Detection and Mitigation of Dependencies with Security Risks

Gleb Naumenko
University of British Columbia
naumenko@cs.ubc.ca

Puneet Mehrotra
University of British Columbia
puneet89@cs.ubc.ca

Anna Scholtz
University of British Columbia
ascholtz@cs.ubc.ca

ABSTRACT

todo

CCS CONCEPTS

• **D.2.8 [Software Engineering]: Design Tools and Techniques - Software libraries**;

KEYWORDS

Vulnerabilities, Security, Python, GitHub

1 INTRODUCTION

Security is an important aspect of software design. As stated in recent reports [16], insecure programs still provide significant loss to companies around the world. Credit Union National Association claims that in less than five years, the annual cost of data breaches at the global level will skyrocket to $2.1 trillion [10]. In addition to the financial aspect, there is another one – users' private information. In accordance with Vigilante.pw [13], more than 2100 websites had their databases breached, containing over 2 billion user entries in total.

Problem The modern software security paradigm makes it challenging for developers to maintain their programs secure. To do so, developers should be familiar with up-to-date security techniques, vulnerabilities and periodically update their software. As is often the case, developers lose awareness of the libraries or functions they use if they do not work on a project for too long [15]. Also, many developers are not aware of the security vulnerabilities in the libraries they use, do not know how to apply fixes or might lack relevant information about vulnerabilities [11]. In general, it is tedious to manually look for updates, and one must remember to do so in the first place. In the absence of active monitoring of the project, its dependencies can stay undetected and outdated for long periods of time, increasing the risk of an attack. These problems can be alleviated with tools that notify the developers of the outdated dependencies in their code, suggest alternative safe methods, look for updates and help with the application of fixes.

In this paper we focus on security risks that correlate with the "Top 4 Common Web Security Vulnerabilities", recently published by TheMerkle.co [12]: weak cryptographic algorithms (e.g. SHA1), weak cryptographic parameters (e.g. RSA with key length of 1024), code injection, file hijacking and outdated dependencies. The first two risks are also important in the context of post-quantum cryptography [14].

Contributions We propose Revelio, which is a tool helping software developers to find and update vulnerabilities in their programs. Revelio should be able to:

- Statically identify locations where the developer has used deprecated or unsafe methods in the code and replace it with safe alternatives
- Detect and update outdated dependencies
- Dynamically run the project's own tests to check whether an update or the usage of a safe alternative breaks the code
- Update existing projects via GitHub pull requests
- Identify vulnerabilities during the design phase via an IDE plugin

By running Revelio against existing GitHub projects and conducting a user survey, we are trying to answer the following research questions:

- R1 Can static or dynamic analysis be used to detect vulnerabilities and to verify if the code still runs after an update or modification?
- R2 How many popular projects have dependencies that pose security risks?
- R3 What are the most commonly detected vulnerabilities?
- R4 How many of the suggested changes were developers willing to implement?
- R5 How useful do developers find the IDE plugin while writing code?

Organization In Section II, we provide background information about security vulnerabilities and propose a set of requirements to a tool to explain the context of this work, motivation and implementation of the prototype are described in Section III. We describe limitations of the prototype in Section IV. Section VI contains empirical results collected by a Pull-request study and User study, which are discussed in Section V. We close the paper with related work (Section Vl), future work (Section VII) and a conclusion (Section Vlll).

2 REVELIO

Our tool has been designed to meet the previously defined requirements. It statically identifies locations where the developer has used deprecated or unsafe methods in the code and suggests safe alternatives. It runs tests to check whether the code is broken and needs attention and can update outdated dependencies.

We chose Python as the primary programming language for implementing our tool since our focus is on detecting vulnerabilities in Python projects. The reasons for why we chose Python are manifold: First, a huge amount of software is implemented in Python. On GitHub alone around 2.5 million Python projects are hosted [5]. It is quite likely that many of these projects are used in a context where security is important and potential vulnerabilities might have a large negative impact. Second, various libraries for parsing and analyzing code are already available and can be integrated into our tool. Third, a wide range of known vulnerabilities in Python is already available on various security related websites [3] [1].

The tool is based on Python 3 and can currently be used as a plugin for Sublime Text [9] or as a standalone command-line tool. It can analyze Python files that are either stored on the local

machine or available in a GitHub repository. The output is a report about detected vulnerabilities, outdated dependencies, vulnerable dependencies and executed tests. In the following, we will give a general overview of the tool architecture and detailed descriptions of the most relevant components.

2.1 Implementation

A general overview of the components Revelio is composed of is shown in Figure 1. Revelio can be started using the command-line or by using our plugin that integrates it into Sublime Text. The command-line interface provides options for analyzing files stored on the local machine as well as GitHub repositories. For working with GitHub repositories it is required to provide the URL to the repository. Revelio will automatically clone the repository into a temporary directory. Once all files are locally available the vulnerability analysis will be executed.

