COMMIT MALWARE ANALYZER: DETECTING MALICIOUS SOFTWARE THROUGH SEMANTIC ANALYSIS OF MALICIOUS SOFTWARE DEVELOPERS' BEHAVIORS, DESCRIPTIVE ATTRIBUTES, AND CODE PUBLICATIONS

A Research Paper submitted to the Department of Computer Science
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

By

Vanessa Barlow

December 10, 2020

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

ADVISOR
Yuan Tian, Department of Computer Science

Commit Malware Analyzer: Detecting Malicious Software through Semantic Analysis of Malicious Software Developers' Behaviors, Descriptive Attributes, and Code Publications

Vanessa Barlow
Computer Science Undergraduate
School of Engineering and Applied Sciences, University of Virginia
Charlottesville, VA
vmb3eb@virginia.edu

***Abstract*—This research expands upon previous GitHub case studies that classify GitHub developers as either malicious or benign. In the current technical climate, it is crucial to identify malicious users on code sharing platforms to prevent and halt the sharing of vulnerable or dangerous code in the computing community. While this project aims to automate the classification process for GitHub users, it also contributes novel methods for GitHub data collection. Furthermore, the tool developed is a character-aware neural language model which is composed of a convolutional neural network and long short-term memory recurrent neural network. Code contributed by GitHub users are inputted into the tool and the output of "malicious" or "benign" is returned. Results indicate the tool can perform as well as a comparable state-of-the-art tool, reaching an efficient performance level when classifying GitHub user code. The project is still on-going and is in the process of adding GitHub users' behavioral and descriptive attributes to the classification task.**

I. Introduction

As technology advances in complexity and code sharing through collaborative software sharing communities, like GitHub and Bitbucket increases, new threats and malicious behaviors emerge within online developer communities. Online developer communities tend to allow any person to join as a user. Most users pose little to no security threats, however prior research claims that some users obtain malicious intent when joining an online developer community [4]. In fact, approximately 25.1% of GitHub users were identified as malicious, meaning that those particular GitHub users had some security flaw or threat within their account which lead to GitHub removing them from the community.

Knowing that online developer communities are becoming increasingly popular as developers share code for personal and professional use, it is imperative to identify malicious users on these platforms to minimize security risks for developers who take part in these communities. This project dives deeper into this issue by constructing an automation tool to recognize malicious or suspicious GitHub users. For the purpose of the project, malicious and suspicious users were considered as synonymous.

The main contribution of this research entails a case study on GitHub users. The model produced was able to classify users based on the code content published onto GitHub repositories, otherwise known as "commits". Additionally, new methods for data labelling and crawling were introduced in this project.

II. Background

The research expands on the project "Detecting Malicious Accounts in Online Developer Communities Using Deep Learning" which was published at CIKM '19 [4]. This paper and data were investigated to understand where improvement to their work could be made since they also chose GitHub as a case study. From analyzing their project, two notable improvements were identified.

The first improvement that could be made was reforming the labelling process. Labelling of GitHub users as malicious or benign was determined by going to users' profile pages (https://www.github.com/username) and validating the HTTP status code. If the status code was "404", then the user was marked as malicious. Otherwise, the user was labeled as benign. The flaw in this labelling method was that a user's profile page could return a "404" status code if the user simply changed their profile name or deleted their account. The latter scenario was mostly common.

Another improvement involved expanding the data collection process to include GitHub user code from repositories that users contributed to. In the previous study, user code was not involved and classification occurred solely based on GitHub user behaviors (e.g. if the user pushes, pulls, forks repositories) and GitHub user attributes (e.g. user name length, job title, number of followers). By adding user code as a new attribute, the content that the user was actually producing could be accounted for when classifying the user as malicious or benign.

III. Motivation

This project stems from investigating a user's reputation on GitHub. This means looking at what a user has done in the past (ie. do they commonly push, pull, fork repositories) and what code the user has actually pushed onto different repositories. A proposed malicious user detection tool, GitSec, has classified users on GitHub by looking at a GitHub users' behaviors (frequency of pushes, forks, pulls, stars) and descriptive attributes (length of user name, job description, etc.) [4]. Our motivation originates from expanding on this project to not only include GitHub use behavioral and descriptive attributes, but also to include the code GitHub users commit from every repository they have worked on. We believe that if the behavioral and descriptive data are accompanied by a code analysis of GitHub user commits, a malicious user detection tool with higher performance than other state of the art tools, like GitSec, can be built.

Furthermore, having an automated detection system can help companies like GitHub and BitBucket, understand what they should be interested in when looking for malicious users on their platforms. By looking past the author's characteristics and into the user's working projects, companies can begin to understand and identify the vulnerabilities and potential security flaws that malicious users embed. This can then help limit the extent of forking and cloning of vulnerable repositories and can quickly identify users who should be removed from the online developer community.

Moreover, GitHub's policy for flagging users consists of a manual procedure. A user has to submit a content removal request in order for GitHub to consider the user as exhibiting malicious or illegal behavior. When the request is submitted, GitHub has to manually make the decision on whether to contact the user or immediately take them off the site [3]. Some user reports could potentially identify significant security flaws that need to be taken down immediately. Oppositely, some users may falsely report content as illegal or malicious. By integrating our tool with sites like GitHub, this process of flagging users can be automated and avoid untimely matters in the case of grave security flaws. Additionally, false user reports can be eliminated through the automation tool.

IV. Research Question

The following questions are hoped to be answered through the experiment design:

1) What type of model can perform an automated process to flag GitHub users as malicious or benign based on the repositories that they contribute to?
2) Can behavioral and descriptive features of GitHub users help find malicious GitHub users alongside their GitHub commits?
3) What qualifies for a GitHub user to be considered as malicious?
4) Is it possible to predict the activity of a GitHub user?
5) What language do malicious GitHub users prefer the most?
6) Can the model detect any zero-day vulnerabilities?

A. Research Question 1

To answer this question, the experiment consisted of trying to model code snippets and content from GitHub commits. Text classification methods were investigated to perform the modeling. Embedded urls in the code were also extracted and observed to see if they are an additive factor when modeling the GitHub commits,

B. Research Question 2

To resolve this question, the model created will need to classify user data that contains code snippets, embedded URLs, description of the repository the user commits to, the types of behaviors the user does on GitHub, and descriptive characteristics of the user. From there, a comparison on the performance of a this model and a model that can classify GitHub commits can be done. The comparison will determine if behavioral and descriptive features of GitHub users are helpful or are confounding variables.

C. Research Question 3

Through the investigation, the labeled data helped us with the analysis of how a malicious GitHub user may behave. Some probable reasons for why a GitHub user may be considered malicious is suspicious code snippets in commits, suspicious embedded URLs in commits, or commit descriptions that are of a foreign language or have some sort of keyword that draws attention to it.

D. Research Question 4

Based on the prior research [4], there are commonalities between malicious GitHub users and the behavioral activities they perform. To expand on this, a goal of this research will be to try and predict the activity of a malicious GitHub user. For instance, is it possible to predict a GitHub user as malicious based on their push, pull, or fork event pattern or frequency?

E. Research Question 5

When identifying traits of malicious GitHub users, the programming languages they use were taken into account to find out the preferred language of malicious users. This property could be a factor into the final decision making model.

F. Research Question 6

When the model is completed and can account for classifying code and user information, GitHub user data that is collected by the team will train and test the model. Investigation into users that are flagged as malicious will take place if the initial label of the user is benign. These users' behaviors, commits, and characteristics will be processed to see if our tool can identify any zero-day vulnerabilities before GitHub can flag them.

V. Data Collection

A. SySeVR Data

For comparison purposes, publicly available data from the SySeVR project was collected from https://github.com/SySeVR/SySeVR [7]. The data includes 126 types of vulnerabilities from the National Vulnerability Database (NVD) [8] and the Software Assurance Reference Dataset (SARD) [10]. One limitation with this data is that it only contains C/C++ code whereas GitHub repositories can contain a multiplicity of different programming languages.

B. GitHub Advisory Crawler

GIT_ADV_CRAWLER is customized to obtain vulnerable code snippets from the Github Advisory Database (https://github.com/advisories). The crawler visits every git repository and lists the visited advisory URL in the 'visited.txt' to avoid revisiting any repository. It creates a folder named 'data' which consists of different folders for malicious code categorized into 4 levels of severity– Critical, High, Medium, Low. The crawler collects the code in the following ways.

It visits the homepage of the repository and parses through the HTML tags of the URL source. Any content that is within <code></code> is considered as a vulnerable code and appended to the list. One main drawback of this is appending meaningless text to the vulnerable code list making the data noisy. In order to fix this, we have imposed the conditions – the text should not be package_name, the text should contain three or more words. We have manually inspected for almost 100 repositories that a single text within the code tags, split by spaces should contain at least 3 words for it to be considerable.

The crawler visits the different URLs on the homepage. Some of the URLs lead the crawler to the commit page where the vulnerabilities are fixed, i.e., the removed code is compared to the added code. We parse through the HTML source code and gather the code snippets within the tags and having data-code-marker = "-", which represents the code snippets that were removed due to the vulnerabilities. Everything under the data-code-marker = "+" is considered the patched versions and appended to the benign_code list.

Our crawler has crawled through 40 pages in all visiting a total of 2,937 repositories. Out of which we were able to obtain the code snippets for 845 repositories stored in 'data.json'. Using the above conditions, we filtered the empty lists, lists containing the package names, and meaningless text and obtained clean, verified data for 285 repositories.

On verification, we have found that 263 repositories have the correct label for vulnerable_codes and 12 repositories have incorrect labels.

Drawback: A lot of the GitHub repository packages have been protected by 'npm' and our crawler cannot get the codes from those repositories. It requires deeper levels of URL extraction. Also, for every repository, we first need to find the affected versions. Compare them with the patched versions and find the corresponding 'commit'.

.

C. GitHub User Dataset

This dataset was formed with the help of the GitHub data collection from the GitSec tool [4]. The GitHub data collection for GitSec consisted of GitHub users who were registered on GitHub by December 31, 2017. These users were crawled by their user ids and were labeled as malicious or benign based on the HTTP status code when visiting the user's GitHub profile page (https://www.github.com/username). If the HTTP status code returned was "404", the user was labeled as malicious. Otherwise, the user was labeled as benign.

A subset of users from the GitSec data collection [4] were used to explore the author's reputation in this project. A similar data collection process was done to gather more and possibly updated information of the users. Each user in the subset of users that were acquired were queried through the GHArchive project [1] from the year 2014 to the year 2017. The GHArchive holds public GitHub event timelines since February 2012. Event timelines include the frequency of forks, commits, deletes, etc. These timelines are essential to understanding the actual behavior of GitHub users. The GHArchive allowed for an updated GitHub event timeline for each user in our data.

To expand on each user in our data, the GHArchive results were examined in order to gain insight on the repositories that the users in our data had GitHub events with. More specifically, we looked for repositories that the users created and pushed content to. We extracted the repositories that fit this description and queried for code content using Google's Bigquery tool [5] and the GitHub API [2]. The code content from the results were examined for embedded URLs. When a URL was found within the code, a snippet of the code containing the URL and the URL itself was added as attributes to the user the code content corresponded to.

The URLs for each user were then used as inputs to VirusTotal [11]. This third party tool allowed for the classification of malicious URLs that were embedded within the code content. If an embedded URL for a user was found to be malicious, then the label for that user would be revised to malicious if necessary. Otherwise, the labels provided by the CIKM '19 paper [4] were taken as the ground-truth. The final dataset composed of 537 malicious users and 92,209 benign users. This final dataset was used for testing purposes of the model.

VI. Experiment Design

A. Exploring CNN Models

The first challenge was deciding on a classifier that could provide a supervised learning mechanism to model code published on GitHub. Since GitHub users publish a variety of different coding languages to their and other users' repositories, supervised learning models that focus on text classification were explored. Through researching existing models, two baselines were identified as potential convolutional neural network (CNN) classifiers for analyzing GitHub commits.

The first CNN classifier explored is described as a character-aware neural language model since it depends on character-level inputs while forming predictions at the word-level [6]. The model consists of a CNN and a long short-term memory (LSTM) recurrent neural network language model (RNN-LM), where the output of the CNN is the input to the LSTM RNN-LM. This model was explored as the model is able to encode semantic and orthographic information of text, which could be extremely useful when analyzing code and syntax styles from GitHub commits.

The second classifier investigated is a character level convolutional network [12]. The network consists of 9 layers: 6 convolutional layers and 3 fully-connected layers. The input consists of text, which is encoded at the character level. As the characters are processed through the network, a 1-D convolution is applied to each character. In addition, a temporal max pooling technique is used. The output of the 6 convolutional layers are then inputted to the 3 fully-connected layers that consist of 2 dropout modules to maintain regularization. Furthermore, the character level convolutional network was concluded to be an efficient model when classifying text created by users, which can include code published by users on GitHub.

B. Error Analysis

Based on the initial performance results when classifying the GitHub User data and the GitHub Advisory data, the model with the highest average f1-score performance was used in further experimentation for classifying GitHub repository code. The next steps of the experiment consisted of tuning the selected model to learn the different types of text and code that appear in GitHub repositories. Manual efforts and case studies for analyzing model mispredictions took place to determine where the model could improve.

C. Evaluation Comparisons

The last part of the experiment involved evaluating the chosen CNN against the state of the art tool, SySeVR [7]. Our research team ran the SySeVR tool with data provided by SySeVR. The results served as a baseline that was compared to results from running the chosen CNN with the SySeVR data.

Additionally, the chosen CNN was evaluated by training, testing, and validating with the GitHub Advisory data. The results were compared and analyzed to the results from running the chosen CNN with the data from the SySeVR project. The comparisons served as a case study rather than a direct comparison since there may be limitations due to the differing sizes of data and the variety of code vulnerabilities within the data.

Finally, the chosen CNN can be tested with the GitHub User data to identify any zero-day vulnerabilities or malicious users based on the code in user repositories.

D. Adding Author Attributes to the Model

The last phase of the experiment consists of adding a feature to the CNN model to account for GitHub user behavioral and descriptive attributes. When the final model is built, it can be trained, tested, validated on the GitHub User data. The performance of the model will then be compared to GitSec [4].

VII. Evaluation and Performance

A. CNN Results

The two CNN classifiers explored were trained and tested with the GitHub User data and the GitHub Advisory data. The results are shown in Table 1. Based on the results, the character-aware neural language model [6] outperformed the character level convolutional network [12] when trained and tested with the GitHub User and GitHub Advisory data. Thus, the character-aware neural language model was used in further experimentation.

TABLE I
Initial Performance Evaluations of CNN Classifiers

Model	Data	Precision	Recall	F1-Score	AUC
Character-Aware Neural Language Model	GitHub User	0.8	0.71	0.74	0.86
Character-Aware Neural Language Model	GitHub Advisory	0.9	0.85	0.87	0.94
Character Level Convolutional Network	GitHub User	0.71	0.65	0.67	0.82
Character Level Convolutional Network	GitHub Advisory	0.81	0.77	0.79	0.9

B. Analysis of Model Predictions

The character-aware neural language model was able to correctly predict code snippets with representative key words. For instance, an ”exec” call in a C++ script would trigger the model to predict the code as malicious. Furthermore, the results showed that there were no false positives, meaning that all benign code was classified as benign.

The model’s mispredictions occurred as a result of various lengths of vulnerable codes. For instance, if a commit is only a few lines, it may be classified as benign even if it has malicious intent. Similarly, if there is a malicious commit containing an entire class, the model may fail to classify the commit as malicious. In this scenario, the model fails to capture the contextual coding information due to the length of the code and the dependency of the commit on other parts of code in the repository.

Further analysis on the data indicated similar conclusions. Regarding the model’s mispredictions, a summary of the conclusions can be placed into two categories: noisy data and diverse programming languages. Noisy data occurs as commits to repositories could include urls, commit or log information, and licensing agreements. All three types of noisy data can occur within benign and malicious commits.

Thus, the classifier has trouble identifying the integrity of the commit. Another key problem that was noticed is that the classifier deals with commits than can span across many different programming languages. For example, there are commits using JavaScript, C++, or Java. The issue with this is that the classifier has a tougher time distinguishing different languages from each other and when learning one language, can misinterpret another.

C. SySeVR Comparison

To evaluate the CNN classifier, the classifier was compared to SySeVR [7]. SySeVR and the character-aware neural language model was ran with the SySeVR data. The results are shown in Table 2. The outcome indicates that the character-aware neural language model performs at a comparable level to the SySeVR classifier. Thus, comparisons between the two models can be developed since the character-aware neural language model has similar scores in each performance field.

TABLE II
BASELINE COMPARISON OF SYSEVR

Model	Data	Precision	Recall	F1-Score	AUC
character-aware neural language model	SySeVR	0.93	0.96	0.94	0.95
SySeVR	SySeVR	0.92	—	0.93	—

The next experiment consisted of training, testing, and validating the character-aware neural language model with the data crawled from GitHub Advisory. The results from this procedure are shown in Table 3.

TABLE III
CHARACTER-AWARE NEURAL LANGUAGE MODEL PERFORMANCE WITH GITHUB ADVISORY DATA

Model	Data	Precision	Recall	F1-Score	AUC
character-aware neural language model	GIT_ADV	0.85	0.80	0.81	0.84

The results in Table 3 show that the character-aware neural language model has the ability to correctly classify malicious and benign code at a high percentage. However, comparing to the results in Table 2 where the model was run with the SySeVR data, the character-aware neural language model has a more difficult time classifying the GitHub Advisory data. This can be due to the size of both data. Furthermore, the CNN shows promise that it can classify GitHub commits as malicious or benign.

The last step of the experiment is to classify the GitHub user data to see if any zero-day vulnerabilities can be identified. This part of the experiment is still being developed by the research team. Additionally, the integration of author attributes to the CNN is still in the process of being constructed by the research team.

VIII. RELATED WORKS

A. GitSec

GitSec [4] is a tool that determines the legitimacy of GitHub users. GitSec takes in account the descriptive features extracted from a GitHub user's page as well as the user's dynamic behavioral data recovered from the GHArchive [1]. The dynamic behavior data is processed by creating two user activity sequences and applying the sequences to a parallel neural network. GitSec proposes its final classification through a supervised deep learning model.

B. SySeVR

SySeVR [7] uses deep learning to detect software vulnerabilities in C/C++ source code. The deep learning approach allows for an extraction of syntax and semantic information from vulnerabilities.

C. OSF

OSF [9] is a vulnerability detection system that uses machine learning techniques to detect vulnerable C/C++ functions. The project not only entails the creation of a new vulnerability detection tool, but also compiles supplementary datasets of potential exploits discovered through static analyzers. The tool is trained with the final dataset using deep feature representation to classify vulnerabilities.

D. VulDeePecker

VulDeePecker [13] is an innovative approach to an automatic vulnerability detection system that focuses on minimizing false negative rates. The tool is composed of a deep learning mechanism that classifies snippets of code, called code gadgets. The code gadgets are transformed into vectors that can be processed by the tool. Results have shown that this tool achieves fewer false negative rates compared to other software vulnerability detection tools.

IX. CONCLUSION AND FUTURE WORK

As of now, the project is a continued study since descriptive and behavioral features of GitHub users have not been integrated into the model. The character aware neural language model will need to include additional features that permit the modelling of different author attributes. Once this occurs, a greater conclusion can be drawn past the conclusions of GitSec [4]. While GitSec provides an understanding of the author attributes that exhibit a high significance to the classification process, our model will be able to add to this by comparing the author attributes' significance to the author's code significance. Additionally, when the final project is completed, research question 2,4, and 6 will have a definitive or better answer.

Throughout the research and static analysis efforts, there was a notable amount of Java code that could be defined as "crypto mining" code. When statically analyzing the code more thoroughly, it was identified that there was a prevalence of GitHub users who have been labeled as malicious due to crypto classes they built and used. Furthermore, the research team was interested in the possibility of identifying different

types of cryptojacking and the frequency it occurs on GitHub. This could be an expanded part of the project as we can look more into the prevention, detection, and recovery of crytpojacking in online developer communities.

Moreover, our model can be expanded past GitHub into other online developer communities such as Bitbucket. The performance from this extended work can be compared to the performance of our model in the GitHub setting.

Lastly, the project contributes important data crawling methods for GitHub user data and GitHub Advisory data. These new methods of data crawling and cleaning can be used in future projects where the study revolves around GitHub data.

X. Teams that Contributed to the Project

The paper's contributions were developed by a team of UVA and UCSB affiliates. The UVA team, directed under Professor Yuan Tian, consisted of 4 computer science students: graduate student Faysal Hossain Shezan, graduate student Kamya Mehul Desai, undergraduate student Vanessa Barlow, and undergraduate student Mahesh Menon (summer intern). The UCSB team, directed under Professor Yu Feng, was composed of one graduate student, Yanju Chen. Both teams worked in conjunction with Google advisors for guidance on the project.

Faysal Hossain Shezan was the team lead for the UVA team. He worked heavily on investigating different CNN models and fine tuning the character aware neural language model. He also oversaw the entire team and the course of their work.

Kamya Mehul Desai worked on implementing and creating the GITADVCRAWLER.

Vanessa Barlow worked on the implementing a crawler for and finalizing the GitHub user dataset. In addition, she worked on investigating the GitSec project [4] to learn more on how the commit malware analyzer could improve on the project.

Mahesh Menon was responsible for helping implement a crawler for the GitHub user dataset and static analyzing code samples.

This full report was written by Vanessa Barlow in fulfillment of a thesis requirement, with the exception of the GITADVCRAWLER section that was written by Kamya Mehul Desai. This technical paper will go into her final thesis portfolio. The research is still on-going with Faysal Hossain Shezan as the current team lead.

References

[1] GH Archive. GH Archive. https://www.gharchive.org/, 2020.

[2] GitHub. GitHub Api. https://developer.github.com/v3/, 2020.

[3] GitHub. Submitting content removal requests. https://docs.github.com/en/github/site-policy/submitting-content-removal-requests, Accessed: 2021-02-02.

[4] Gong, Q., Zhang, J., Chen, Y., Li, Q., Xiao, Y., Wang, X., and Hui, P. Detecting Malicious Accounts in Online Developer Communities Using Deep Learning. In *In The 28th ACM International Conference on Information and Knowledge Management (CIKM '19)* (Beijing, China, November 2019).

[5] Google. Google Bigquery. https://cloud.google.com/bigquery, 2020.

[6] Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. Character-Aware Neural Language Models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)* (Phoenix, Arizona, USA, February 2016).

[7] Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., and Chen, Z. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities . *arXiv preprint arXiv:1807.06756* (2018).

[8] National Vulnerability Database . NVD. https://nvd.nist.gov/, 2018.

[9] Rebecca L. Russell, Louis Kim, L. H. H. T. L. J. A. H. O. O. P. M. E. M. W. M. Automated vulnerability detection in source code using deep representation learning. *arXiv preprint arXiv:1807.04320* (2018).

[10] Software Assurance Reference Dataset. SARD. https://samate.nist.gov/SRD/index.php, 2018.

[11] VirusTotal. VirusTotal. https://www.virustotal.com/gui/home/upload, 2020.

[12] Zhang, X., Zhao, J., and LeCun, Y. Character-level Convolutional Networks for Text Classification. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)* (Montreal, Quebec, Canada, December 2015).

[13] Zhen Li, Deqing Zou, S. X. X. O. H. J. S. W. Z. D., and Zhong, Y. Vuldeepecker: A deep learning-based system for vulnerability detection. *arXiv: Cryptography and Security* (2018).