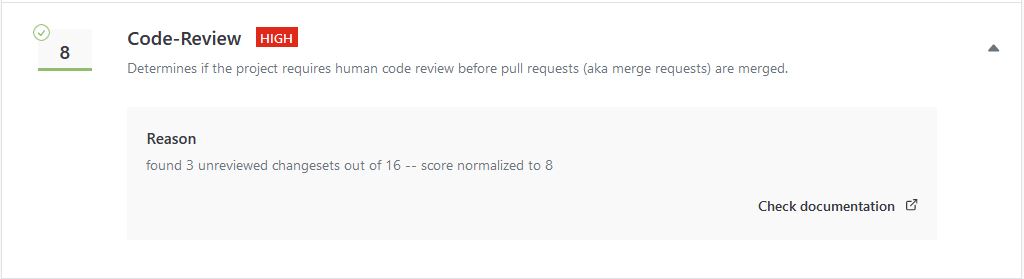


The indicates the two [High Level] threats that I focused on, while the gggggg indicates [High Level] & [Medium Level] threats that I completed as bonuses.

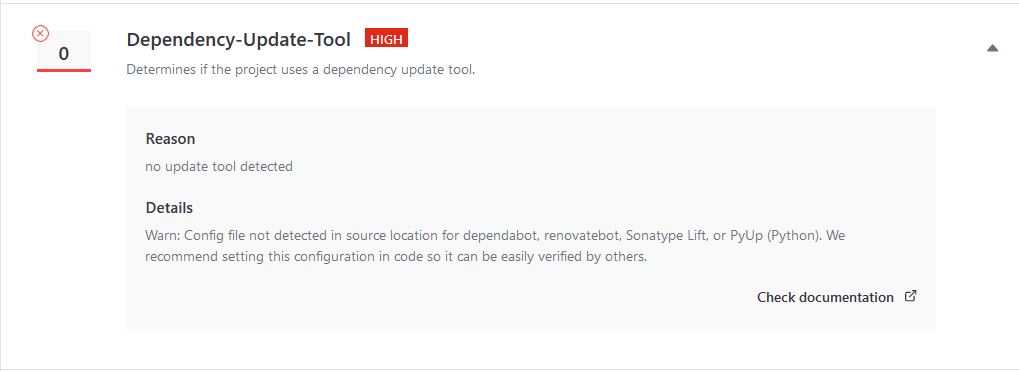
**[High Level] Threat One: (Code Review)**



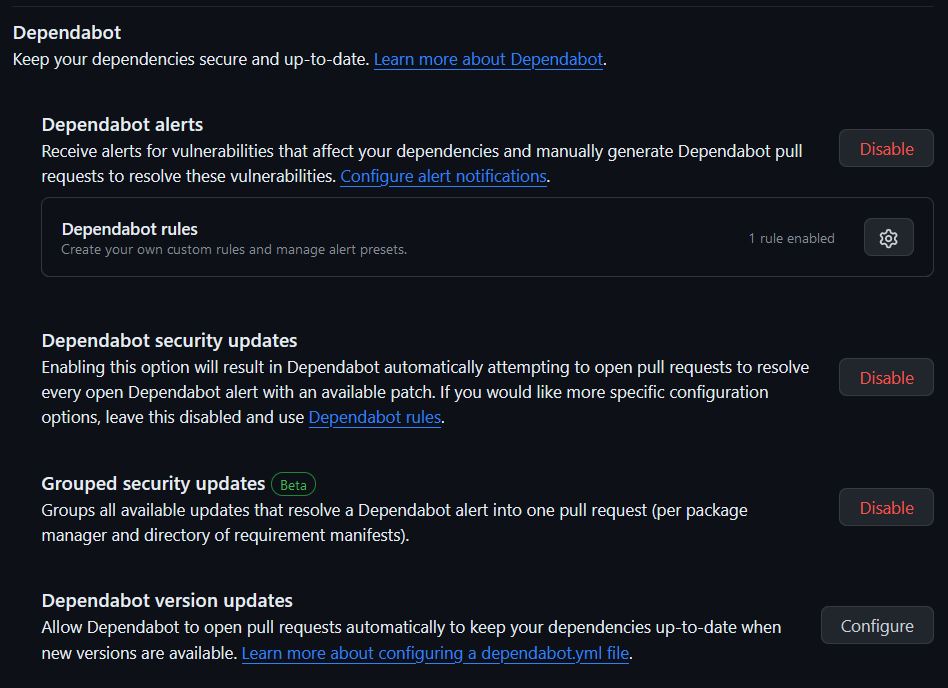
This [High Level] security threat is straight forward. The idea is that every time an individual pushes new source code to the repository, a peer-review from a different individual with confirmation of that code before anything new can merge. There must also be consistency with reviewed “changesets” over the course of the project’s lifespan. To fix this threat, I created a separate account so that I could pretend to be a different person so that the reviewed code has the same feeling as a normal coding environment.