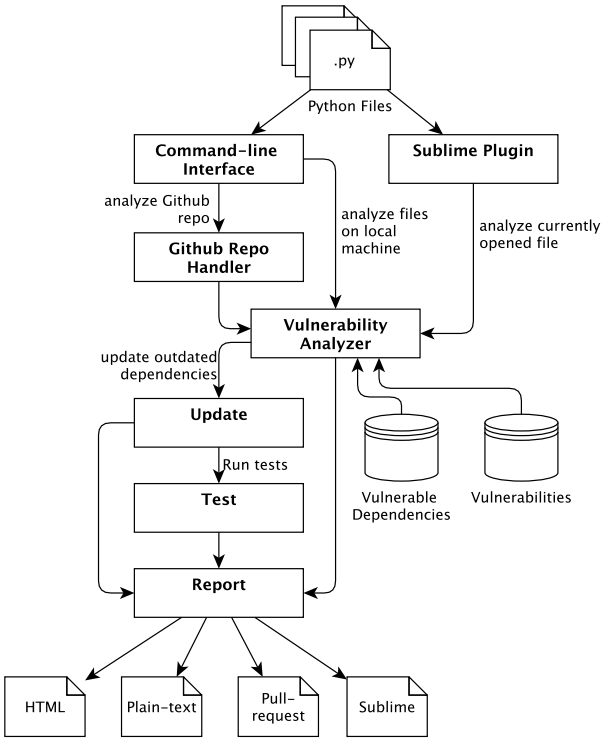

Figure 1: Simplified architecture of Revelio

`Vulnerability Analyzer` implements the core functionality. This component detects vulnerable functions as well as vulnerable dependencies and optionally replaces vulnerable functions if replacement suggestions are available. For this, it relies on known vulnerabilities and known vulnerable dependencies that are stored in databases. These databases are currently maintained manually. They contain information about the severity of a vulnerability, the reason for why it is not secure and, optionally, secure replacements as shown in Listing 1. At the time of writing, we identified 21 vulnerable functions[1]. For detecting vulnerable dependencies, we use safety-db [6] which provides information about insecure versions of Python dependencies.

```
1  [...]
2  "yaml.load":{
3    "severity": "critical",
4    "type":"pickle",
5    "update_with": "yaml.safe_load(___0)",
6    "reason": "Untrusted input can result
7          in arbitrary code execution."
8  },
9  [...]
```

Listing 1: Entry in the vulnerability database

Once the analysis is done, Revelio can check for and update outdated dependencies. Furthermore, it can automatically execute the projects own tests, if available. This is especially useful to make sure replacing vulnerable functions or updating dependencies does not break the code. However, these steps can also be skipped and are currently only available through the command-line interface. Revelio provides several options for generating different reports containing the vulnerability, update and test results: reports are available as HTML or plain text printed to the command line. When using the Sublime plugin, vulnerabilities will be highlighted inline with additional information. Furthermore, for GitHub repositories, Revelio can automatically create pull-requests which replace vulnerabilities with safe alternatives and provide more security-related information.

2.1.1 Detecting Vulnerable Functions. Vulnerable functions are uniquely identified using their full name including module and submodule names (cf. Listing 1 line 2). However, the naive approach of performing a plain-text search using this identifier to detect vulnerabilities in files does not work in Python. The reason for this is that in order to access code in other modules or external packages, these dependencies need to be imported. Python allows to import specific names of a module, as shown in Listing 2 on line 1, and to define aliases for imported modules (cf. 2 line 2). Both of these methods do not introduce the module name from which the imports are taken in the local symbol table. Therefore, developers will use the aliases as well as shortened names in their source code making it impossible to match with the identifiers in the database.

```
1  from Crypto.Hash import SHA
2  from Crypto.Cipher import ARC4 as A
3
4  def main():
5      [...]
6      hash1 = hashlib.md5()
7      [...]
8      cipher = A.new('tempkey')
9      h = SHA.new()
```

Listing 2: Usage of vulnerable functions in Python

Instead, Revelio performs its analysis on the AST (abstract syntax tree) of the Python code. The first step is to extract all function calls

[1] https://github.com/scholtzan/cpsc-507/blob/master/src/data/crypto.json

from the Python file to be analyzed. This will not only extract the full names but also the location in the file. Next, all import statements are determined including aliases. These contain the names of the modules and submodules which can be used to correlate which module provides each function. This way the full function name consisting of module and submodules can be determined and in the next step compared to the known vulnerable functions. At the end of this step, Revelio will have a list of vulnerabilities for each analyzed Python file.

2.1.2 Replacing vulnerabilities with safe alternatives. Revelio offers to replace vulnerable functions with safe alternatives. After the vulnerable functions have been detected and their exact locations have been determined, Revelio will iterate through the Python AST and replace these function calls, if safe alternatives are available. These alternatives are again stored in the database and need to be written as valid Python code (cf. Listing 1 line 5). It is also possible to define which function parameters should be used in the replacement. For this, parameters are identified by their location in the parameter list and followed by "___". For example, in line 5 in Listing 1, `___0` indicates that the first parameter should be used in the replacement function as the first parameter. Finally, Revelio will write the modified Python AST back to the file.

2.1.3 Detecting vulnerable dependencies. The standard way to handle dependency management in Python is specifying requirements in a `requirements.txt` file. While it is widely accepted as a best practice, it is scarcely enforced. There is no one tool like Maven[2] for Java that handles the many diverse ways in which people handle project dependencies and packaging. This variation and lack of consensus on best practices can make it challenging to detect what dependencies are used and handle the dependency upgrading.

