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

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 In the absence of active monitoring of the project, its dependencies can stay undetected and outdated for long periods of time, increasing the risk of an attack. These problems can be alleviated with tools that notify the developers of the outdated dependencies in their code, suggest alternative safe methods, look for updates and help with the application of fixes.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(e.g. SHA1), weak cryptographic parameters (e.g. RSA with key length of 1024), code injection, file hijacking and outdated dependencies.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.

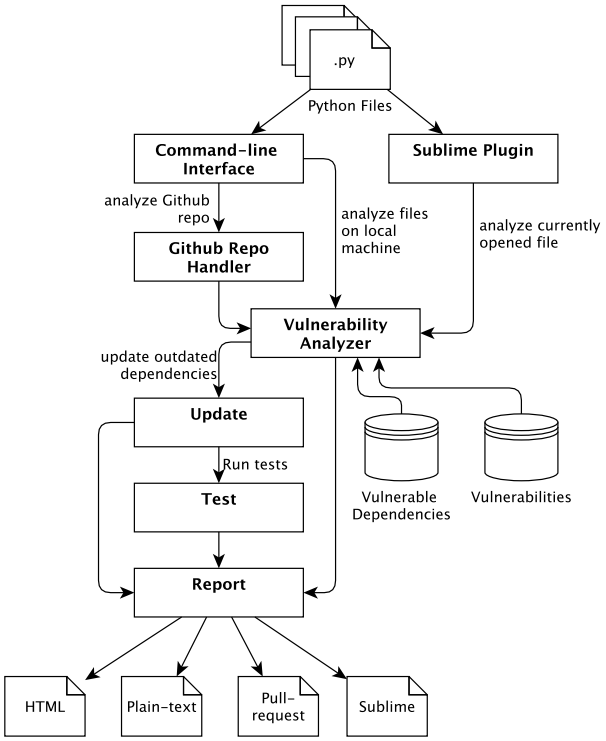

Figure 1: Simplified architecture of Revelio

`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.

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.

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

```
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

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 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.

2.1.5 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 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

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

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

```
$ 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

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` 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

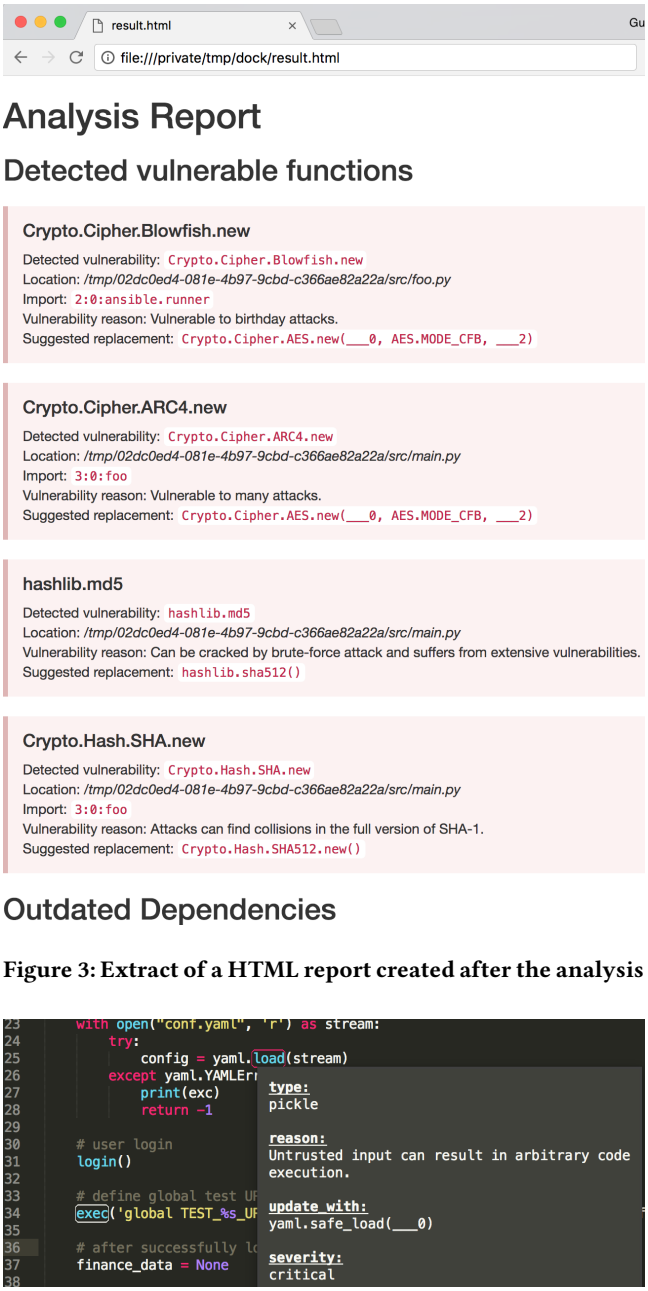

Figure 3: Extract of a HTML report created after the analysis

Figure 4: Sublime Text Plugin

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

4 DISCUSSION

5 RELATED WORK

6 FUTURE WORK

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.