Assignment 6 221214 Matteo Franzil

**Usage of components with known security vulnerabilities**

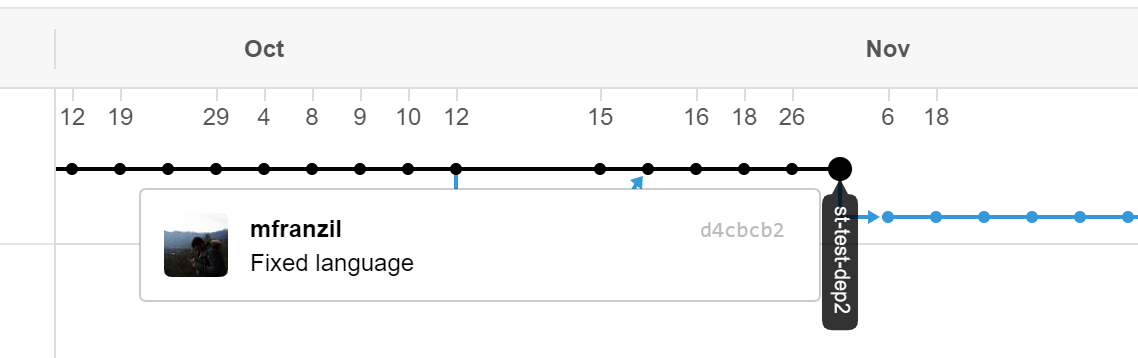
**Part 1 – Selection of open-source project**

Firstly, I chose a suitable open-source project for this assignment. After having examined various public and well-known public projects, I instead chose to focus on a project I did by myself (with some third party collaboration). This project is a Python webapp functioning as a Telegram bot. The bot itself is tasked with sending daily messages in my student dormitory (NEST)’s kitchen chat, informing people on their daily cleaning turns. This bot has been refined, refactored and changed over time implementing several functionalities. Now, it sits under the InnovationTeamNEST umbrella repo:

<https://github.com/InnovationTeamNest/cooking-club-nest-bot>

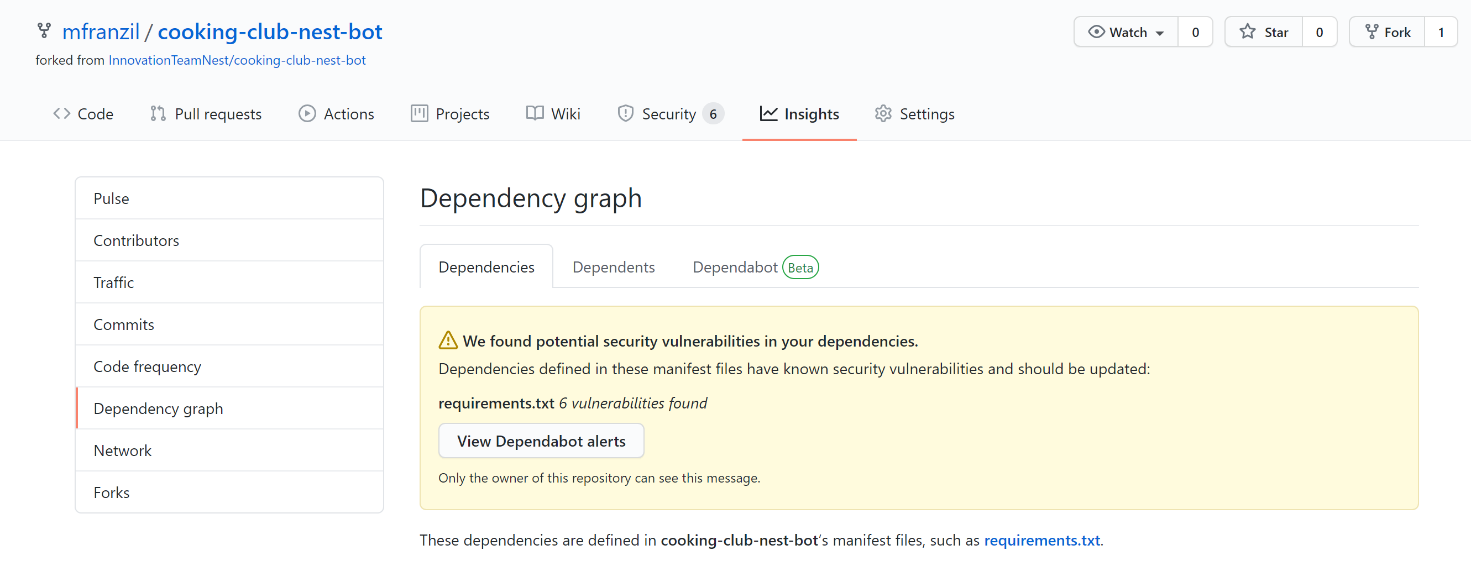
Since 2018, all of the changes to the bot were made by me, including library updates. In fact, Dependabot has been active for a while on this repo, prompting me with emails in case vulnerabilities were discovered.