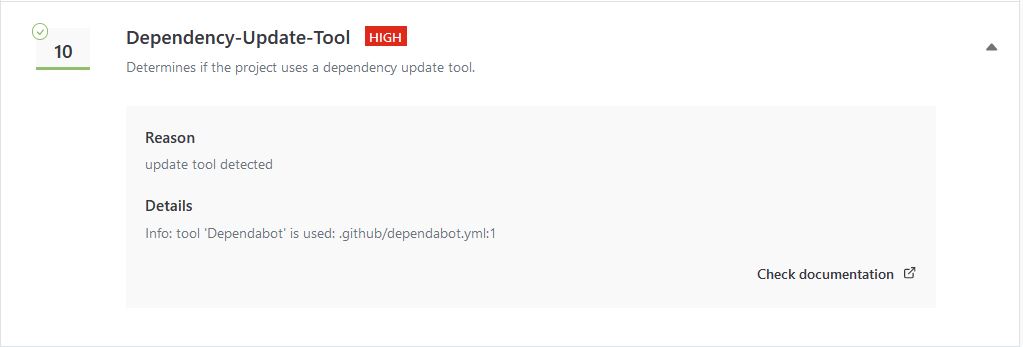
**[High Level] Threat Two (Dependency Update Tool)**



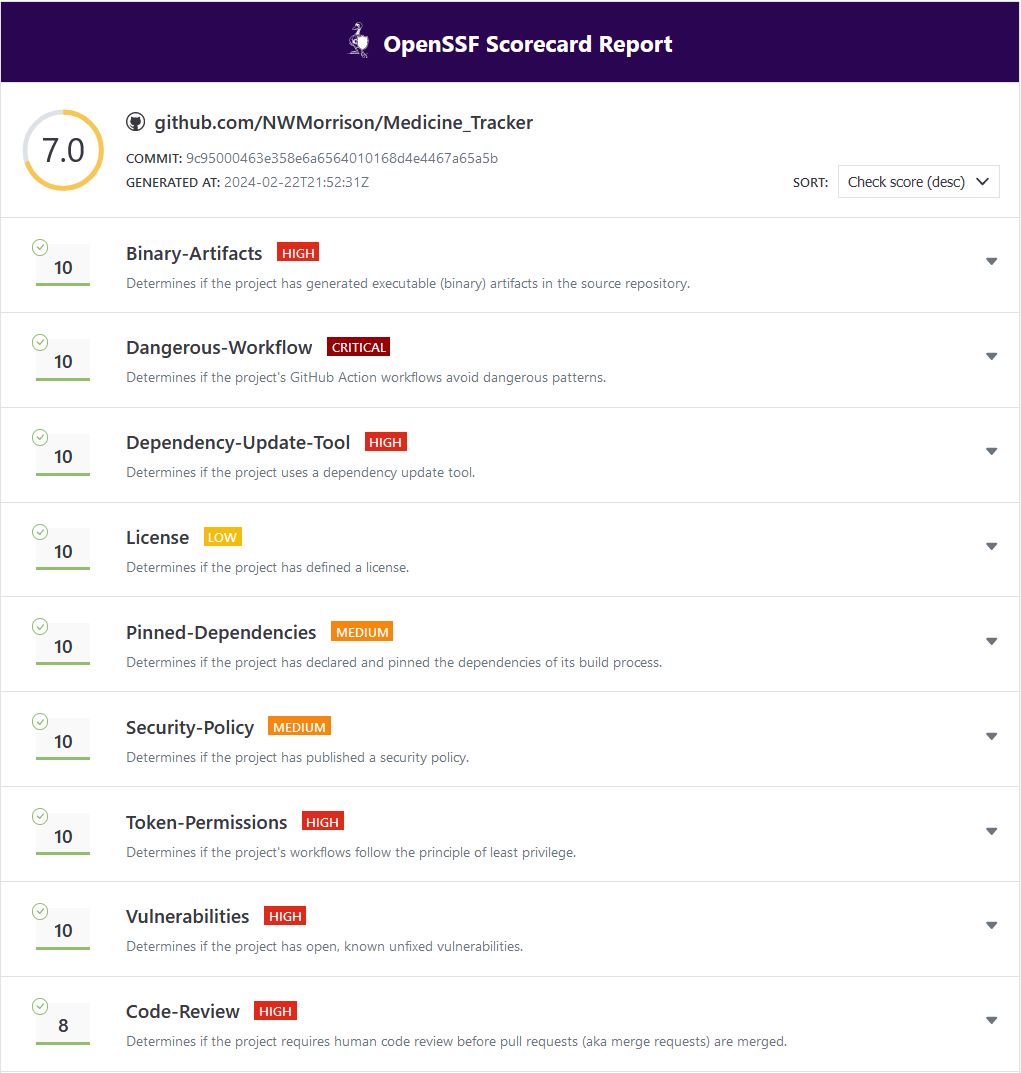
Our second [High Level] security threat checks to see if we have a dependency update tool enabled within our Github repository. This tool allows for the automation of dependency processes which will take care of any security problems down the road. We fix this security threat by turning on Dependabot in our Github repository:

****

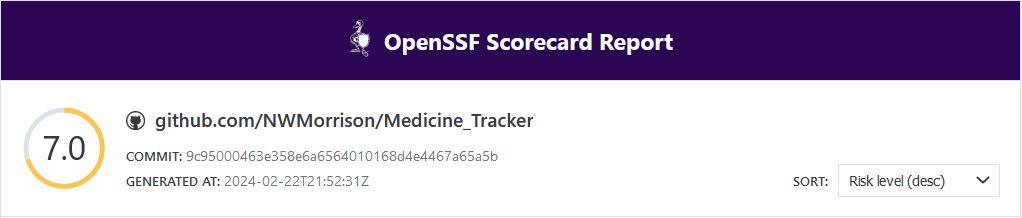
Turning on Dependabot will establish and create another [.yaml] file . That merges into our workflow repository. Once turned on, we have an increase in our score:



To show proof that I did in fact work on [High Level], [Medium Level], and [Low Level] threats outside of the parameters of this assignment, I provided the following picture:



**Improvement On Score:**



After all our corrections, our score improved by almost half. Scoring a 7/10, there is room for improvement, but this is a step in the right direction to better secure our software.

**Conclusion:**

In conclusion, this lab taught me valuable lessons on how Github functions and operates. Learning how to provide these security badges to our projects will provide future employers with a chance to see that you are thinking about security concerns while working on your projects. This also taught me how to effectively use Github Push, Pull. And Accepting Code Improvements to merge into the main branch. Genuinely stumped as to how that entire process worked, but this lab was worth its weight in gold with the knowledge gained.

