# Detecting Vulnerabilities in Source Code

CSCI 680: Tanvi Padhye [z1906477]

## Problem Significance

Unrecognized flaws in software often can be the cause of attackers compromising a program and forcing it to perform undesired behaviors including crashing or misusing user information.Thousands of such vulnerabilities are reported each year to Common Vulnerabilities and Exposures Database. (Total CVE records as of December 2021 are : 164960)[1] These vulnerabilities are often caused due to some subtle errors but they can propagate fast due to code reuse or software being open source. The most famous example of such a vulnerability is the Equifax Breach which was termed as the most economically damaging hack in the U.S. history. Hackers took advantage of a known vulnerability in Apache struts Web application framework which is an open source project and this resulted in a massive exposure of personal information of 143 million US citizens. [9]

Due to the volume of code being developed and the number of features available, developers can no longer verify security flaws manually.Additionally not all developers are software security experts. Traditional vulnerability detection tools rely on static or dynamic code analysis, symbolic code execution or taint analysis.Custom manually generated templates or heuristics which have been developed for a particular task are required for addressing specific vulnerability. Dynamic code analysis techniques such as fuzzing trigger vulnerabilities to find them however they can only check executed parts of the code. Fuzzing makes the analysis unmanageable since it exhaustively executes every single path in the program. Detecting vulnerabilities is one of the tasks which is addressed with new ML approaches. A property of such approaches is that they can work with numeric vectors (feature vectors as inputs).Code embedding which is converting source code into such vectors can be done either manually or using automation by applying ML based techniques. Adapting NLP techniques such as text representation are popular. The accuracy of any prediction model relies on the input provided empirical data is required for the SE task to be solved. Source control management systems such as github are popular in development fields for source code management. Techniques like static and dynamic analysis are used for finding vulnerabilities and are very popular. Machine learning techniques help in shifting the focus from raw source code and mining information beyond the code. With the amount of open source code available for analysis we can utilize this information to train and prevent software vulnerabilities.

## Related Work

Currently there exist static analysis tools,which help in uncovering existing vulnerabilities. Some of the famous tools are Flaw finder, IST4, RATS etc. These tools however are used for specific languages and can only handle certain types of vulnerabilities. FlawFinder [15] for example findsbuffer overflow,risks, race conditions etc. PHP Aspis [11] does dynamic taint analysis to identify XSS and SQL vulnerabilities. KLEE [6] is another tool which is open source. The problem with KLEE is that it requires manual annotation and modification of source code which results in runtime growing exponentially. In addition to such tools work is also being done in the field of machine learning to use tools for program analysis. Some of the works are in [4], [7]

Dynamic analysis is another popular mechanism which is used for detection of vulnerable code. It works on analyzing code by executing it on real inputs. ZAP [3] is one such tool which finds certain types of vulnerabilities in web applications. It however requires users to define scan policies before scanning and requires lot of manual intervention.

Besides all these techniques, machine learning is also becoming popular in detecting vulnerabilities. Works such as [8],[14],[16] focus on detecting vulnerabilities with a combination of analysis of source code.

## Reasearch Hypothesis

Techniques such as static and dynamic analysis are used to identify potential vulnerabilities in source code. Most of the existing work is focused finding new vulnerabilities in source code but what about the existing ones? The existing tools can prove to be problematic due to false positives.With the increase in open source codes being used for development, the chances of vulnerabilities propagating also increases. It is not possible to track and prevent all these vulnerabilities manually hence an automated system is needed. Another purpose is, to track existing defects and work on them not reentering code.