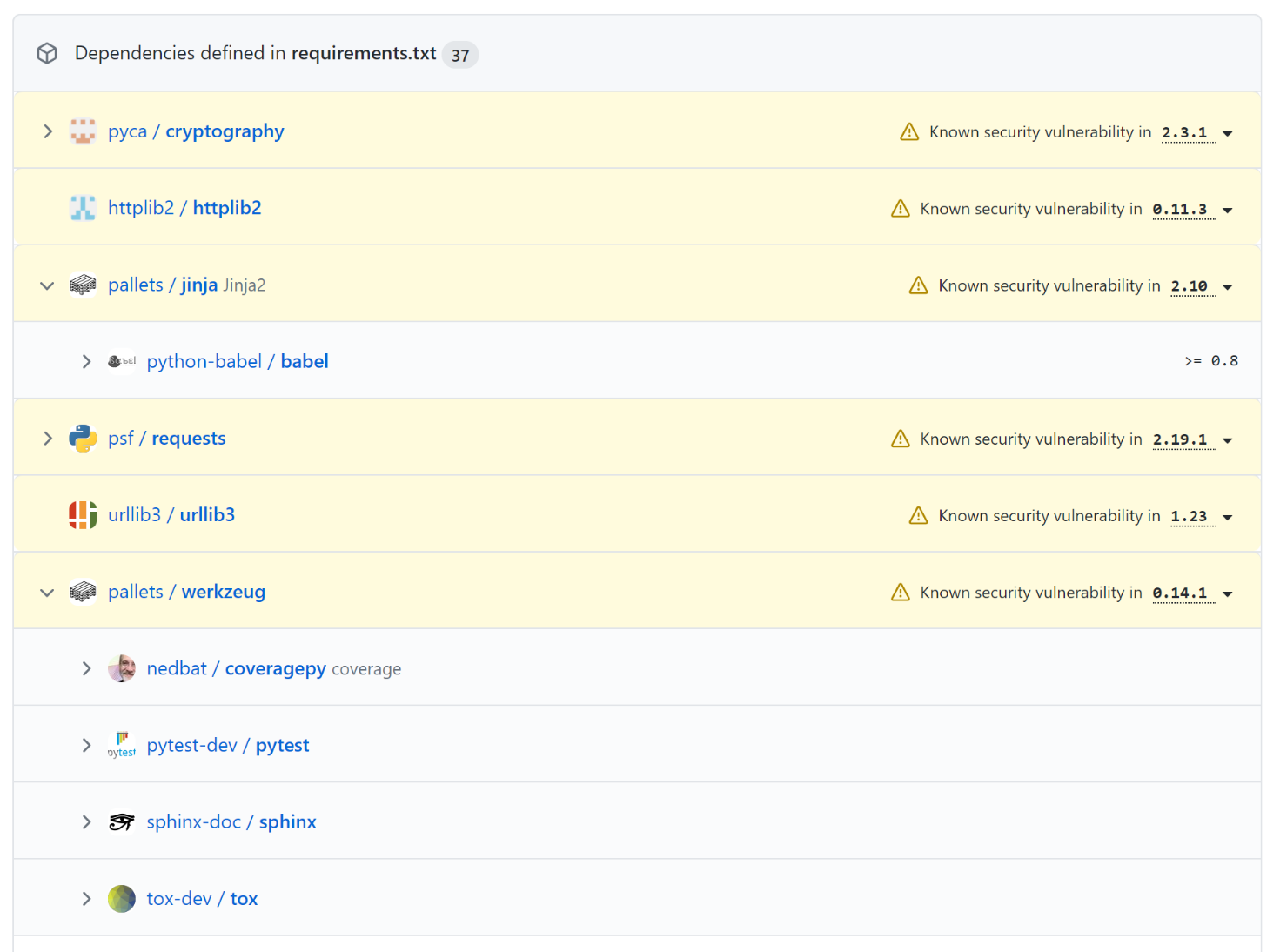
**Part 2 – Project fork**

As the current snapshot, the project offered no known vulnerabilities. I therefore chose to backtrack to commit d4cbcb2, one of the first featuring the requirements.txt file (needed by the dependency tracker) but old enough to warrant a plethora of vulnerable libraries.

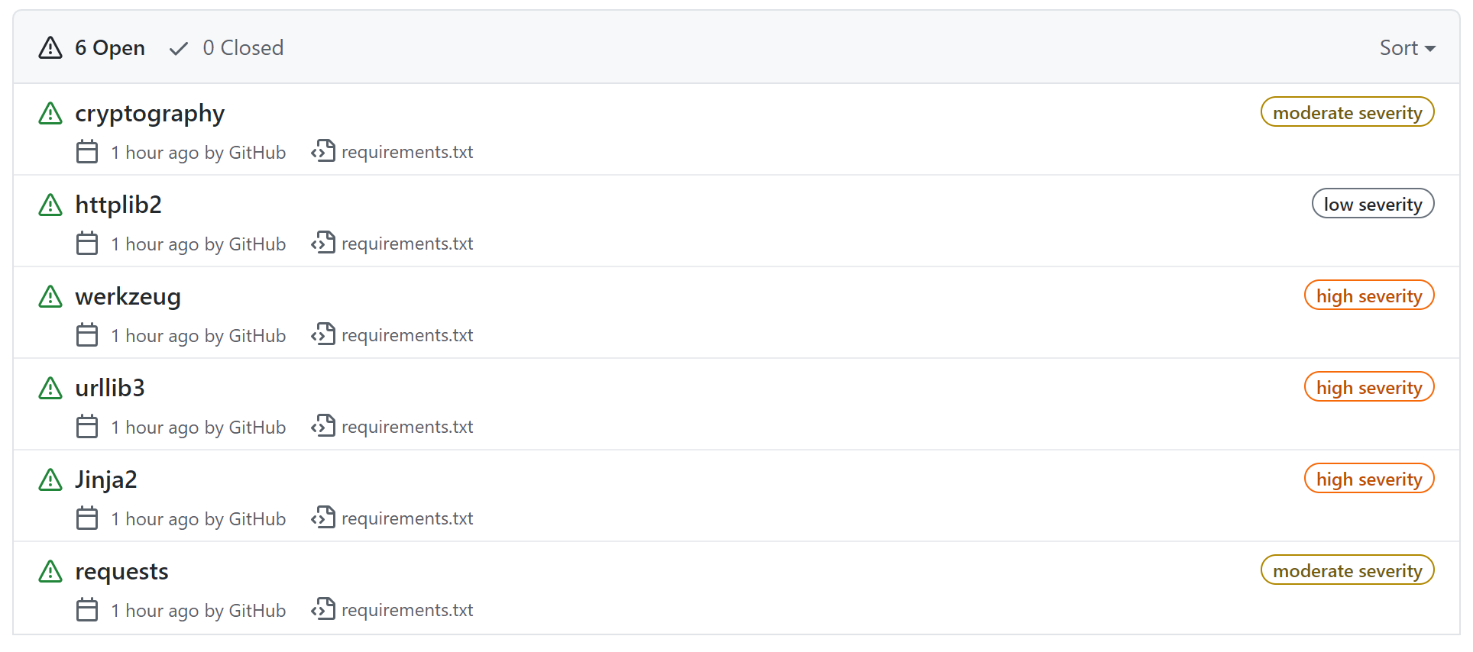
This commit was forked into its own repo (available under mfranzil/cooking-club-nest-bot). in this new repo, the dependency tracker and Dependabot were enabled.

**Part 3 – Dependency graph**

The next step was to visit the Dependency graph under the Insights tab.

As the dependency graph would take pages to be fully visualized in this report, the following is a screenshot highlighting only the vulnerable packages found by Dependabot.

