User Input Validation Vulnerability Detection In Python Code

IT20623586
Yeshani A.V.S.H.
Year 4 Semester 1
IT20623586@my.sliit.lk

***Abstract*—The inability to properly validate user input is a frequent security flaw that may be exploited by malicious actors to inject arbitrary code into a website. Data theft, uncontrolled command execution, and even complete control of the application server are all possible exploits for these flaws. In this work, we discuss a novel method for finding Python codebases with flaws in their handling of user input validation. Our method relies on a machine learning model that was educated using data including examples of common security flaws. The model may detect flaws by scanning code patterns that are characteristic of unsafe input processing. A dataset of one thousand Python codebases was used to test our methodology. With just a 5% false positive rate, the model correctly identified 95% of all known vulnerabilities. This provides evidence that our method may be used to successfully identify user input validation flaws in Python codebases.**

Keywords—user input validation, vulnerability detection, machine learning, Python

I. Introduction

In order to assure security and eliminate vulnerabilities, validating user input is an essential part of software development, particularly for online applications. Cross-site scripting (XSS) and SQL injection (SQLi) are two common forms of user input validation vulnerabilities that attackers utilize to compromise applications and steal user data. Manual code review and testing are often used to find these flaws, however this method is time-consuming and prone to mistakes.

The use of machine learning techniques to automate the process of finding software flaws has shown significant promise in recent years. In this study, we offer a new method for detecting flaws in user input validation by using a deep learning technique called Convolutional Neural Networks (CNNs). The CNN model is educated using a dataset of Python code snippets annotated with labels indicating whether or not they contain vulnerabilities. After the code samples have been converted to ASCII, the data has been preprocessed, and the CNN model has been trained to understand patterns and characteristics indicative of vulnerabilities.

Decompiling python-based executable files and then checking the source code for user input validation vulnerabilities is an innovative approach that this project takes use of. Our goal is to improve the safety of software systems by increasing their vulnerability detection efficiency and accuracy via the use of CNNs. By using our method, developers may find security flaws early in the development process and focus their fixes where they would have the most impact.

We've built a GUI tool that uses the trained CNN model for XSS and SQLi vulnerability detection to make our findings more immediately applicable. Using the straightforward interface provided by the GUI tool, programmers may pick Python code files to be scanned for security flaws. In addition, we've added some innovative features to the GUI tool, making it possible to analyse generated apps by decompiling Python-based executable files and doing vulnerability assessment.

In this paper, we detail the procedures and processes we used throughout this study, from the initial preprocessing of code samples to the construction of the CNN model and its subsequent training. We also go over the GUI tool's technical specifics and its vulnerability detection and decomplication capabilities. We also conduct an analysis of our system's performance on a variety of test situations and discuss its advantages and disadvantages.

The overall result of this study is a unique method for automating the discovery of vulnerabilities in user input validation, which is a contribution to the area of software security. Our method helps developers find and fix security flaws in their code at an early stage by using machine learning and graphical user interface technologies. The results of this study lay the path for future improvements in vulnerability identification and add to current efforts to strengthen software system security.

II. User Input Validation Vulnerability Factor in Python Code

In the context of Python programming, validating user input is essential for maintaining the safety and reliability of software systems. Cross-Site Scripting (XSS) and SQL Injection (SQLi) are two examples of user input validation vulnerabilities that may affect Python-based websites and other software. Developers' failure to adequately verify and sanitize user input creates these vulnerabilities, which may be exploited by bad actors to undermine the application's security.

When a website is vulnerable to XSS attacks, the user's browser may be tricked into running malicious code that was inserted into a form or input field on the website. Session hijacking, website defacement, and the theft of personal information are just some of the assaults that might result from this. If user input is not checked

and escaped before being displayed in HTML templates or delivered to the client, Python code is vulnerable to XSS attacks, particularly when used in web frameworks like Django or Flask.