## Data

For the data for types of vulnerabilities,github was scraped using the github api. A sample of the json is as below:


"https://api.github.com/repos/liviucotfas/ase-web-and-cloud-applications-
security/commits/5567b056c8e2ec5621902cccd4c345c69ccf50bc", "sha": "5567b056c8e2ec5621902cccd4c345c69ccf50bc",
"node_id": "MDY6Q29tbWl0Njk0NTA0MDI6NTU2N2IwNTZjOGUyZWM1NjIxOTAyY2NjZDRjMzQlYzY5Y2NmNTBiYw==", "html_url":
"https://github.com/liviucotfas/ase-web-and-cloud-applications-
security/commit/5567b056c8e2ec5621902cccd4c345c69ccf50bc", "comments_url":
"https://api.github.com/repos/liviucotfas/ase-web-and-cloud-applications-
security/commits/5567b056c8e2ec5621902cccd4c345c69ccf50bc/comments", "commit": {"url":
"https://api.github.com/repos/liviucotfas/ase-web-and-cloud-applications-
security/git/commits/5567b056c8e2ec5621902cccd4c345c69ccf50bc", "author": {"date": "2021-05-
22T23:55:46.000+03:00", "name": "Liviu-Adrian Cotfas", "email": "lcotfas@gmail.com"}, "committer": {"date": "2021-
05-22T23:55:46.000+03:00", "name": "Liviu-Adrian Cotfas", "email": "lcotfas@gmail.com"}, "message": "Update 10 -
Vulnerabilities - CSRF (Cross-Site Request Forgery).md", "tree": {"url":
"https://api.github.com/repos/liviucotfas/ase-web-and-cloud-applications-
security/git/trees/084d1c88b3034f19b56e7a300d20f5b54fb712de", "sha": "084d1c88b3034f19b56e7a300d20f5b54fb712de"},
"comment_count": 0}, "author": {"login": "liviucotfas", "id": 2981923, "node_id": "MDQ6VXNlcjI5ODE5MjM=",
"avatar_url": "https://avatars.githubusercontent.com/u/2981923?v=4", "gravatar_id": "", "url":
"https://api.github.com/users/liviucotfas", "html_url": "https://github.com/liviucotfas", "followers_url":
"https://api.github.com/users/liviucotfas/followers", "following_url":
"https://api.github.com/users/liviucotfas/following{/other_user}", "gists_url":
"https://api.github.com/users/liviucotfas/gists{/gist_id}", "starred_url":
"https://api.github.com/users/liviucotfas/starred{/owner}{/repo}", "subscriptions_url":
"https://api.github.com/users/liviucotfas/subscriptions", "organizations_url":
"https://api.github.com/users/liviucotfas/orgs", "repos_url": "https://api.github.com/users/liviucotfas/repos",
"events_url": "https://api.github.com/users/liviucotfas/events{/privacy}", "received_events_url":
"https://api.github.com/users/liviucotfas/received_events", "type": "User", "site_admin": false}, "committer":
{"login": "liviucotfas", "id": 2981923, "node_id": "MDQ6VXNlcjI5ODE5MjM=", "avatar_url":
"https://avatars.githubusercontent.com/u/2981923?v=4", "gravatar_id": "", "url":
"https://api.github.com/users/liviucotfas", "html_url": "https://github.com/liviucotfas", "followers_url":
"https://api.github.com/users/liviucotfas/followers", "following_url":
"https://api.github.com/users/liviucotfas/following{/other_user}", "gists_url":
"https://api.github.com/users/liviucotfas/gists{/gist_id}", "starred_url":
"https://api.github.com/users/liviucotfas/starred{/owner}{/repo}", "subscriptions_url":
"https://api.github.com/users/liviucotfas/subscriptions", "organizations url":


In order to access those first we need to create github auth token which can be done from https://github.com/settings/tokens. For the data for commits [10] was used.