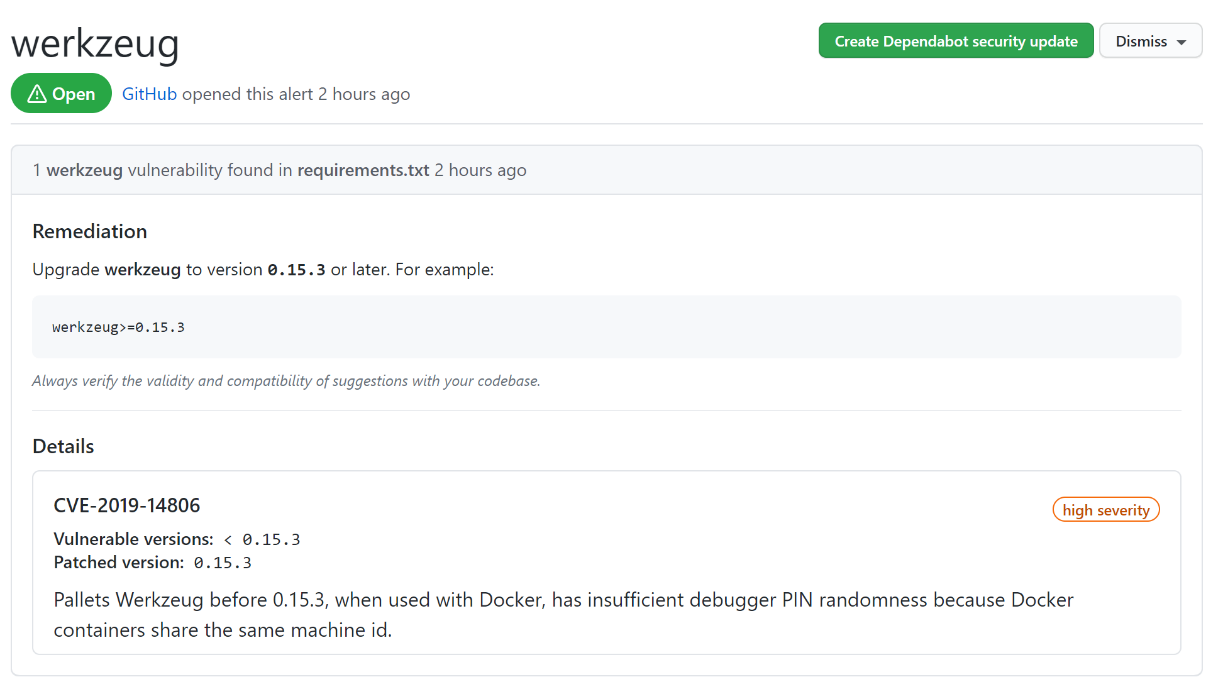
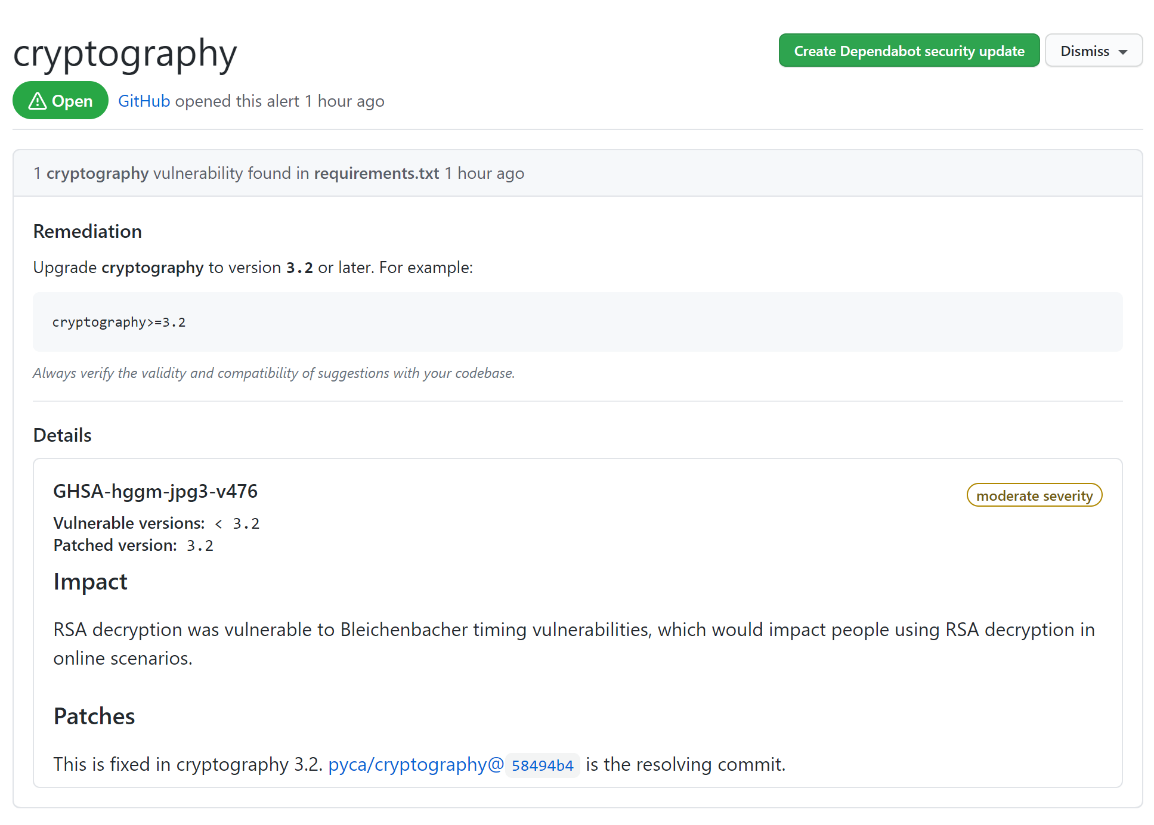
**Part 4 – Dependabot**

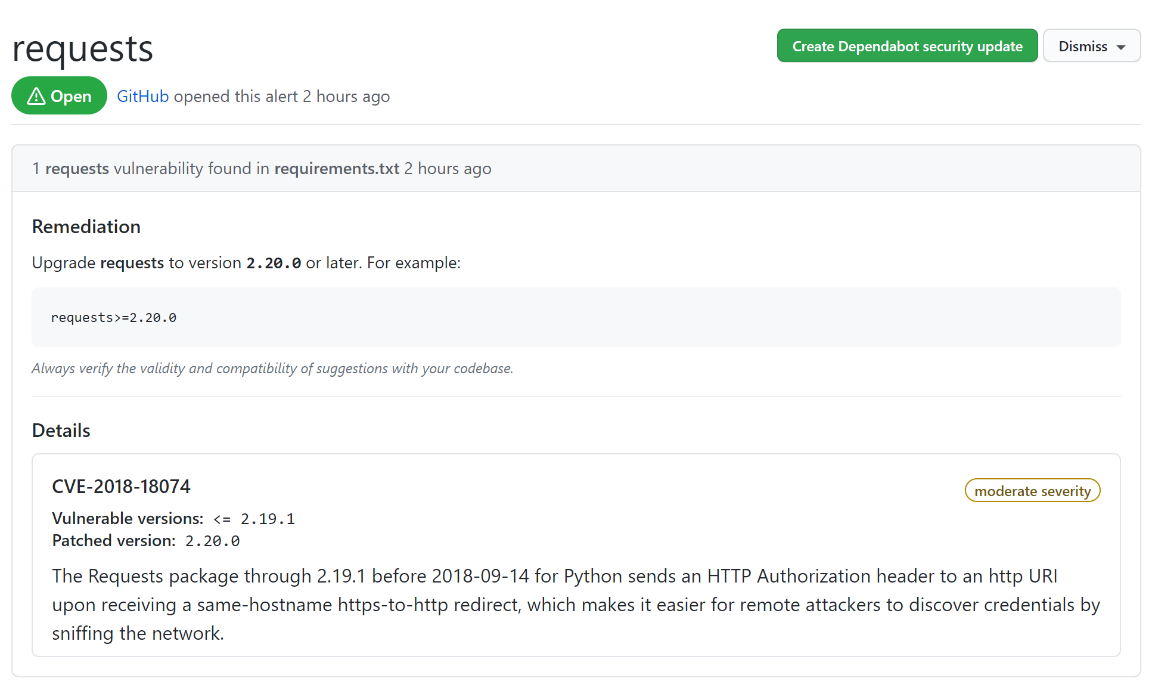
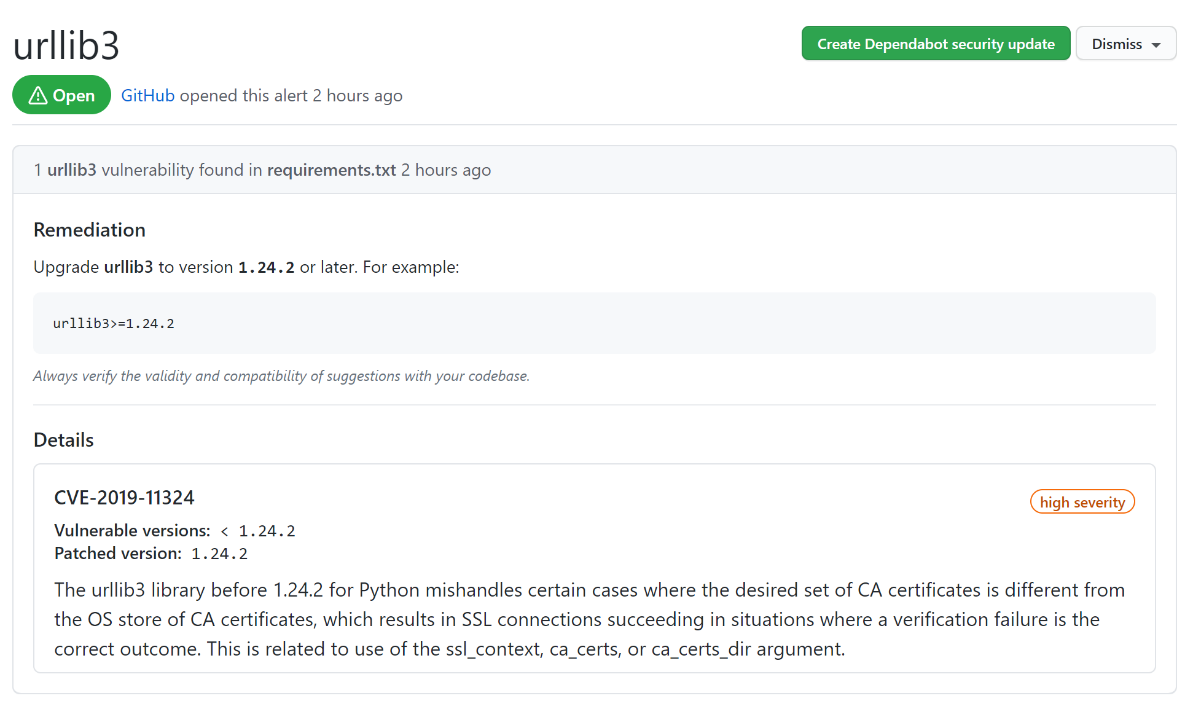
Visiting the Security > Dependabot tab yields the following view, once alerts are enabled:

Clicking a vulnerability opens a detailed page view, highlighting the issues, the suggested remedy, the details of the security breach and more.

Usually, the implementation of such security fixes is as simple as changing a line in the requirements file (such as requirements.txt in Python or pom.xml in Maven-based Java projects). Indeed, all vulnerabilities found in the project were very minor, usually requiring the aforementioned “line fix”. Usually, library makers – although sometimes lazy – separate logically their updates in major, minor, and patch versions. Patch versions usually feature the biggest amount of security fixes, but usually do not come with API changes or significant alterations in the library’s behavior. This usually means that major and minor versions are the most difficult to update to, with diligent library makers often providing migration guides to support developers in their transition.

Either way, a healthy round of tests is absolutely needed no matter the update. While 99.9% of the times a library version change (speaking about patches) should not affect the code, something may always go wrong. This consequently results in more work to do the bigger the project.

The following pages contain screenshots of selected vulnerabilities for cooking-club-nest-bot. All of these require simple fixes in the requirements file.



To verify that not only simple project like mine are affected by security breaches, I opted to fork another project, Jenkins. I backtracked several months, forked the project and visited Dependabot. I received 21 security alerts. Yet, most of them required little work just like the alerts on cooking-club-nest-bot. The following is a screenshot from the aforementioned fork.