To tackle this challenge, Revelio will only look at `import` statements in the code. All packages that are used in the code need to be imported at some point and thus allows retrieving all dependencies used in the code. To determine if a project uses vulnerable dependencies, Revelio first extracts all `import` statements and compares the imported modules to the database containing information about vulnerable dependencies. For each vulnerable dependency, Revelio will return the versions that are insecure as well as a reason for the insecurity. Project maintainers can use this information to inform users about the dependency versions they should avoid.

2.1.4 Detecting outdated dependencies. Checking whether dependencies are outdated is done by extracting imports from the AST and then using the package management system pip[3] to determine the currently installed version. Next, pip can retrieve all available versions of a module of which the newest will be installed. Revelio will run available tests to check if the update breaks the code. If tests fail that were executed successfully before the update, then Revelio will go back to the old version of the dependency. Currently, it is possible to update all outdated dependencies at once or to incrementally update and check if the code still runs. The latter option, however, might be very time-intensive since executing all tests over and over again can take a significant amount of time.

[2]https://maven.apache.org/
[3]https://pypi.python.org/pypi/pip

2.1.5 Testing. Tests are optionally executed after insecure functions have been replaced with safe alternatives or outdated dependencies have been updated. There are several testing frameworks that exist for the python ecosystem, however, there are clear favorites that exist among the developer community. From a preliminary search on GitHub, we determined that *pytest*[4], *nose*[5], and *unittest*[6] are the most commonly used. Each of these testing frameworks has their own unique ways to organize, discover, and run tests [4] [2]. While this divergence is something any automated testing environment has to reckon with, it is also understood well enough that tools have evolved to help deal with this challenge.

Tox[7] is a tool that was created with the aim to standardize the testing effort for python projects. Tox has been designed in a way that makes it continuous integration ready, while still being able to support a wide variety of testing practices. It offers great flexibility to developers in specifying how they want their projects to be tested. Tox allows the user to create a config file for the project that allows the developer to specify the package dependencies that must be fulfilled to test the project, the various versions of the python interpreter that the project needs to be tested against, and allows the user to differentially specify tests that must be run against each target.

Given the popularity of the tox project, it became a natural choice for Revelio. Revelio has a simple strategy for running tests: for a project that has a `tox.ini` file in the repository, use it as is; for a project that doesn't have one, create one on a best-effort basis by filling in details in a template config file. A sample `tox.ini` file is as described in Figure:

```
1  [tox]
2  ignore_errors = True
3  envlogdir = {envdir}/log
4  ignoreoutcome = True
5  envlist = py35, py36
6  skip_missing_interpreters = True
7
8  [testenv]
9  setenv =
10     PYTHONPATH = {toxinidir}:{toxinidir}/
11 whitelist_externals = /usr/bin/env
12 install_command = /usr/bin/env LANG=C.UTF-8 pip
       install {opts} {packages}
13 commands =
14     py.test --timeout=9 --duration=10 --cov --cov-
           report= {posargs}
15 deps =
16     -r/home/puneet/scratch/home-assistant/
           requirements-merged.txt
17     -c/home/puneet/scratch/home-assistant/
           homeassistant/package_constraints.txt
```

Listing 3: Sample tox.ini file

There are several details that need to be considered to fill in the template file:

(1) **Python interpreters:** A project might support multiple python environments. A project usually specifies the python

[4]https://docs.pytest.org/en/latest/index.html
[5]http://nose.readthedocs.io/en/latest/
[6]https://docs.python.org/2/library/unittest.html
[7]https://tox.readthedocs.io/en/latest/

environments that it is designed for in its `setup.py` file that is used by distutils to install the project. If this information is not found in the setup.py file, it defaults to using `['py35','py27','py26','py32','py33','py36']`

(2) **Requirements and Constraints File:** A project may specify several requirements and constraint files that are usually scattered throughout the project hierarchy. The developer might have several reasons for creating multiple requirements files, and they might be used for executing different test suites. The uncertainty in knowing how to use these files poses an interesting challenge while creating the `tox.ini` file. Revelio merges all requirements and constraint files it discovers in the project hierarchy, and for any inconsistencies in the version numbers for packages, it selects the lower version.

(3) **Python path for the project:** This is the root location where the main source code is located in the project hierarchy. It is used because often tests are defined inside some subdirectory and expect the python path to be set accordingly. We currently do not handle the scenario where tests are not defined in the project base directory.

(4) **Test Runners:** A test runner is a framework for executing tests for a project. The test runners that Revelio has been tuned for are pytest, nose and unittest. Revelio utilizes the common underlying mechanism that all test runners utilize pytests and nose work by finding tests that subclass unittest. This also presents an interesting property that is utilized by Revelio: pytest and nose can be used interchangeably to run the tests. Given this equivalence, Revelio tries to use pytest to run the tests. If the tests cannot be run, the errors are logged and later shown to the user.