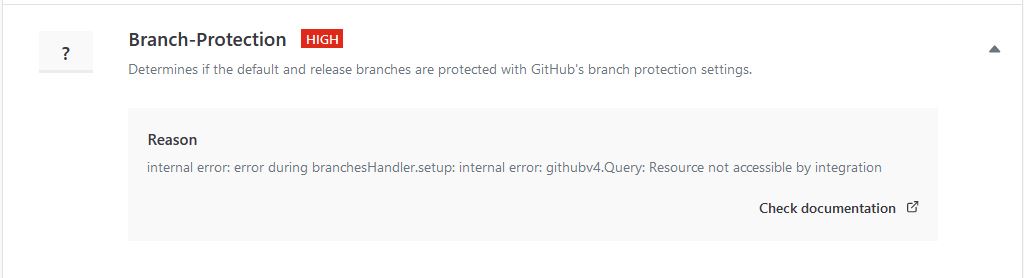
-Nash Morrison

**Addendum Below**

**Addendum (Extra)**

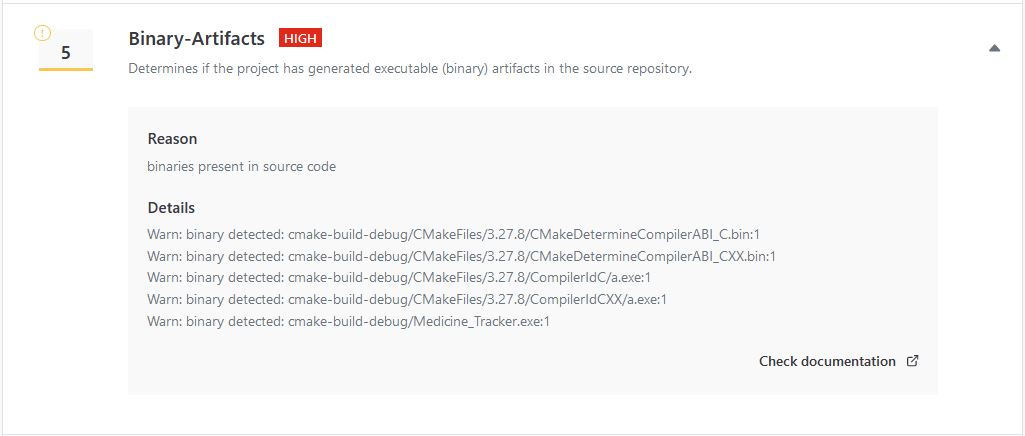
**[High Level] – Branch Protection**

A screenshot of a computer

Description automatically generated

I was going to initially use this, however after researching, I found that Github causes a bug where once you establish this protection, there is confusion with the pull request when transmitted data pushes back over to the Scorecard website. There is a solution which requires manipulation of code in a specific file, and as though that was a little too much. You can still establish branch protection; it will just provide the scorecard with an “?” score.

