International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 10, October-2022

Detection of Vulnerable GitHub Workflows usingMachine Learning

Sudarshan B
Dept. of ETE
R V College of Engineering
Bengaluru, Karnataka

Neeraj L
Dept. of ETE
R V College of Engineering
Bengaluru, Karnataka

Nakul R S
Dept. of CS
R V College of Engineering
Bengaluru, Karnataka

Dr. Aravind K Machiry
Dept. of ECE Purdue University
West Lafayette, Indiana

Dr. H V Kumaraswamy
Dept. of ETE
R V College of Engineering
Bengaluru, Karnataka

Dr. Vinay V Hegde
Dept. of CS
R V College of Engineering
Bengaluru, Karnataka

***Abstract*—Having revolutionized the field of software develop- ment and maintenance, Continuous Integration and Deployment (CI/CD) is now a service provided by various commercial platforms of which GitHub is a part of. GitHub, which provides technology for CI/CD, has seen a seismic increase in number of contributors towards open-source projects, thus prompting the need to ascertain and address the various security risks or 'vulnerabilities' lurking within GitHub Workflows. These vulnerable GitHub workflows may be exploited by adversaries and third-parties thus allowing them to gain unauthorized access to privileged data or information. This paper explains some forms of vulnerabilities possible within a GitHub Workflow which corresponds to the features of the Workflow. Secondly, the process of vectorization used to encode these workflow files is explained along with a concise survey of Machine Learning Algorithm used. Finally, the results of the Algorithm is also presented.**

***Index Terms*—CI/CD, GitHub, Workflows, vulnerabilities, vec-torization, Machine Learning**

I. Introduction

Continuous Integration and Delivery, commonly referred to as CI/CD, are software development practices that involve automating integration, testing, and delivery of software in a consistent, regular and automated manner. GitHub Workflows allows the automation of repetitive tasks, enabling the software developers to spend more time in pushing real time updates rather than dealing with the logistics. Having a secure GitHub Workflows run alongside the code mitigates the need for a dedicated DevOps engineer to build a reliable CI/CD pipeline. In order to realize the vulnerabilities that maybe present in a GitHub Workflow, the various components of the GitHub Action must first be understood.

The ease of CI/CD adoption, thanks to third-party services, has its trade-offs. Now developers need to trust third-party CI/CD services to secure the code, artifacts, and secrets from supply-chain attacks [1]. These attacks could have devastat- ing effects, as demonstrated by the recent SolarWinds [2] attack. It is essential to ensure that CI/CD pipelines are correctly configured and not have any security vulnerabilities. Unfortunately, developers are known to mis-configure their CI/CD pipelines [3]. The CI/CD infrastructure itself could have security vulnerabilities [5], thus jeopardizing the security of all the repositories using the corresponding infrastructure. Vulnerabilities may arise in GitHub Workflows when the user configures them incorrectly. This could lead to third party attackers to access sensitive contents and/or pass malicious in- put into the Workflows or repositories as a whole. Such attacks may be expression, command injection, HTTP DOS and so on. This paper aims to detect and classify the Workflows based on their vulnerabilities using a sophisticated machine learning model.

Workflows are configured as YAML files. YAML files have an eclectic of syntaxes which may be unprepossessing to any ML algorithm. The paper first aims to convert the files into vectors by the process of vectorization, where string data is converted into meaningful integer vectors. The vectors are then passed to a Supervised machine learning algorithm to detect vulnerabilities.

II. Proposed Work

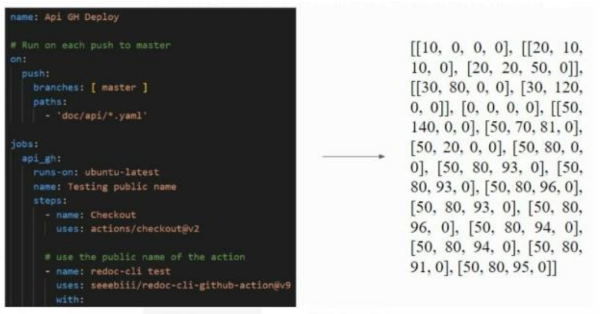
The paper aims to summarize the different modules of the software which eventually classify the GitHub workflows into a binary classification based on their vulnerability. Since, YAML files are very arbitrary and cannot be fed directly into any machine learning algorithm or at least will not be able to have a desired efficiency, they are converted into an alternative form. This alternative form is known as vectors where string data is converted into an array of integers (1's and 0's). Vectorization of YAML files lead to flagging of desired syntaxes which may give pattern of a vulnerability within them.

Once the vectorization is performed on all YAML files, they are saved as a simple `<.csv>` file to be fed into the multi-layer perceptron neural network. However, the size of the vectors is 7040 columns most of which have redundant and unnecessary information. Information gain is performed on the vector where entropy is calculated to reduce the size from 7040 to 1000 columns. The optimal parameters of the network are set after multiple iterations of the training data to get a high efficiency. The training data has a mixture of both classes of the workflows. Once the training is performed on the new vector, it is tested on sample testing data for its classification.

III. METHODOLOGY

The methodology of the project can be divided into three subsections broadly. They are as follows:

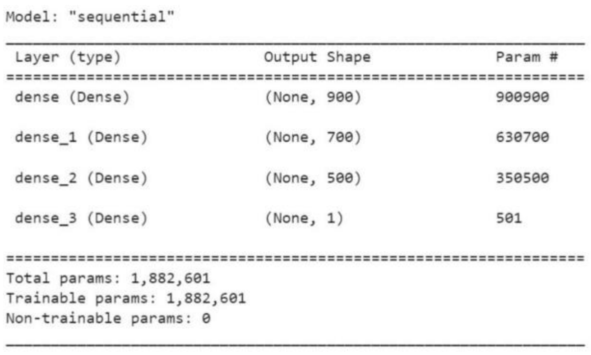
- **Feature Extraction** - The first subsection deals with feature extraction of GitHub workflows from their cor- responding YAML files. Since most of the content of the YAML files are irrelevant, identifying the vulnerabilities has to be first done manually. Here, YAML parsing is a crucial step since it is required for the implementation of machine learning algorithm. This requires a clear understanding of GitHub actions and its security features. A survey of all the high-level and low-level syntaxes possible in these workflows lead to a lookup table which will further be used in the process of vectorization. Since the paper deals with Supervised learning, all the known vulnerabilities have to be listed down and a way to identify them have to be specified.
- **Vectorization** - The second step deals with the conversion of these features into vectors or traces. This is known as trace generation and is required to find the vulnerabilities. Vectors also provide an easier mode of handling and identifying vulnerabilities when it is difficult to directly find them in lengthy and myriad workflows. An easier way to understand vectors, is a pointer to a pointer which gives a continual view of the YAML files rather than a complex user defined structure. YAML files are converted into a vector of 0's and 1's. Each workflow is parsed and its corresponding syntaxes is juxtaposed in the lookup table created in step1. 'on', 'env', 'defaults', 'permis- sions', 'concurrency' which belong to high-level syntax is parsed by their respective functions and appended to the main vector. 'jobs' and 'steps' contain the majority of the vulnerabilities and hence requires a meticulous function to parse them. Each 'step' within a 'job' is checked for its corresponding actions being used and flagged for predefined values. Therefore, each action within a 'step' is parsed into a vector of length 8. All the vectors are appended into a main vector and is flattened and saved as a CSV file (Comma Separated Value).
- **Machine Learning** - The final step of the project han- dles the classification of workflows into two sub-classes (vulnerable and non-vulnerable). This is implemented as a machine learning model using Supervised learning. This process is implemented as a neural network. A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. A multi-layer Perceptron neural network is defined with an equilibrium number of neurons (empirical number based on the size of the vector generated) and is trained. Training parameters are set before hand and the dataset is polarized for class 1 and class 2 workflows. Once the model is trained and ready to be deployed, it is tested scrupulously on other set of workflows.



```
name: Api GH Deploy

# Run on each push to master
on:
  push:
    branches: [ master ]
    paths:
      - 'doc/api/*.yaml'

jobs:
  api_gh:
    runs-on: ubuntu-latest
    name: Testing public name
    steps:
      - name: Checkout
        uses: actions/checkout@v2

      # use the public name of the action
      - name: redoc-cli test
        uses: seeebiii/redoc-cli-github-action@v9
        with:
```

```
[[10, 0, 0, 0], [[20, 10, 10, 0], [20, 20, 50, 0]], [[30, 80, 0, 0], [30, 120, 0, 0]], [0, 0, 0, 0], [[50, 140, 0, 0], [50, 70, 81, 0], [50, 20, 0, 0], [50, 80, 0, 0], [50, 80, 93, 0], [50, 80, 93, 0], [50, 80, 96, 0], [50, 80, 93, 0], [50, 80, 96, 0], [50, 80, 94, 0], [50, 80, 94, 0], [50, 80, 91, 0], [50, 80, 95, 0]]
```

Fig. 1. Deception of Vectorization



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 900)               900900

 dense_1 (Dense)             (None, 700)               630700

 dense_2 (Dense)             (None, 500)               350500

 dense_3 (Dense)             (None, 1)                 501

=================================================================
Total params: 1,882,601
Trainable params: 1,882,601
Non-trainable params: 0
_________________________________________________________________
```

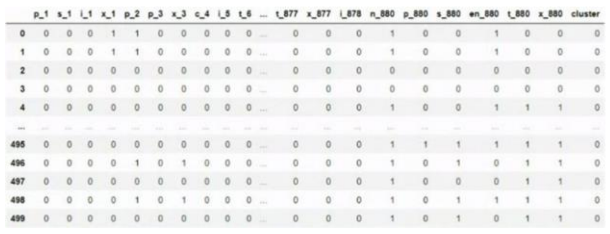
Fig. 2. MLP Model Summary

IV. VECTORIZATION

Vectorization was done in order to convert the string data available in the form of YAML files to integer array data and the results are shown in Fig. 1. This nest array was later flattened and converted to a <.csv> format which served as an input to the MLP Model. The input consisted of a mixture of Class 1 and Class 2 workflows, i.e., Vulnerable and Non- Vulnerable Workflows, as shown in Fig. 3.

V. RESULTS

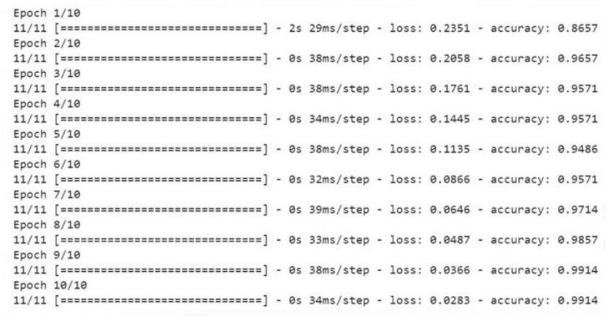
The Training results of the MLP are as shown in Fig. 4. The MLP was designed efficiently with an accuracy of about 94.34% with limited number of workflows and vectors. The Learning rate was set to 0.1 whereas the number of hidden layers was set to 3. Each hidden layer consisted of 900, 700 and 500 hidden neurons respectively and was trained for 10 epochs. The designed MLP was tested out for untrained data vectors and was found to be very efficient with an accuracy



	p_1	s_1	i_1	x_1	p_2	p_3	x_3	c_4	i_5	t_6	...	t_877	x_877	i_878	n_880	p_880	s_880	en_880	t_880	x_880	cluster
0	0	0	0	1	1	0	0	0	0	0	...	0	0	0	1	0	0	1	0	0	0
1	0	0	0	1	1	0	0	0	0	0	...	0	0	0	1	0	0	1	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	1	1	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
495	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	1	1	1	1	1	0
496	0	0	0	0	1	0	1	0	0	0	...	0	0	0	1	0	1	0	1	1	0
497	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	1	1	0
498	0	0	0	0	1	0	1	0	0	0	...	0	0	0	1	0	1	1	1	1	0
499	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	1	0	1	1	0

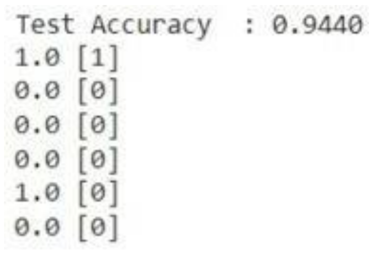
Fig. 3. Input Data to MLP Model

of 94.44% as shown in Fig. 5. The Model was trained ondifferent parameters and an optimal condition was set.



```
Epoch 1/10
11/11 [==============================] - 2s 29ms/step - loss: 0.2351 - accuracy: 0.8657
Epoch 2/10
11/11 [==============================] - 0s 38ms/step - loss: 0.2058 - accuracy: 0.9657
Epoch 3/10
11/11 [==============================] - 0s 38ms/step - loss: 0.1761 - accuracy: 0.9571
Epoch 4/10
11/11 [==============================] - 0s 34ms/step - loss: 0.1445 - accuracy: 0.9571
Epoch 5/10
11/11 [==============================] - 0s 38ms/step - loss: 0.1135 - accuracy: 0.9486
Epoch 6/10
11/11 [==============================] - 0s 32ms/step - loss: 0.0866 - accuracy: 0.9571
Epoch 7/10
11/11 [==============================] - 0s 39ms/step - loss: 0.0646 - accuracy: 0.9714
Epoch 8/10
11/11 [==============================] - 0s 33ms/step - loss: 0.0487 - accuracy: 0.9857
Epoch 9/10
11/11 [==============================] - 0s 38ms/step - loss: 0.0366 - accuracy: 0.9914
Epoch 10/10
11/11 [==============================] - 0s 34ms/step - loss: 0.0283 - accuracy: 0.9914
```

Fig. 4. Training Results for MLP



```
Test Accuracy  : 0.9440
1.0 [1]
0.0 [0]
0.0 [0]
0.0 [0]
1.0 [0]
0.0 [0]
```

Fig. 5. Testing Results of MLP

VI. Related work

Application Programming Interfaces (API's) have been on a burgeon since they offer an interface between software and hardware capabilities of a computer. This paper is a subset of user-guided API misuse detection. API's (methods, functions or kernel calls) can have misuse errors which may be difficult to detect and correct. 17% of the errors come from API misuse errors. These are detected by a series of steps; trace generation, trace encoding and ML through active learning. ARBITRAR
[6] which is the father paper for this, summarizes the detection of misuses in API's and their sophisticated detection process. Vectorization of YAML files which have been discusses above is related to the process of trace generation and encoding.

Timothy Kinsman *et. al.* [7] conducted the first research on how software developers use GitHub actions, a platform which provides automated workflows to maintain GitHub reposito- ries. Since the introduction of this feature many Actions have been built and are being used by repository maintainers but they have not been evaluated properly. Here, the authors aim to understand the impact by addressing three research questions i.e. How do OSS projects use GitHub Actions? How is the use of GitHub Actions discussed by developers? and What is the impact of GitHub Actions? The results of this study indicated an increase in the pull requests being rejected on a monthly basis which accompanied by a decrease in the commits on merged pull requests.

VII. Conclusion and Further Prospects

The paper has summarized the research and implementation to find vulnerable GitHub workflows using Supervised learn- ing. Therefore, the workflows are classified based on known vulnerabilities. The meticulous process of vectorization of YAML files, can further be put across Unsupervised learning models such as clustering to detect unknown vulnerabilities. This will ameliorate GitHub and its underlying CI/CD process to be secure to a great extent.

The approach taken by the paper is sufficient to detect a raft number of vulnerabilities such as expression attacks and command injection attacks. Further enhancement to convert files into vectors will be efficacious in bringing out vulnera- bilities to a greater extent. Since most of the CI/CD software use a pipeline to deploy and update segments of code, this approach of detecting vulnerabilities can be used here to make the pipelines secure and free of attacker-controlled inputs. MLP can used on the vulnerable clusters to understand the patterns and help the user prevent such inadvertent mistakesin the near future.

References

[1] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cap- pos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *28th USENIX Security Symposium (USENIX Security 19)*, (Santa Clara, CA), pp. 1393–1410, USENIX Association, Aug. 2019.

[2] "Solarwinds supply chain attacks." Available athttps://www.mandiant.com/resources/evasive-attacker-leverages- solarwinds-supply-chain-compromises-with-sunburst-backdoor.

[3] C. Vassallo, S. Proksch, H. C. Gall, and M. Di Penta, "Automated report- ing of anti-patterns and decay in continuous integration," in *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19,p. 105–115, IEEE Press, 2019.

[4] C. Vassallo, S. Proksch, A. Jancso, H. Gall, and M. Di Penta, "Configu- ration smells in continuous delivery pipelines: A linter and a six-month study on gitlab," pp. 327–337, Association for Computing Machinery (ACM), 2020.

[5] C. Paule, T. F. Düllmann, and A. van Hoorn, "Vulnerabilities in continuous delivery pipelines? a case study," *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 102– 108, 2019.

[6] Z. Li, A. Machiry, B. Chen, M. Naik, K. Wang, and L. Song, "Arbitrar: User-guided api misuse detection," in *2021 IEEE Symposium on Securityand Privacy (SP)*, pp. 1400–1415, 2021.

[7] T. Kinsman, M. Wessel, M. A. Gerosa, and C. Treude, "How do software developers use github actions to automate their workflows?," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pp. 420–431, IEEE, 2021.

[8] "Keeping your github actions and workflows secure part 1: Preventing pwn requests." Available at https://securitylab.github.com/research/github-actions-preventing-
pwn-requests/.

[9] "Keeping your github actions and workflows secure part 2: Untrusted in- put." Available at https://securitylab.github.com/research/github-actions- untrusted-input/.

[10] "Keeping your github actions and workflows secure part3: How to trust your building blocks." Available at https://securitylab.github.com/research/github-actions-building-blocks/.

[11] "Understanding Github Actions." Available at https://docs.github.com/en/actions/learn-github-actions/understanding- github-actions.

[12] "How we discovered vulnerabilities in ci/cd pipelines of popular open- source projects." Available at https://cycode.com/blog/github-actions- vulnerabilities/.

[13] "Feature selection techniques in machine learning with python." Avail- able at https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e.