If no tests were discovered in the project hierarchy, we flag the same to the user. We believe this is important to do since, given the absence of tests, there is no way to analyze the correctness of fixes provided by Revelio. In this case, we cannot vouch for the validity of the patch and whether the tests will pass on applying it. Our warning to the user serves as a disclaimer to this effect.

2.2 Demonstration

2.2.1 Command-line Interface. The command-line interface for Revelio is shown in Figure 2. In this example Revelio was used to analyze a GitHub repository and to generate an HTML report with the results as shown in Figure 3. The results are also printed out on the command-line. Reports contain information about the location of the vulnerability in the code, the reason for why it might be insecure, a severity level and a suggested alternative. Additionally, information about vulnerable or outdated dependencies is provided and an overview of how many tests successfully executed after safe alternatives and updates were applied. For the example in Figure 2, no tests were available and all dependencies were up to date. Also, Revelio could not detect any vulnerable dependencies.

2.2.2 Sublime Text Plugin. The Sublime Text 3 plugin was developed as a part of the Revelio tool. In the current implementation, the plugin has 3 functions: highlighting security vulnerabilities in the code, displaying details related to the selected vulnerability and replacing vulnerable functions with secure alternatives. There are 2

```
$ python cli.py --url https://github.com/scholtzan/cpsc-507-test --html /tmp/result.html --help
Options:
  --url TEXT   URL to a github repository
  --path TEXT  Path to a local project directory
  --replace    Automatically replace vulnerabilities
  --push       Automatically creates pull-request with changes
  --html TEXT  Create HTML report in provided file
  --update     Automatically update outdated dependencies
  --help       Show this message and exit.

No tox.ini file was found. Revelio will create one now at /tmp/1fe9e57e-9451-4699-bd4a-f67f81d6debf

Detected vulnerable functions:
/tmp/1fe9e57e-9451-4699-bd4a-f67f81d6debf/src/foo.py:12:17
        Import: 2:0:ansible.runner
        Detected vulnerability: Crypto.Cipher.Blowfish.new
        Vulnerability reason: Vulnerable to birthday attacks.
        Suggested replacements: Crypto.Cipher.AES.new(___0, AES.MODE_CFB, ___2)
        Severity: critical


/tmp/1fe9e57e-9451-4699-bd4a-f67f81d6debf/src/main.py:8:13
        Import: 3:0:foo
        Detected vulnerability: Crypto.Cipher.ARC4.new
        Vulnerability reason: Vulnerable to many attacks.
        Suggested replacements: Crypto.Cipher.AES.new(___0, AES.MODE_CFB, ___2)
        Severity: critical


/tmp/1fe9e57e-9451-4699-bd4a-f67f81d6debf/src/main.py:18:12
        Detected vulnerability: hashlib.md5
        Vulnerability reason: Can be cracked by brute-force attack and suffers from extensive vulnerabilities.
        Suggested replacements: hashlib.sha512()
        Severity: critical


/tmp/1fe9e57e-9451-4699-bd4a-f67f81d6debf/src/main.py:21:8
        Import: 3:0:foo
        Detected vulnerability: Crypto.Hash.SHA.new
        Vulnerability reason: Attacks can find collisions in the full version of SHA-1.
        Suggested replacements: Crypto.Hash.SHA512.new()
        Severity: critical


Executed Tests:
No tests found or tests could not be executed

Outdated dependencies: All up-to-date
```

Figure 2: Revelios command-line interface

types of highlighting implemented in the plugin. Critical dependencies are highlighted with a red frame (see Figure 4 line 25), others âĂŢ with a white frame (see Figure 4 line 34).

Information related to the vulnerability is shown by hovering over a vulnerability. Displayed details include the vulnerability type, reason, safe alternatives and the severity. There are 3 shortcuts introduced to help developers replace vulnerable functions automatically: replace the selected vulnerability, replace all occurrences of the vulnerability in the file and replace all vulnerabilities in the file.

2.3 Limitations

2.3.1 Python 3 Support. Currently, Revelio is written in Python 3 and only supports analysis of projects written in Python 3. This might pose a problem for older projects.

2.3.2 AST Formatting. Revelio translates Python code into the corresponding Python AST. All operation, such as replacing vulnerabilities, are executed on the AST. After the analysis, the AST is written back as Python code into the original file. However, for some cases, the formatting of the Python code is different from the original formatting written by the developers because the formatting is automatically generated by the Python `ast` library which might follow different formatting rules.

2.3.3 Manually Maintaining the Database. The databases for insecure functions and dependencies with vulnerabilities are maintained manually. Vulnerabilities were collected from different websites [1] [3] [7] [8]. Safety DB is updated once per month but needs to be manually synced with Revelio. Therefore, Revelio might not be able to detect all existing or the most recent vulnerabilities.

2.3.4 Usage Context. Revelio does not consider the context in which a vulnerable function is used. Some of the functions pose a security threat only in certain contexts. For example, `hashlib.md5`

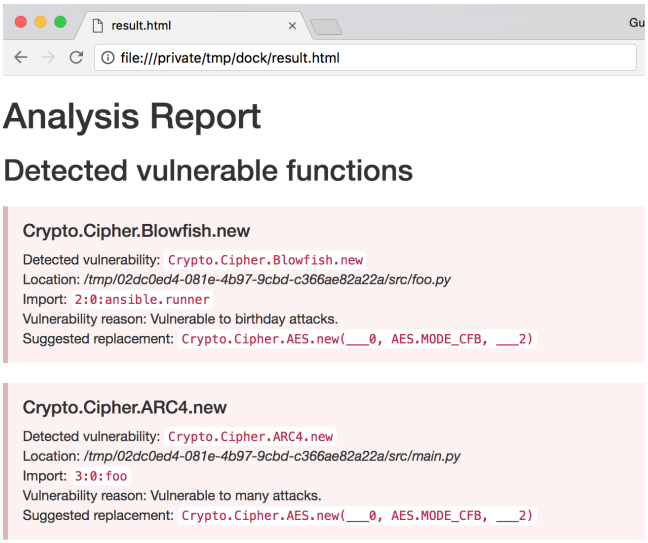

Figure 3: Extract of a HTML report created after the analysis

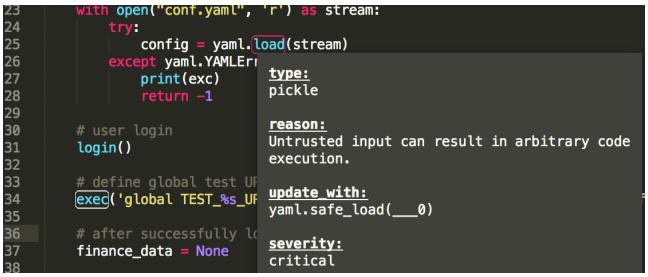

Figure 4: Sublime Text Plugin

would be safe to use for comparing files but not safe in the context of hashing and storing passwords. However, Revelio flags both usages as unsafe and suggests alternatives.

2.3.5 Test Dependencies. Revelio can detect and automatically execute available tests in Python. However, often projects have other external dependencies that are not Python dependencies. For example, some projects required `cmake` to successfully run and execute tests. If not installed, none of the tests can be executed. We ran our pull-request study in a Docker container that had the most commonly used dependencies installed. However, most projects depended on very specific tools that were not installed, thus, most tests could not be executed.

2.3.6 IDE Integration. The Sublime Text plugin currently supports only a subset of the features of the command-line tool. Currently, it is not possible to automatically execute tests or to update outdated dependencies because both require a significant amount of time and would slow down developers.

3 EVALUATION

To answer our research questions and evaluate Revelio we conducted a pull-request study and a user survey.

3.1 Pull-request Study

To evaluate Revelio we analyzed the top 100 and the top 900-1000 Python projects on GitHub. These projects have a large user base, are actively maintained and security vulnerabilities or stale dependencies will have a negative impact on a large number of people and systems. Our goal was to determine how many of the popular Python projects pose security risks and if there is a difference between very and less popular repositories. Revelio cloned each repository, analyzed them for vulnerable dependencies and functions, replaced vulnerabilities with safe alternatives, ran tests and created a pull-request with the modified files and the report. For the evaluation, we decided to send the pull-request only for critical security issues since the security threat of less severe detected vulnerabilities heavily depends on the usage context.

To answer **R2**, Table 1 shows the results of our study. Overall, most of the detected vulnerabilities were less severe. In total, 6 repositories of the top 100 and 8 of the top 900-1000 Python projects had critical vulnerabilities. While the number of vulnerabilities was slightly lower for top 100 projects, the difference is not very significant.

	#Less-critical vulnerable functions (#repos)	#Critical vulnerable functions (#repos)	#Vulnerable imports
Total	103 (40)	41 (14)	182
Top 100	55 (18)	17 (6)	62
Top 900-1000	58 (22)	24 (8)	103

Table 1: Number of vulnerabilities in GitHub projects. The first number indicates the total number of occurrences. The number in parantheses indicates how many projects were affected.

Next, we analyzed the most common insecure functions that were used in order to answer **R3**. The results are shown in Table 2. `exec` and `eval` were detected most often. Both functions execute Python code that can be passed as a string parameter. However, this might allow execution of malicious code. `hashlib.sha1` can also pose a critical security vulnerability when used in the context of password encryption. Through manual inspection, we discovered that this was the case for one project. The other projects use it, for example, to create hashes of files check if they are the

same. We deleted the pull-requests to these projects since they pose no security threats in these cases. `yaml.load`, `pickle.load` and `cPickle.load` are used to read, serialize and deserialize text from files. These functions do not provide strong separation of data and code, and thus allow code to be embedded inside the input [1]. However, some of these libraries provide methods such as `yaml.safe_load`.

	#in top 100	# in top 900-1000
`exec` (warning)	21	16
`eval` (warning)	9	20
`hashlib.sha1` (critical)	5	14
`pickle.load` (warning)	11	18
`hashlib.md5` (critical)	3	8
`yaml.load` (critical)	9	2
`cPickle.load` (warning)	2	0
`tempfile.mktemp` (critical)	0	1

Table 2: Most common vulnerabilities that Revelio detected

Revelio created pull-requests for all repositories with critical vulnerabilities. However, after manual inspection, we deleted six pull-requests. All of these flagged and replaced the usage of `hashlib.sha1` or `hashlib.md5`, however, the usage was not in a security critical context, such as a login or storage of passwords. We collected the status of the pull-requests as well as comments after a duration of 10 days. Table 3 gives an overview of the security vulnerabilities in the remaining pull-requests.

Project	Vulnerabilities	Status after 10 days
https://github.com/localstack/localstack	`yaml.load`, `hashlib.sha1`, `exec`	Merged
https://github.com/facebookresearch/Detectron	`pickle.load`, `yaml.load`, `cPickle.load`	Pending
https://github.com/tensorflow/magenta	`yaml.load`, `exec`, `hashlib.sha1`	Assigned to reviewer
https://github.com/youfou/wxpy	`hashlib.sha1`	Pending
https://github.com/pallets/click	`tempfile.mktemp`, `eval`	Pending
https://github.com/darknessomi/musicbox	`hashlib.md5`	Pending
https://github.com/openai/universe	`yaml.load`	Pending
https://github.com/bugrevelio/nltk	`yaml.load`, `pickle.load`, `exec`, `eval`	2 thumbs up, sticky issue

Table 3: Results of the pull-request study

Two of the analyzed repositories used `hashlib.md5` as well as `hashlib.sha1` to hash passwords, however for these projects were no responses to the pull-requests received. Our modified files did not break any tests that were executed after the changes were pushed to GitHub.

For investigating **R4**, we analyzed the comments and reactions we received on Revelios pull-requests. All of them were positive. Developers of one project even suggested alternatives for `exec` and `eval`: "*Regarding eval(), I agree that there should be a better way to read the data and an alternative would be to use literal_eval() to evaluate strings*"

Although the projects have between 5 to 231 contributors, not all pull-requests got responses within 10 days. The average time between creating a pull-request to merging was between 2 to 24 days for the eight projects. Since no pull-request was rejected, it is quite possible that it might get merged after the study ended.

Overall, one pull-request was merged, for another project a sticky issue was created in order to find fixes for security vulnerabilities that Revelio detected but did not have safe alternatives for. Another pull-request was assigned to a reviewer but did not get merged within 10 days. All in all, we think that developers found the automatically created pull-requests quite useful and are not objected to merging them into their code.

3.2 User Survey

To assess the utility of our sublime plugin and to answer **R5**, we interviewed 7 developers about their experience in writing secure code in Python. To better understand their experience with the domain, we asked some preliminary questions that probed the participants about their expertise in analyzing code for security flaws. The interview also sought to examine the workflow developers use to update their knowledge about the fast-moving space of security, and the information sources they consult to do the same.

After the initial round of questioning to establish basic facts about the participants, we asked the developers to read a code snippet and try to find vulnerabilities. We had constructed a 50 line code snippet[8] for this task and had included 5 unsafe function calls. The participants were provided a simple text editor for this task, and no tooltips were available to help them reason about the validity of their reasoning. The participants were not allowed to consult online resources either.

Next, we explained to the participants the way Revelio plugin for Sublime Text works and how it indicates the vulnerabilities in their code, and the messages it displays for every vulnerability it finds. We asked the developers about their experience using the plugin and asked them questions about how it fits with their workflows, and what changes might make them adopt it.

Most participants we interviewed, as show in Table 4, either did not care about security vulnerabilities in their code [HG,LC,AJ,FR], or relied on a formal code review process [LP,ND,VK] to become aware of them. The participants who cared less about security were academics and revealed that they cared less because their code was primarily used for research prototypes. The participants did not have a well-defined source of information when it came to checking for new vulnerabilities, and they relied on transparent and non-disruptive library upgrades, or some explicit notification (like a review comment or pull request) to take notice of the defect [FR,ND,VK,LP]. All participants preferred directed comments about specific items to change in code, than generic advice about scanning for vulnerabilities.

[8] https://github.com/scholtzan/cpsc-507/blob/master/userstudy/small-example.py

Developer	Age	Job Title	Proficiency in Python	Domain expertise	Number of vulnerabilities identified	Is the tool useful	Usability of the tool	Will you use it?
HG	24	Graduate Student	3	2	3	4	5	2
LC	23	Undergraduate Student	2	1	0	5	5	3
AJ	26	Graduate Student	2	2	4	4	5	4
FR	25	Graduate Student	3	2	4	4	5	4
LP*	29	Solutions Integration Engineer, JDA Software	2	1	1	5	5	3
ND*	28	Member of Technical Staff, NetApp	4	3	4	3	5	4
VK*	28	Technology Associate, Goldman Sachs	4	2	2	4	5	4

Table 4: User demographics data. Participants marked with an asterix(*) were interviewed remotely over Skype. The ratings are on a scale of 1 to 5, where 1 is the worst and 5 is the best possible score for that question

The number of vulnerabilities participants identified is mentioned in Table 4. Depending on their experience with the language, the participants spent varying amounts of time analyzing the code. The most commonly identified vulnerability was the use of `md5` hashing to save passwords (all participants other than LC were able to identify this), and none of the participants managed to identify the `yaml.load` vulnerability.

All participants liked the Sublime plugin as it helped to confirm or dispel doubts about the code they were inspecting and in general were favorable of the idea to use a static analysis tool to highlight potential security vulnerabilities in code. All participants agreed strongly to the utility of having such a tool available to them either as an IDE plugin or as part of a pre-commit lint checker.

When asked about potential improvements to the tool, most participants wanted some form of integration with DevOps tools like pre-commit lint checking[ND,FR,LP,AJ,VK]. Another common ask was to have the ability to dismiss warnings and prevent certain kinds of warnings from being flagged [HG,LC,AJ]. Some developers had concerns about the effort involved in updating the vulnerability list [FR,AJ], and indicated the scenario for having a self-updating vulnerability database.

4 DISCUSSION

To answer **R1**, as shown in our evaluation, Revelio is able to detect all functions and dependencies that are considered as insecure and stored in the databases using static analysis. However, as mentioned before, some functions like weak cryptographic functions might be safe to use, for example in non-cryptographic contexts. These functions are still flagged as vulnerabilities which results in a relatively high false positive rate for certain vulnerabilities. For instance, in our pull-request study, pull-request of six projects were removed since they used weak cryptographic functions in safe contexts. Although the false positive rate might be high, we believe that it is still valuable for developers to be aware of potential security issues.

Revelio can also dynamically verify whether the code is still running after updating dependencies or replacing vulnerable functions. However, for this projects need to provide tests that conform to the format described in Section 2.1.5. Also, tests might fail to run if other required dependencies are not installed.

4.1 Threats to Validity

Most of our participants in the user study work in academia. Academics are less concerned (confirmed through the user study) with the implications of insecure code and have less experience. The results might be different for developers working in industry. None of the participants have any specialized experience dealing with security issues in software. This we believe this is the norm, and therefore should not impact the generalizability of our results.

None of the people we interviewed for the user study was very experienced in writing production-ready Python code. This can impact the validity of our claim regarding the utility of this tool. However, we believe that even for experienced developers, having a handy resource that verifies their code against a CVE database is useful and can help them scan for vulnerabilities that they are not abreast of, or might forget to look for.

Non-standard test scripts: There is a lot of variation among Python projects in the location and mechanism of test scripts. This can make it difficult to run the test from the tox environment. Also, some python packages needed by the projects we analyzed had dependencies that needed compiling in special ways - some packages had C bindings that needed to be compiled. We made a best-effort installation effort to get all the dependencies ready

for testing the project, however, for most projects could not be executed. While this does not have a direct influence on detecting vulnerabilities, it might have had a minor impact on the pull-request study. Not all of the projects had continuous integration set up. Project maintainer might have been reluctant to merge the changes if they did not know whether the code would break. However, for the project that automatically ran tests after pushing code, all of the tests succeeded.

We only selected 200 relatively popular Python projects from GitHub. To get more representative results a much larger number of projects would be necessary. However, we believe that this number already gives a good indication of whether our tool might help developers in detecting vulnerabilities.

Our tool marks functions as potential vulnerabilities although they might not pose a security threat in the context they are used. An example for this is `hashlib.md5` as discussed earlier. This might have resulted in many false positives. Nevertheless, we think it is still good to know for developers that they use functions that might result in vulnerabilities. This way they can make sure that they actually use these function safely.

5 RELATED WORK

There has been a lot of prior work in analyzing developer response to pull-request style notifications to update dependencies in their projects and many empirical studies on awareness and perception of security vulnerabilities. There has also been some prior work that analyzes the impact of dependency management systems and software ecosystems on the ease of managing vulnerable dependencies.

5.1 Awareness of Outdated Dependencies and Security Vulnerabilities

Requires.io[9] sends notifications if a Python dependency is expired. It monitors git repositories, however, a free plan is only available for using public repositories. All security advisories are confirmed manually and it does not provide the possibility to update outdated dependencies.

Greenkeeper.io[10] updates `npm` dependencies of Github JavaScript projects in real-time. It runs tests and notifies when the code breaks. Greenkeeper.io offers several pricing plans however no free version is available.

GitHub provides badges[11] that can be included in the project description and indicate, for example, if the project uses outdated dependencies or fails to compile. While this gives some indication to users and developers, it does not actively try to fix these issues.

Our developed tool offers a command-line interface as well as an IDE integration to analyze locally stored Python projects as well as repositories on Github. Dependencies get updated and the developer will see whether the updated code is broken by running all unit tests. In addition to dynamically checking whether the code breaks our tool also employ static analysis to check where unsafe or deprecated methods are used.

[9] https://requires.io/
[10] https://greenkeeper.io/
[11] https://github.com/badges

5.2 Impact of Automated Pull Request Mechanisms

There have been previous studies that have examined the impact of automated pull requests on the chances that the developer might incorporate suggestions for updating dependencies [17]. In their work, Mirhosseini et.al analyzed several GitHub projects to observe if the use of badges, automated pull requests and notifications had any change in the upgrade behavior. Their results find that projects that use these automated mechanisms have a higher upgrade turnover. This work is based on an empirical study that does not differentiate between security vulnerabilities in code and out-dated libraries a project uses. By providing updates via pull-requests, Revelio is useful for developers to upgrade their project dependencies as well as keep their code safe and up-to-date.

6 FUTURE WORK

6.1 Accuracy

As it was mentioned in Section lV, current implementation of Revelio has a high rate of false positives for certain functions. Recent studies have shown that false positives might dramatically affect user experience and encourage developers to disable tools [19] [18]. As for Revelio, one of the most obvious causes of false positives is the use of weak cryptographic functions in non-cryptographic contexts (for example, MD5 for internal checksums). We see two ways of solving this problem: first, more sophisticated static analysis to identify which warnings can be ignored could be implemented. Second, Revelio could introduce a user interface or an annotation-based notation to developers allowing them to disable warnings in a particular context.

6.2 Support of Python 2 and Python 3

Currently, only Python 3 projects can be analyzed. In the future it would be useful to also provide support for Python 2 projects. This would allow to analyze older projects that might have more vulnerabilities than newer projects.

6.3 Managing Vulnerabilities

In the current implementation the vulnerability databases have to be managed manually. For this we extracted known vulnerabilities from different websites. For future versions it might be valuable to automatically extract potential vulnerabilities from websites or other sources. However, solving this problem might be very difficult since vulnerabilities are mostly described using natural language.

6.4 Support Other Environments

While Revelio only supports the analysis of Python projects at the moment, we believe that it would be helpful for other environments, too. However, having support for projects written in other programming languages would require a lot of changes in the current implementation of Revelio. Also executing tests or retrieving dependencies is very specific for each programming language. Nevertheless, many concepts Revelio is based on could be reused to develop a more advanced version for supporting other environments.

6.5 IDE Improvements

Our user survey revealed a few improvements users would like to see in the Sublime Text plugin. One is to be able to manually dismiss warnings or to indicate warnings in the scroll bar of Sublime Text for easier navigation. Additionally, it might be useful to be able to execute tests from Sublime Text and to update outdated dependendencies.

7 CONCLUSION

Developing and maintaining secure programs is a challenge, due to the big effort and cyber security knowledge required. Our study showed that vulnerabilities are present in popular projects on GitHub. To help developers, we designed a tool Revelio, which detects vulnerabilities in Python code, provides safe alternatives and updates outdated dependencies. Automatically generated pull-requests submitted via Revelio to various GitHub repositories fixing security issues were appreciated by developers. A user survey of the Sublime Text plugin has shown that developers find it useful and easy to use for finding vulnerable functions while writing code.

We think that answers to the research questions in this study are valuable for tools to maintain low-level software design awareness as well as for tools for detection and mitigation of vulnerabilities. Our code is published on: https://github.com/scholtzan/cpsc-507

REFERENCES

[1] Avoid dangerous file parsing and object serialization libraries. https://security.openstack.org/guidelines/dg_avoid-dangerous-input-parsing-libraries.html.

[2] Changing standard (python) test discovery. https://docs.pytest.org/en/latest/example/pythoncollection.html.

[3] Cve details - python. https://www.cvedetails.com/vulnerability-list/vendor_id-10210/product_id-18230/Python-Python.html.

[4] nose - finding and running tests. http://nose.readthedocs.io/en/latest/finding_tests.html.

[5] Python projects on github. https://github.com/search?l=Python&q=language%3APython&ref=advsearch&type=Repositories&utf8=%E2%9C%93.

[6] Safety db. https://github.com/pyupio/safety-db.

[7] Security tracker - python. https://securitytracker.com/archives/target/1631.html.

[8] Stackoverflow - exploitable python functions. https://stackoverflow.com/questions/4207485/exploitable-python-functions.

[9] Sublime text. https://www.sublimetext.com/.

[10] Data breach costs will soar to $2t: Juniper. http://news.cuna.org/articles/105948-data-breach-costs-will-soar-to-2t-juniper, 2015.

[11] Cloudpassage study finds u.s. universities failing in cybersecurity education. https://www.cloudpassage.com/company/press-releases/cloudpassage-study-finds-u-s-universities-failing-cybersecurity-education/, 2016.

[12] Top 4 common web security vulnerabilities. https://themerkle.com/top-4-common-web-security-vulnerabilities/, 2017.

[13] Vigilante.pw - the breached database directory. https://www.vigilante.pw/, 2018.

[14] J. Buchmann and J. Ding. *Post-Quantum Cryptography: Second International Workshop, PQCrypto 2008 Cincinnati, OH, USA October 17-19, 2008 Proceedings*, volume 5299. Springer Science & Business Media, 2008.

[15] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue. Do developers update their library dependencies? *Empirical Software Engineering*, 23(1):384–417, 2018.

[16] K. Lab. Damage control: The cost of security breaches it security risks special report series. https://media.kaspersky.com/pdf/it-risks-survey-report-cost-of-security-breaches.pdf, 2017.

[17] S. Mirhosseini and C. Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 84–94, Piscataway, NJ, USA, 2017. IEEE Press.

[18] T. Muske and U. P. Khedker. Efficient elimination of false positives using static analysis. In *Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on*, pages 270–280. IEEE, 2015.

[19] J. Park, I. Lim, and S. Ryu. Battles with false positives in static analysis of javascript web applications in the wild. In *Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on*, pages 61–70. IEEE, 2016.