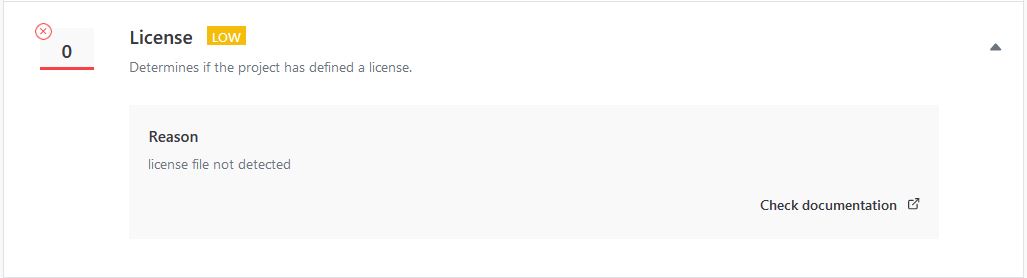
**[High Level] – Binary Artifacts**





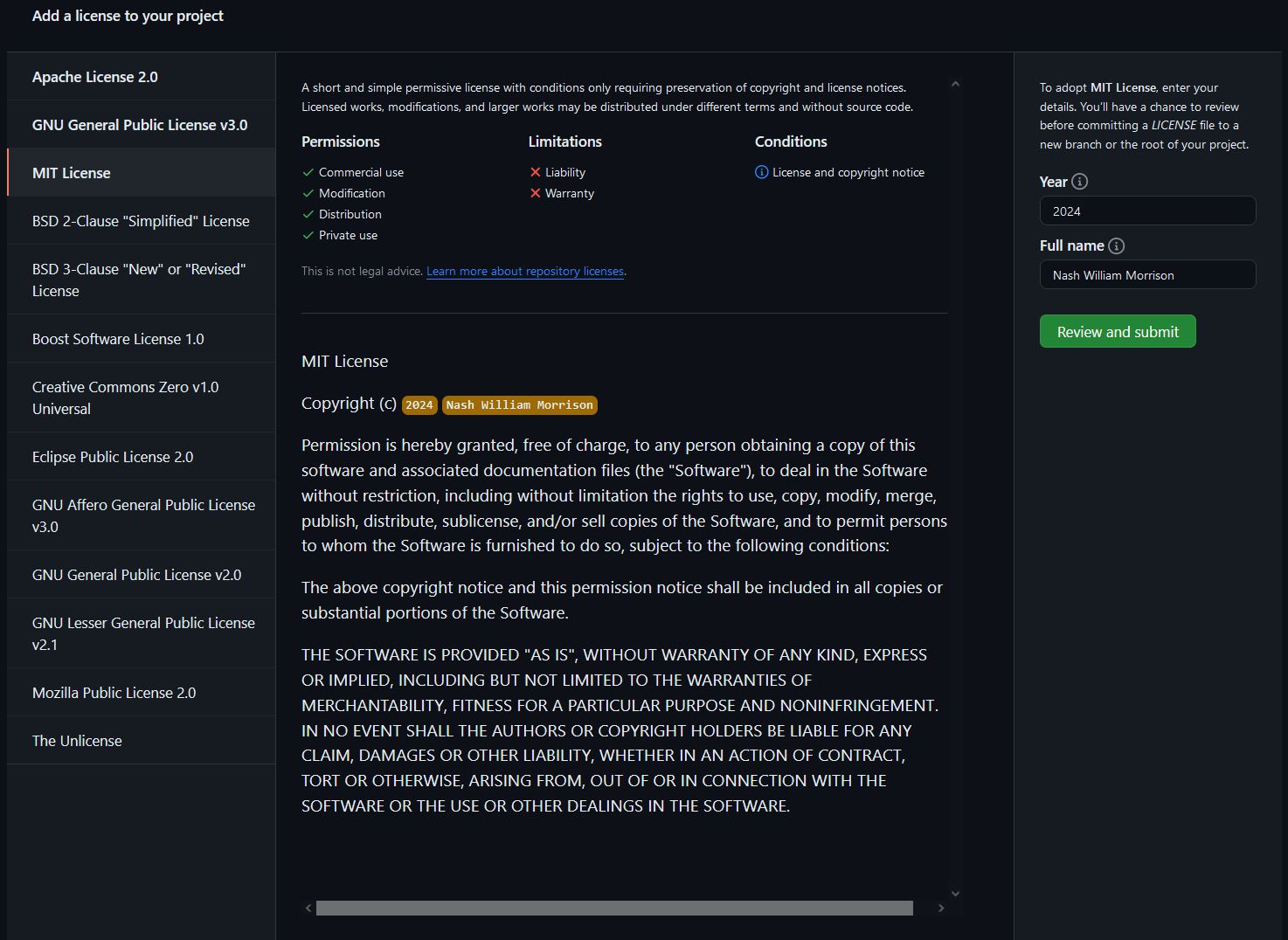
How this works is that this requires you to make sure that you do not have any executable binary files. The possibility of altering files with binary code by an outside source can cause major security issues. You can fix this by simply getting rid of specific [.Cmake] or [.exe] files and it will correct the concern.

**[Low Level] – License**



To fix this, we must create a [LICENSE] within our repository:

This will allow you to select which license you would like to provide with your software.



I decided to go with the MIT License. On the righthand side of the page we must type in the year and our full name, review and submit, and then commit the LICENSE to our repository to receive our final score:

