International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 03 | March - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Security Mechanisms and Vulnerability Detection in Python Packages: A Comprehensive Review

[1]Shaista Fatma , [2]Ariba Kausar, [3]Anu Priya, [4*]Md. Irfan Alam

[1]Student, Department of CSE, Amity University, Ranchi Jharkhand, India

[2] Student, Department of CSE, Ramgarh Engineering College, Ramgarh, Jharkhand, India

[3]Assistant Professor, Amity Institute of Information Technology, Amity University, Patna, Bihar, India

[4]Associate Professor, Department of CSE & IT, Jharkhand Rai University, Ranchi, Jharkhand, India

*Corresponding author e-mail: irfan2.alam2@gmail.com

Abstract

Because of the language's simplicity, flexibility and massive libraries supporting artificial intelligence, web development and data science python has become one of the widely used programming language. But this speedy growth has led to software security and vulnerability concerns. This paper examines the security provisions available in python programming and their role in developing secure applications. Herein the analysis is done based on security mechanisms such as validation of inputs, authentication and authorization, libraries available for cryptography, management of dependency tools, static code analysis tools and secure web frameworks.

The paper highlights the common security challenges faced in python programming and emphasis on the importance of secure coding practices and comparison of vulnerability detection tools implemented in the past research works. The finding proclaims that python provides several built-in tools and mechanisms which when properly implemented can notably improve reliability and security of software systems.

Keywords: Python Security, Secure Coding Practices, Vulnerability Detection, Machine Learning Based Security Analysis, Static Code Analysis, Cryptography in Python, Software Security

1. Introduction

Python is an interpreted, high-level programming language, and is readable, simple. It was written by Guido van Rossum and it was first introduced in 1991. Since its inception, the language had a great influence and has become a staple in the software industry as well as academics [3].

One of the reasons why Python is widespread is its syntax. It allows programmers to write complex applications with fewer lines of code than most other languages. Moreover, the flexibility of Python can be explained by the fact that it supports the number of programming paradigms, which increases its popularity [4] such as:

i. Artificial Intelligence and Machine Learning: Python has emerged as the language of choice in the creation of artificial intelligence algorithms owing to availability of robust library supporting such artificial intelligence algorithms such as tensorflow, PyTorch, and Scikit-learn[8].
ii. Web Development: Frameworks like Django and Flask are used for the development of web applications[2,3].
iii. Data Science and Data Analytics: Python offers support libraries, such as NumPy, Pandas, and Matplotlib, to handle data processing and analysis [11,12].

On comparison of Python with different programming languages, the study shows that python has one of the least percentage of vulnerability exposure [15].

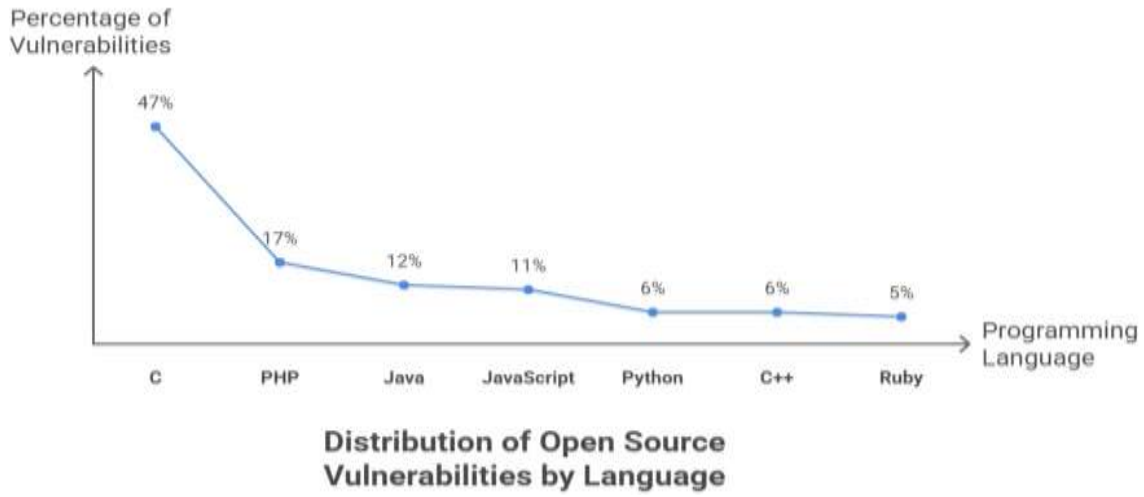

Figure 1: Python vulnerability comparison with other programming languages

Nonetheless, the popularity of Python as an application development framework has driven a demand to add security capabilities in order to secure the software systems against any vulnerabilities or attacks.

1.1 Importance of Security in Programming

Security of software is also an essential issue in the modern computer world. This is attributed to the fact that the application programs have been known to deal with highly sensitive information such as financial information, personal information and secret company information. Therefore, any vulnerability of the software may have dire consequences, such as security breach and financial losses.

Recent studies have indicated that the issue of vulnerability of software is still one of the biggest problems of the modern world of programming. The reason is that security vulnerabilities have been identified in the application programs as a result of programming bugs and use of external libraries in an abusive way [1]. Indicatively, anin-depthresearch has shown that the Python application programs that are designed to be accessible through Python Package Index (PyPI) repository have security vulnerabilities [12]. Moreover, the rise of the number of cyberattacks has underscored the need to put security principles as part of the application programs. This is due to the fact that attackers have exploited the vulnerabilities of the application programs such as the misuse of the application programs and failure to validate application programs.

Researchers have also focused on the potential of the automated approach to identify the possible security vulnerabilities in the application programs, such as the use of machine learning and deep learning methods of identifying potential security vulnerabilities in the source code of the application programs with better effectiveness [6,14]. Therefore, the security concepts within the application programs have formed a significant aspect in the application program development process to make the application programs secure and reliable.

1.2 Security Challenges in Python Applications

Although the Python language offers a number of advantages that could be achieved through the use of the language, various security issues arise when creating an application using the language. These issues can arise in case appropriate security control is not established.

The significant security issues that are likely to arise during development of applications in terms of the language are:

(i) Dynamic Typing and Runtime Behavior

Python is dynamically typed languages that allows the determination of variable types during runtime rather than compile time. This provides benefits to the developer because the language can be more flexible and can allow faster development. However, runtime errors may emerge when using the language, which may lead to potential security vulnerabilities if not properly addressed.

A study on vulnerability detection has indicated that dynamically typed languages may pose challenges to static determination of vulnerabilities because several behaviors may occur during runtime rather than compile time [6].

(ii) Heavy Dependency on Third Party Libraries

Applications of Python heavily rely on third party libraries that can be obtained from several repositories like the Python Package Index (PyPI). These libraries may provide benefits to the developers because they may allow faster development of applications compared to building applications from scratch. However, several security vulnerabilities may emerge when using the language because several security analyses of Python packages indicated that several open-source packages may contain security vulnerabilities like unsafe coding practices and dependencies [1, 12].

(iii) Code Injection Vulnerabilities

In Python application systems, the possibility of a code injection attack occurs due to insufficient validation of user input. Functions like eval or exec or insecure commandline execution can be used by hackers to inject their own code into the system. Injection attacks occur in the following forms:

a. SQL injection
b. Command injection
c. Code injection with dynamic execution

A research paper on the detection of vulnerabilities in programming languages revealed that injection attacks are the most common type of security risk in software systems [10].

(iv) Insecure Handling of User Input

A considerable amount of security risks occurs due to insufficient input validation.In case the input received by the user is directly entered into the system to execute system commands or perform database queries, the hacker can modify the input in a bid to introduce a security threat. Secure programming rules suggest that validation of input should be performed in order to prevent security threats.

(v) Dependency and Package Management Risks

In Python application systems, many packages are used to develop the application. These packages may contain security risks. If a package is outdated or vulnerable to a security risk, the entire application system can be at risk. A research paper on the Python package ecosystem revealed that many software systems can be vulnerable to security risks due to package dependencies [12].

(vi) Difficulty in Detecting Vulnerabilities in Large Codebases

Python application systems have a growing size of the application along with the code. The application system size makes it hard to detect the security risks or vulnerability in the Python application system. Python application systems of largescale size would not be efficiently detected using manual methods of detecting security risks or vulnerabilities. A study paper about the vulnerability detection in Python application systems indicated the use of machine learning and deep learning to identify vulnerabilities in Python application systems [6, 14].

(vii) Misconfiguration and Deployment Risks

Security risks in Python application systems are as a result of misconfiguration or insecure installation. Most of the causes of misconfiguration include:

a. Exposure of sensitive information
b. Insecure APIs
c. Weak authentication mechanisms

These misconfigurations can create a security risk for the system[1].

2. Literature Review

The rapid growth of Python as a programming language has led to increased research on software security, vulnerability detection, and secures coding practices. Due to the popularity of Python as a web application, artificial intelligence, and data processing system, the security of Python-based applications is becoming a significant field of research. Even the domain of vulnerability detection, vulnerability packages, vulnerability dataset creation and cryptographic security mechanisms on Python have been considered by the researchers. The following section is the

review of the key research works associated with the topic of Python security and the limitation of the current research.

In order to comprehend the current research environment in Python security, a number of research articles were examined. The table below presents the summary of the reviewed studies and their major focus and contributions.

Paper Title	Main Focus	Methodology	Key Contribution
A Large-Scale Security-Oriented Static Analysis of Python Packages [1]	Security vulnerabilities in Python packages	Static analysis of PyPI packages	Identifies common security weaknesses in Python libraries
VUDENC: Vulnerability Detection with Deep Learning on Python Code [2]	Automated vulnerability detection	Deep learning model for source code	Detects vulnerabilities in Python programs using neural networks
An Empirical Study of Vulnerability Handling in CPython [3]	Vulnerability management	Empirical analysis of vulnerability reports	Analyzes how security issues are identified and fixed
Exploring Security Commits in Python Projects [5]	Security fixes in Python repositories	Data mining of GitHub commits	Studies patterns of vulnerability patches
PyBugHive: A Comprehensive Database of Reproducible Python Bugs [5]	Python vulnerability dataset	Dataset creation and validation	Provides reproducible bug dataset for research
LLM-Based Command Injection Vulnerability Detection [6]	AI-based vulnerability detection	Large Language Models	Detects command injection vulnerabilities in Python
Adversarial Evaluation of ML-Based Vulnerability Detectors [7]	Testing ML security tools	Adversarial analysis	Identifies weaknesses in ML-based detection systems
Secure Coding with AI: Detection and Repair [9]	AI-assisted security	AI-based vulnerability detection and repair	Shows automated vulnerability fixing techniques
Conformer-Based Python Vulnerability Detection [9]	Deep learning vulnerability detection	Conformer neural architecture	Improves vulnerability detection accuracy
Deep Recurrent Architectures for SQL Injection Detection [10]	SQL injection vulnerability detection	Recurrent neural networks	Detects injection vulnerabilities in Python applications
Machine Learning Techniques for Python Security [11]	Vulnerability detection	Machine learning models	Identifies insecure coding patterns
Empirical Study of Python Package Vulnerabilities [13]	Python ecosystem security	Empirical analysis	Highlights dependency-related risks
Hybrid Security Protocol Using Python [13]	Secure communication	Cryptographic protocol implementation	Demonstrates secure data transmission

Table 1: Summary of Reviewed Research Papers

A comparative analysis of the reviewed studies helps identify the datasets, models, and methodologies used by researchers for vulnerability detection and security analysis in Python programming.

Paper	Dataset Used	Model / Classifier	Methodology	Accuracy	Key Findings
VUDENC: Vulnerability Detection with Deep Learning on Python Code [2]	Python source code vulnerability dataset	Convolutional Neural Network (CNN)	Deeplearningbased source code analysis	~90%	Demonstrates deep learning can automatically detect vulnerabilities in Python code
DiverseVul: A New Vulnerable Source Code Dataset for Deep Learning Based Vulnerability Detection [14]	DiverseVul vulnerability dataset	Transformer-based deep learning model	Large-scale dataset training for vulnerability detection	~91%	Improves vulnerability detection using diverse vulnerability samples
Vulnerability Detection in Popular Programming Languages with Language Models [6]	Open-source Python repositories	Large Language Model (CodeBERT / GPT-based models)	AI-based vulnerability detection	~88–92%	Detects command injection vulnerabilities in Python programs
Conformer-Based Python Vulnerability Detection [9]	Python vulnerability dataset	Conformer Neural Network	Block-level code representation analysis	~92%	Achieves higher detection accuracy using advanced neural architecture
Deep Recurrent Architectures for SQL Injection Detection [10]	SQL injection vulnerability dataset	Long Short-Term Memory (LSTM) Network	Deep recurrent neural network vulnerability detection	~89%	Effectively detects injection vulnerabilities in Python programs
Machine Learning Techniques for Python Security [11]	Python source code dataset	Random Forest Classifier	Machine learning based vulnerability classification	~85%	Identifies insecure coding patterns in Python programs

Table 2: Comparative Analysis of previous research papers implementing Machine Learning and Deep Learning techniques for vulnerability detection

Despite the fact that the examined works discuss the specifics of vulnerability detection and Python security in an informative manner, a number of shortcomings still exist. The table below observes the significant limitations of the current research.

Paper	Main Contribution	Limitation
Static Analysis of Python Packages [1]	Identifies vulnerabilities in PyPI ecosystem	Focuses only on package vulnerabilities
VUDENC [2]	Deep learning vulnerability detection	Requires large datasets and computational resources
CPython Vulnerability Study [3]	Analyses vulnerability resolution process	Does not address prevention methods
Security Commits Study [4]	Studies security fixes in repositories	Limited analysis of secure coding practices
PyBugHive [5]	Provides dataset for vulnerability research	Does not discuss prevention techniques
LLM Vulnerability Detection [6]	Uses AI for vulnerability detection	Focuses on specific vulnerability types

ML-Based Detection Studies [11]	Automated detection of insecure code	Limited focus on built-in Python security mechanisms
Hybrid Security Protocol[13]	Demonstrates secure communication	Narrow focus on cryptography only

Table 3.: Limitations of Existing Research

Based on the review, it is possible to note that the majority of studies are devoted to vulnerability detection methods based on machine learning or deep learning models. Nonetheless, there has been little literature on the study of the default security features that are offered by the Python programming language, secure coding, cryptography, dependency management and security frameworks. This study will, therefore, be in an endeavour to offer a detailed discussion of security provisions in Python programming.

3. Methodology

This paper is qualitative research that will examine the security provisions offered in Python programming. The methodology aims at analysing the Python inbuilt security controls, analysing the available research papers and testing security tools applied to detect any vulnerability in Python applications. This methodology is intended to learn the way that Python helps in safe programming and how various tools and techniques can be used in enhancing the security of software.

3.1 Analysis of Python Documentation

The initial stage of this study was to analyse official Python documentation and developer resources in order to learn what security features Python programming language offers by default. The documents have been reviewed so as to study the modules and libraries that facilitate secure programming and those cryptographic modules like the hashlib, ssl modules and the cryptography library. Also, resources on popular Python frameworks like Django and Flask were consulted to gain knowledge of how these frameworks use security functions such as authentication systems, session management, and also secure against standard webvulnerabilities.

3.2 Study of Python Security Libraries

Here in this case the Python security libraries and tools have been learned which assists the developers to develop secure applications. Libraries like hashlib, cryptography, and ssl were studied to learn the ways they help to implement encryption, hash passwords and secure communication protocols. Evaluation of these libraries was based on their purpose of securing sensitive data, and unauthorized access as well as secure data transfer among systems.

3.3 Review of Existing Research Papers

The overall analysis of available research articles dealing with Python security and vulnerability detection was carried out. The studies were able to give details of typical security vulnerabilities in Python applications as well as the methods of identifying these vulnerabilities. The papers reviewed contain the works on vulnerability detection with the help of machine learning, Python package ecosystems analysis, security commit analysis, and vulnerability datasets, including PyBugHive. These papers have assisted in the identification of the available solutions, the general security issues as well as research gaps in the areaof Python security.

3.4 Evaluation of Security Tools

The last methodology stage entailed the review of the Python development tools of security testing and analysis. The Bandit, PyLint, and Flake8 tools were researched to learn how they can identify the insecure coding patterns and possible weaknesses in Python programs. These tools are used to perform a program analysis of the code through scanning rather than executing the code. They assist the developers to detect bugs, including non-secure use of functions, hard-coded secrets, unsafe file operation and inefficient management of errors. The assessment of these tools contributed to the achievement of the goal of determining their usefulness in enhancingsoftware security and supporting secure coding practices.

4. Security Provisions in Python Programming

Python supports security by a mix of secure programming best practices, in-built security measures, cryptography code libraries, dependency managers, and automated security vulnerability detectors. Since Python is popular in web applications, artificial intelligence systems and data processing platforms, it is necessary to make sure Python applications are secure to prevent attacks like code injection, data security breach, and hacking.Python's ecosystem has many mechanisms and tools that can be used to enhance the security of software and mitigate vulnerabilities[18,19,20].

4.1 Input Validation and Sanitization

Basic security measures to make sure that malicious or unexpected input will not influence program behaviour are input validation and sanitization. A large number of software vulnerabilities occur when user input is being processed without any validation of the input. Such weaknesses can be used by the attackers to carry out attacks of the form of SQL injection, command injection, and code injection.Input validation makes sure that the user inputs are in the expected formats, types and values before the application processes the inputs. Developers often use techniques such as:

i. Validation rules for input to restrict acceptable data formats

ii. Quoting commands to avoid the implementation of malicious commands.

iii. Securing database operations by use of parameterized queries.

The vulnerability detection in programming language research has indicated that injection vulnerability is one of the most prevalent security vulnerability in any software system and therefore, careful input processing in Python programs is important [1].

4.2 Authentication and Authorization

Authentication and authorization are the mechanisms that are employed to ensure that only the authorized users can be able to access some of the resources in an application. It provides a validity check on who a user is and also controls the kinds of activities the authenticated user is entitled to do.