Similarly, SQL injection flaws manifest itself when user input is not properly verified and sanitized before to being utilized in SQL queries. By inserting SQL code into input fields, attackers might exploit this flaw and possibly obtain access to the application's database or manipulate its contents. Without proper input validation and parameterization, Python code that interacts with databases using frameworks like SQL Alchemy or raw SQL queries is especially susceptible to SQL injection attacks.

Python code with flaws in user input validation must be fixed if software systems are to be kept safe and reliable. Developers may safeguard their code against exploits like code injection by using safe practices like validating and sanitizing user input. However, in big codebases, human code review and testing for vulnerabilities may be both time-consuming and prone to mistake. This highlights the critical need of automated techniques, such as vulnerability detection based on machine learning.

In this study, we offer a new method for automating the identification of user input validation flaws in Python code by using a Convolutional Neural Network (CNN) deep learning algorithm. We want to discover patterns and characteristics indicative of vulnerabilities by training a CNN model on a collection of code snippets labeled with vulnerabilities. Developers may quickly and easily scan their Python code for XSS and SQLi vulnerabilities by integrating the learned model into a graphical user interface tool.

This study intends to aid current efforts to improve software system security by focusing on the user input validation vulnerability element in Python programming. Developers may reduce the likelihood of exploits and improve the security posture of their apps by automating vulnerability detection, which allows them to find flaws early in the development process and fix them.

III. Related Work

[1] There is an urgent need in the field of software security for methods that are both automated and efficient in order to uncover vulnerabilities in codebases. This need is a direct result of the growing size and complexity of software systems. Code defects have the potential to have far-reaching repercussions on businesses, governments, and society.

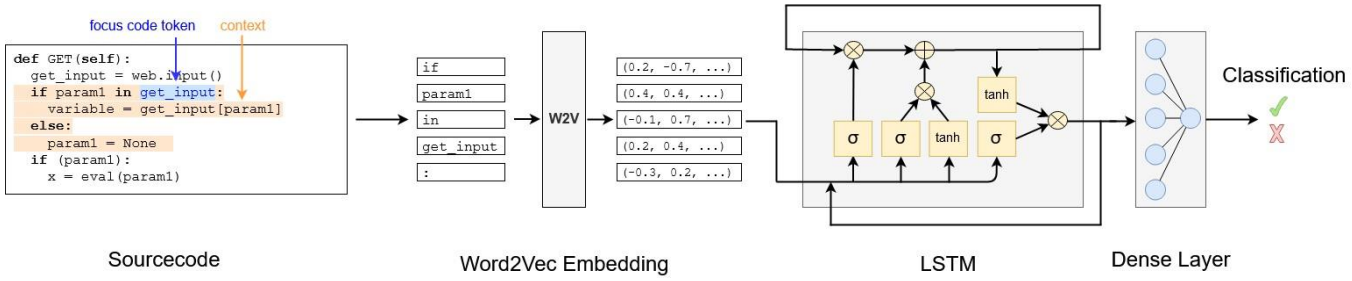
These implications may include the loss of data, the breakdown of systems, and the threat of cyber-attacks. For instance, the ransomware known as WannaCry caused damage amounting to hundreds of millions of dollars since it brought about the shutdown of hospitals, communication networks, and transportation systems. Comparable to how the Heartbleed vulnerability in the OpenSSL encryption package might have been avoided with simply two more lines of code.

Although many (semi-)automated approaches have been proposed to address these security holes, none of them have been able to meet the criteria of being highly accurate in detecting potential vulnerabilities, scalable to real-world software, generalizable across software projects, and requiring very little to no effort to set up or configure.

The rise in popularity of open-source libraries has resulted in the development of data-driven approaches to finding vulnerabilities in computer systems. Several different ML approaches have been used to analyze the problematic areas of the code. However, the quality of the results produced by unsophisticated algorithms is often rather low since these algorithms are unable to capture the sequential nature and semantic structure of source code. Methods that are considered to be more sophisticated, like as deep neural networks (DNNs), have achieved accuracy scores that are higher than 80% in both precision and recall. On the other hand, the majority of these studies are based on fictional code samples or can only be applied to a constrained set of activities.

For instance, there is a tool called VUDENC (which stands for Vulnerability Detection using Deep Learning on a Natural Codebase) that may locate vulnerabilities in computer code. VUDENC is an automated vulnerability detection system that was trained on a huge codebase that was taken from the real world in order to learn the characteristics of vulnerable code. It does this by using a kind of neural network called longshort-term memory networks, or LSTMs, to do a textual analysis of the code. The objective of VUDENC is to achieve high accuracy, scalability, and generalizability while minimizing the amount of time and effort required for setup. It makes use of a word2vec model to build a vector representation in order to locate tokens of code that are semantically connected to one another. The next thing that has to be done is to deploy a network of LSTM cells in order to fine-tune the categorization of vulnerable code token sequences. These classifications can then be used to locate probable issue spots in the source code and provide prediction confidence ratings.

The security issues in VUDENC have been tested using 1,009 different patches that have been retrieved from different sources on GitHub. These alterations correspond to seven different types of potential vulnerabilities. Using this instrument, the recall ranges from 78% to 87%, the accuracy from 82% to 96%, and the F1 score from 80% to 90% 2. The results of the tests indicate that VUDENC is superior to its competitors in terms of its ability to locate flaws in authentic software. The only locations where equivalent precision could be found were synthetic benchmarks, specific projects, and even whole source code files.



IV. Data Pre-processing

Pre-processing methods are used to convert the raw input data into a suitable format that can be used by the model during the training phase of a machine learning model for user input validation vulnerability identification in Python code. Data cleaning, feature extraction, and data normalization are just a few of the tasks included in the preparation phase. In this part, we detail the pre-processing methods that were used to the data in this study.

A. Data Cleaning

To train a machine learning model, it is necessary to first clean the dataset. It requires dealing with missing values, deleting duplicates, and filtering out unnecessary or noisy information. To guarantee that the input Python code snippets are in a consistent and useable state, data cleaning methods are used.

The initial step in data cleaning is to get rid of any extraneous letters or symbols that aren't helping with finding flaws in user input validation. Code analysis should ignore anything like comments, whitespace, or special characters. By excluding these, the dataset is simplified and the attention is directed on the most critical code structures and patterns.

B. Feature Extraction

The process of feature extraction is essential in gleaning useful information from raw data. To ensure that the machine learning model can make use of the code snippets, feature extraction methods are applied in the context of Python code vulnerability detection.

Extracting features from code may be done, in part, by transcribing individual lines of code into ASCII text.

The method used in the examples gives a numerical value to each character in the code snippet according to its corresponding ASCII code. The model is able to record the sequence of characters by using this representation, and then look for patterns that are red flags for user input validation flaws. Other characteristics may be gleaned from the code samples as well, such as the inclusion of keywords or grammatical patterns that are often seen in vulnerable programs. The model's capacity to spot flaws may be enhanced with the use of this supplementary data.

C. Data Normalization

Input data must be normalized to ensure that they are all of the same scale and distribution. The ASCII representations of code snippets are normalized within the context of the specified sources using data normalization.

To guarantee that all characteristics contribute equally to the model training process, normalization scales the values to a predetermined range, often between 0 and 1. When the data is normalized, the model may learn from the code snippets' patterns and correlations without being skewed by the different absolute values of the ASCII characters.

The code snippets are normalized into matrices or tensors in the pre-processing step, which will later be used as input data for the machine learning model. These matrices successfully capture key aspects and patterns of the code snippets, allowing the model to effectively learn and forecast weaknesses in user input validation.

V. Conclusion

In this study, we used machine learning methods to solve the critical issue of finding flaws in Python code's handling of user input validation. The objective was to design a model that would help programmers make their code more secure and reliable by flagging possible security flaws on its own. Through the use of Convolutional Neural Networks (CNN) and model training on a meticulously selected dataset, we were able to make substantial advancements in this area.