## Method

In general the steps performed were

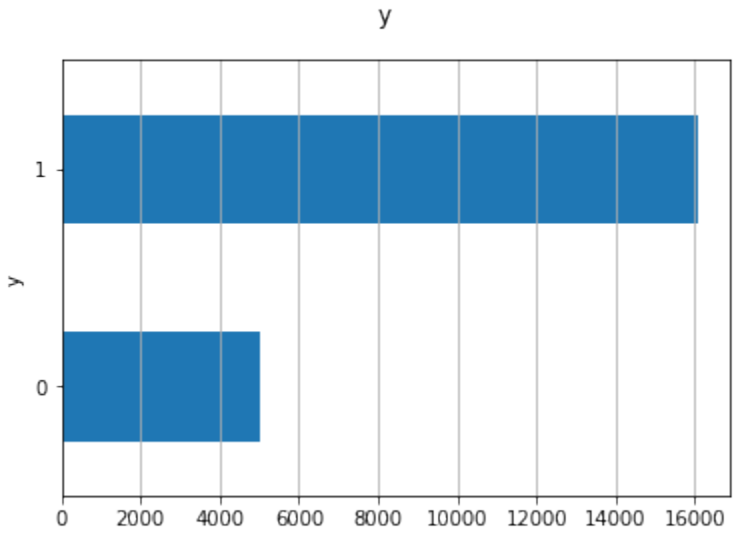
- Setup: which included importing the necessary python modules.
- Bag of Words: Feature engineering and feature selection
- Word Embedding: Fitting a word2vec model with gensim. Deep learning model
- Evaluation

Setup involved importing the libraries such as nltk, sklearn, tensorflow were used. For visualizations, matplotlib and seaborn were used.Data is extracted from github using rest API. The link parameter

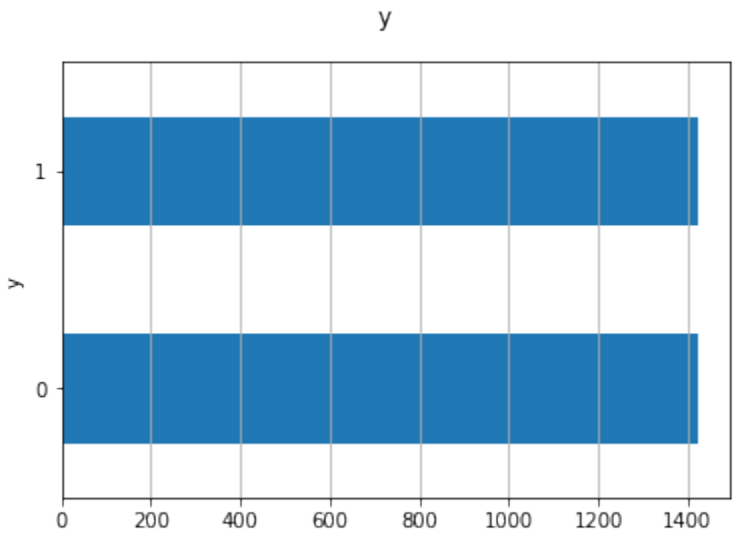
is used to control the pagination in the response. Each type of vulnerability is stored in the respective json file(s). The url for search is url = https://api.github.com/search/commits it takes parameters which were provided as common keywords for different types of vulnerabilities. The keywords were based on [5] After combining all the vulnerabilities training data was created. The final dataset for detecting type of vulnerabilities looks like:

| message | type | filename |
|---|---|---|
| Iranian Government-Sponsored APT Cyber Actors ... | ImproperAccess | /drive/MyDrive/git_commits_Improperaccess2.json |
| exiv2: Fix CVE-2021-3482\n\n References\n... | ImproperAccess | /drive/MyDrive/git_commits_Improperaccess2.json |
| exiv2: Fix CVE-2021-3482\n\n References\n... | ImproperAccess | /drive/MyDrive/git_commits_Improperaccess2.json |
| exiv2: Fix CVE-2021-3482\n\n References\n... | ImproperAccess | /drive/MyDrive/git_commits_Improperaccess2.json |
| exiv2: Fix CVE-2021-3482\n\n References\n... | ImproperAccess | /drive/MyDrive/git_commits_Improperaccess2.json |
| ... | ... | ... |
| Fix security vulnerability in DPMS\n\nChanged ... | ImproperAccess | /drive/MyDrive/git_commits_Improperaccess64.json |
| Fix security vulnerability in DPMS\n\nChanged ... | ImproperAccess | /drive/MyDrive/git_commits_Improperaccess64.json |
| Fix security vulnerability in DPMS\n\nChanged ... | ImproperAccess | /drive/MyDrive/git_commits_Improperaccess64.json |
| Fix security vulnerability in DPMS\n\nChanged ... | ImproperAccess | /drive/MyDrive/git_commits_Improperaccess64.json |
| Fix security vulnerability in DPMS\n\nChanged ... | ImproperAccess | /drive/MyDrive/git_commits_Improperaccess64.json |