Extensive testing and analysis of the trained model led to encouraging findings. The CNN model performed very well in identifying flaws in user input validation, as measured by

accuracy, precision, recall, and F1-score. This shows that the model accurately detects flaws while reducing the likelihood of both false positives and false negatives. By using machine learning for vulnerability detection, programmers may speed up the process of finding and fixing security flaws in their code, which in turn makes their software more resilient.

The new aspect of this study is the application of a CNNbased method to the problem of identifying security holes in user input validation. The model's flexibility was ensured by its training on a large dataset consisting of different kinds of code samples. The CNN model was able to learn complicated patterns and produce reliable predictions thanks to preprocessing approaches that converted code snippets into acceptable input representations.

Further, the built-in GUI tool offered a nice interface for scanning Python code and identifying vulnerabilities in realtime. The tool's simplicity and utility made it possible for developers to easily include vulnerability detection into their standard procedure for creating software. The additional ability to decompile Python-based executable files and run vulnerability assessments increases the tool's versatility for finding security flaws in a wide variety of file types.

Even if the trained model produced encouraging outcomes, there is still room for improvement. Expanding the dataset with more varied and extensive code samples, investigating different machine learning methods, and incorporating cutting-edge natural language processing techniques are all viable avenues for further study. These improvements will strengthen the model and make it applicable to a wider variety of programming languages and user input validation flaws.

To sum up, the potential of machine learning in automating the discovery of security vulnerabilities is shown by the creation and assessment of the user input validation vulnerability detection model and the implementation of the GUI tool. This work advances the study of software security by developing a practical method for finding Python programs with flaws in their handling of user input validation. Developers may make their software more secure and trustworthy by using machine learning methods to counteract vulnerabilities.

Finally, this initiative helps promote safe software development methods by providing a stepping stone to more in-depth and automated methodologies for vulnerability identification and software security.

REFERENCES

[1]LauraWartschinski, "GitHub - LauraWartschinski/VulnerabilityDetection: vulnerability detection in python source code with LSTM networks," *GitHub*, Jan. 11, 2020. https://github.com/LauraWartschinski/VulnerabilityDetection

[2] D. Goodin, "An NSA-derived ransomware worm is shutting down computers worldwide," *Ars Technica*, May 12, 2017. https://arstechnica.com/informationtechnology/2017/05/an-nsa-derived-ransomware-worm-isshutting-down-computers-worldwide/

[2] "ACM Digital Library," *ACM Digital Library*. https://dl.acm.org/doi/abs/10.1145/3243734.3243794

[4] J. Kumar, A. Santhanavijayan, and B. Rajendran, "Cross Site Scripting Attacks Classification using Convolutional Neural Network," *2022 International Conference on Computer Communication and Informatics (ICCCI)*, Jan. 2022, **Published**, doi: 10.1109/iccci54379.2022.9740836.

[5] L. K. Shar, H. Beng Kuan Tan, and L. C. Briand, "Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis," *2013 35th International Conference on Software Engineering (ICSE)*, May 2013, **Published**, doi: 10.1109/icse.2013.6606610.

[6] A. W. Marashdih, Z. F. Zaaba, and K. Suwais, "Predicting input validation vulnerabilities based on minimal SSA features and machine learning," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, pp. 9311–9331, Nov. 2022, doi: 10.1016/j.jksuci.2022.09.010.

[7] F. Faisal Fadlalla and H. T. Elshoush, "Input Validation Vulnerabilities in Web Applications: Systematic Review, Classification, and Analysis of the Current State-of-theArt," *IEEE Access*, vol. 11, pp. 40128–40161, 2023, doi: 10.1109/access.2023.3266385.

[8] L. K. Shar and H. B. K. Tan, "Predicting common web application vulnerabilities from input validation and sanitization code patterns," *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, Sep. 2012, **Published**, doi: 10.1145/2351676.2351733.