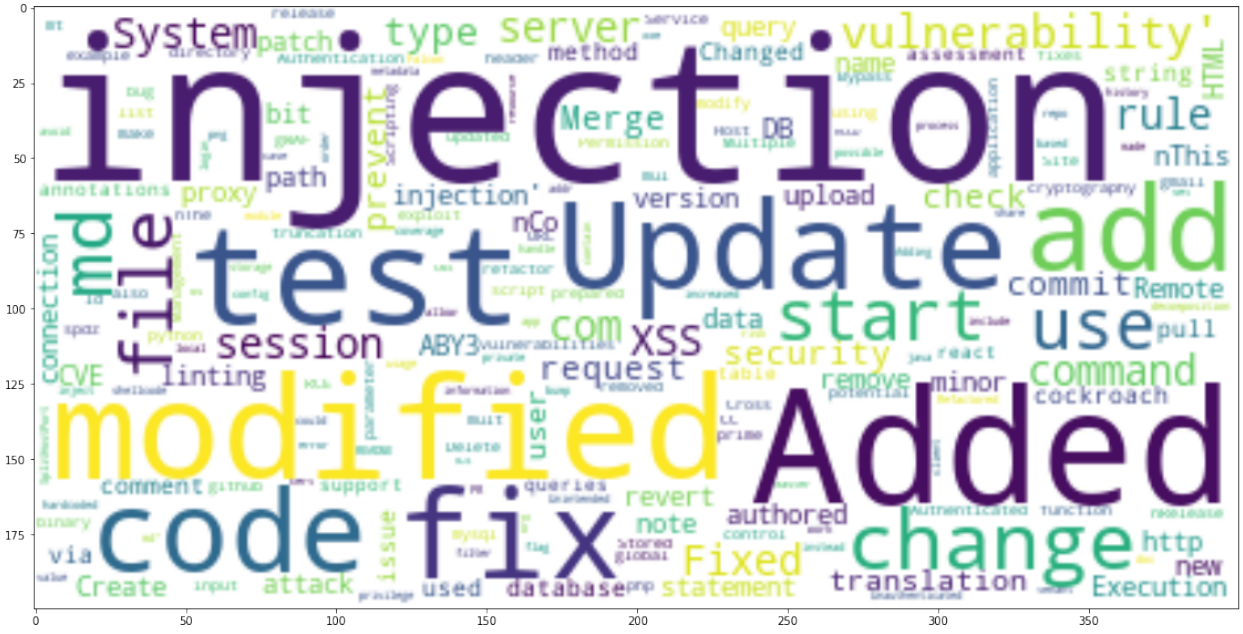
The data for vulnerable and non-vulnerable commit messages was labelled as 0,1 with 0 being commit message for non- vulnerable message and 1 for vulnerable ones. The univariate distribution for the data set shows that it is imbalanced data set.




The group by function was used to generate a balanced data set. The final dataset for vulnerable vs non vulnerable looks like




Word clouds are useful visualization tools, they display most frequent words in the text where the relative size of the words correlates with frequency. Below are the wordclouds for 2 types of vulnerabilities cross site scripting and sql injection which show the frequent words in the messages:




Once the data collection is completed preprocessing was performed with the standard NLP steps of tokenization, removal of stop words etc. After this step the next step was splitting the data into training and testing data sets. The data was split into the ration 70:30 for both the tasks detecting vulnerability as well as detecting the type of vulnerability in it. A bag of words model was used to create the vectors for feeding it to the machine learning model. The Bag of words model builds corpus from the vocabulary and counts the frequency of words in it. As a result of this each word in the corpus becomes a feature and a document is represented by a vector. For example

| | love | programming | also |
|---|---|---|---|
| 1 → I love programming | 1 | 1 | 0 |
| 2 → Programming also loves me | 1 | 1 | 1 |

Count vectors

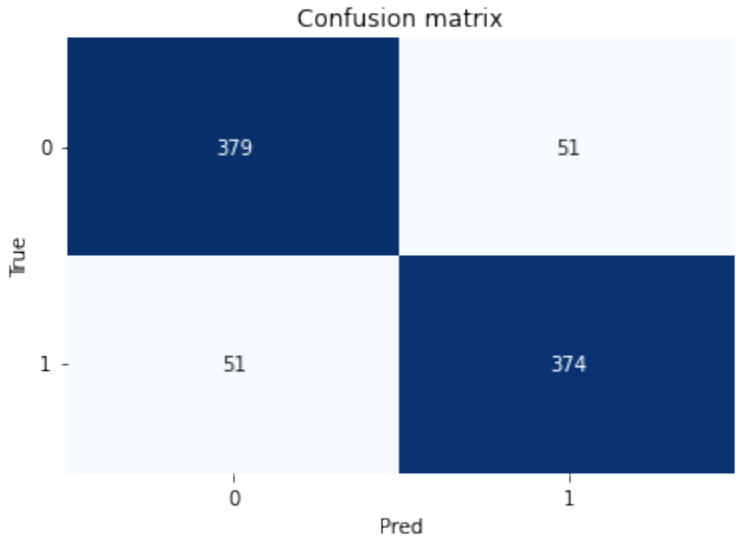
Source : NLP Tutorial for Text Classification in Python [13]

The next step is Feature Engineering. Since it is difficult for machine learning models to work with text data we need to vectorize the text data.
Feature engineering is the process to create features from the data. Tf-idf Vectorizer has been used with a limit of 10,000 words capturing unigrams and bigrams. Word2vec overcomes one of the major drawbacks of using Bag of words technique which is capturing meaning from words. Next the vectorizer was used on preprocessed corpus of the training data. The next step is training and testing the machine learning models. For this purpose I have used a Naive Bayes algorithm which is a probabilistic model which uses Bayes' Theorem which uses probability to make predictions based

on prior knowledge of related conditions. The classifier was trained on the feature matrix created and then tested on the test set using a scikit-learn pipeline. In the next steps evaluation was done. (Reference for the method [12])

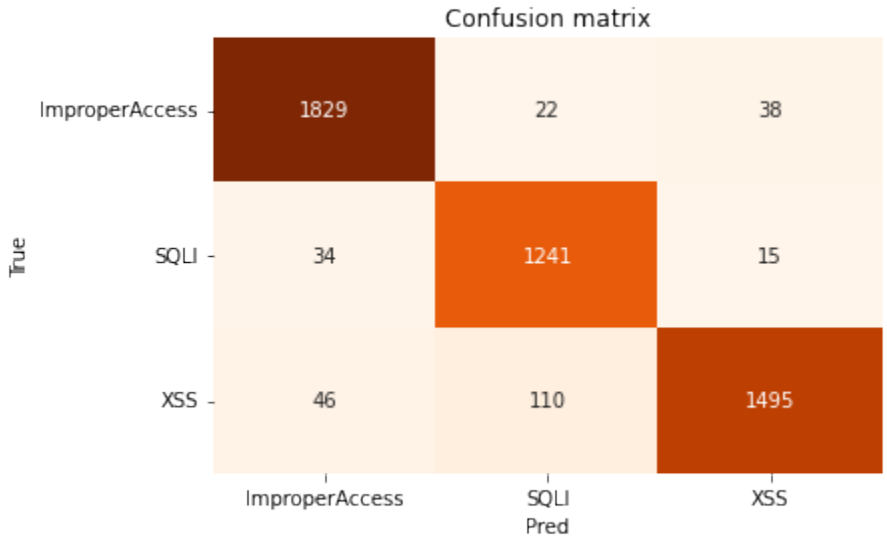
## Evaluation

For the evaluation of Bag-of-Words model I have used accuracy, precision and recall.The reason these metrics have been chosen is because Precision is a reflection of the ratio of true positives vs false positives. For the first part of the project for detecting whether the commit message is for a vulnerability or not, the accuracy is 0.88. The confusion matrix is below




The Bag of Words model got a accuracy of 88%. The explainability of the predictions can has been explained using the lime package. A random sample was picked up from the data set and we can see how the prediction happened.


From the image we can see that the highlighted words are SQL, injection and vulnerability which the model picked up as SQLI vulnerability related words and predicted text to be vulnerable. For the Naive Bayes model the accuracy for detecting type of vulnerability was 95% and the confusion matrix is :




## Conclusion and Future Work

For this project I have implemented a vulnerability identification and classification system. It will help in reducing manual intervention for finding vulnerabilities in code. The results on test and validation data can help in gaining confidence of utilizing machine learning techniques to track and detect vulnerabilities. The model has achieved an accuracy of as high as 88% when dealing with the task of detecting whether the commit message is related to a vulnerability or not. Some of the aspects which can be improved in the project are
1) including other types of vulnerabilities. Currently I have only focused on 3 types of vulnerabilities which are Cross site scripting, sql injection, and improper access. However there are other types as well which can be included. These three type of vulnerabilities were chosen because of their appearance in OWASP top 10 vulnerabilities. [2]
2) In the data set building process I have used a keyword based search approach, there can be places where the search eliminated commits which were actually vulnerabilities.
The next steps can also be to use models like code2vec to analyze and train pieces of code which may introduce vulnerabilities.

## References

[1] Cve.

[2] Owasp top 10 vulnerabilities.

[3] Owasp zap.

[4] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. N-gram-based detection of new malicious code. In *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, volume 2, pages 41–42 vol.2, 2004.

[5] A. Bosu. Identifying the characteristics of vulnerable code changes: an empirical study. 11 2014.

[6] C. Cadar, D. Dunbar, and D. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. volume 8, pages 209–224, 01 2008.

[7] S. Cesare and Y. Xiang. Malware variant detection using similarity search over sets of control flow graphs. *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 181–189, 2011.

[8] I. Medeiros, N. F. Neves, and M. Correia. Automatic detection and correction of web application vulnerabilities using data mining to predict false positives. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, page 63–74, New York, NY, USA, 2014. Association for Computing Machinery.

[9] D. Z. MORRIS. How equifax turned its massive hack into an even worse 'dumpster fire'. *https://fortune.com/2017/09/09/equifax-hack-crisis/*, 2017.

[10] G. Nikitopoulos. Cross-Language Vulnerability Dataset with File Changes and Commit Messages, Jan. 2021.

[11] I. Papagiannis, M. Migliavacca, and P. R. Pietzuch. PHP aspis: Using partial taint tracking to protect against injection attacks. In A. Fox, editor, *2nd USENIX Conference on Web Application Development, WebApps'11, Portland, Oregon, USA, June 15-16, 2011*. USENIX Association, 2011.

[12] M. D. Pietro. Text classification with nlp: Tf-idf vs word2vec vs bert. 07 2020.

[13] V. Rani. Nlp tutorial for text classification in python. *https://medium.com/analytics-vidhya/nlp-tutorial-for-text-classification-in-python-8f19cd17b49e*.

[14] D. Sahoo, C. Liu, and S. C. H. Hoi. Malicious url detection using machine learning: A survey, 2019.

[15] D. A. Wheeler. Flawfinder. *https://dwheeler.com/flawfinder/*.

[16] D. Wijayasekara, M. Manic, J. L. Wright, and M. A. McQueen. Mining bug databases for unidentified software vulnerabilities. *2012 5th International Conference on Human System Interactions*, pages 89–96, 2012.