The existing python systems include embedded authentication systems and access control measures as in the Django. Using these systems, the developers are able to use role-based access control (RBAC) whereby different users are provided with different permissions based on their roles (i.e. administrator, moderator, regular user). Appropriate authentication and authorization systems are to be installed to guide against the unauthorized access and privilege elevation attacks.

4.3 Cryptography libraries in Python

Python is also very supportive when it comes to cryptographic applications because it has in-built modules and external libraries that enable developers to establish viable data protection measures. Cryptographic techniques can be properly used to safeguard sensitive data and integrity of data in Python applications [4].

Common cryptographic libraries include:

i. Hashlib - It is applicable in creation of safe hash values like SHA-256 to use in password hashing.

ii. Cryptography - It is ageneralized encryption and decryption library and key management library.

iii. SSL - It offers secure communication with encrypted network connections.

4.4 Secure Package and Dependency Management

Python programs have a high number of external libraries and packages that are usually consumed. In as much as these dependencies are very efficient in terms of development, they may end up posing security vulnerabilities. Studies performed on Python package repositories have shown that there are some open-source libraries with security vulnerabilities or old dependencies that can pose applications to attacks unless addressed properly [2,16, 17].

In order to deal with those risks, there are a number of tools that could be used in order to manage dependencies securely:

i.pip - It is the package manager of Python that is utilized to install and manage libraries.
ii.pip-audit – It searches installed packages for known security vulnerabilities.
iii.Safety – It analyses Python dependencies to vulnerability databases.

These tools can be used to detect insecure or old packages and upgrade them to newer and safer ones by the developers. Effective dependency management mitigates the threat of attack on the supply chain and exploitable software elements.

4.5 Error Handling and Exception Management

The other applicable aspect of secure programming is the error handling. In case of error messages that display too much knowledge of the system, it can be utilized by the attacker to exploit vulnerabilities.

Python provides structured exception handling through try-except blocks, which allow developers to safely manage runtime errors without revealing sensitive information, such as:

The logging must be secure, exceptions have to be handled, and stack traces are not to be made so that the behaviour of the system could be kept track of and that the potential security threats can be found, but the secret information is not to be provided to the attackers.

4.6 Static Code Analysis Tools

Code analysis tools Static code analysis tools helped the developer to identify potential vulnerabilities in the source code that can result in security vulnerabilities in the software even before the software is available. They are the computer-assisted tools that explore the computer code to detect insecure coding conventions and programming errors.

In Python, some of the security analysis tools used include:

1. Bandit -It is used to scan Python source code to find security vulnerabilities.

2. PyLint - It is an anti-piracy that checks the quality of the code and determines possible problems with programming.

3. Flake8 - It is a style checker that identifies code errors.

The tools of the software analysis are especially useful in detecting such issues as:

i.insecure function usage
ii. hardcoded passwords
iii. unsafe file operations
iv.unsuitable exceptional handling.

The new study has examined automated methods of vulnerability detection based on machine learning models to enhance the accuracy of analysis of security flaws in the source code [3]. [3].

4.7 Secure Web Development Frameworks

Python web frameworks too extend some inbuilt security features to counter common web attacks against the applications.

Django: A framework contains various security measures, which are:

i.Cross-site Request Forgery (CSRF) security.
ii.ORM-based query SQL injection prevention.
iii.Cross-site Scripting (XSS) security.
iv.Secure password hashing

Flask: It is a lightweight web framework that provides flexibility while allowing developers to add security features through extensions. Flask applications can implement:

i. secure session management

ii. authentication systems

iii. security extensions such as Flask-Login and Flask-Security

Despite the fact that Flask offers less built-in defences than Django, its architecture can be scaled to security requirements, as developers can readily add security features.

In general, the vulnerability of Python web applications can be greatly minimized with secure frameworks and with the practice of recommended security guidelines.

5. Results

As demonstrated in the analysis made in this study, Python has a number of mechanisms that help ensure secure software development. Security measures like validation of input, data sanitization and proper error management are useful in avoiding all the typical vulnerabilities such as SQL and command injections. Python is also well supported with cryptographic functions including hashlib, cryptography and ssl which allow storage of passwords, encryption of data and secure transmission of information between systems.

The research paper also emphasizes the need to control the third-party dependencies since most Python applications are dependent on third-party libraries. Pip-audit and Safety are used to find vulnerable packages and minimize the risks of insecure dependencies. Moreover, the use of such tools as static analysis, such as Bandit, PyLint, and Flake8 help the developers to detect insecure coding patterns and the vulnerabilities that might occur in the development process. Also, the current Python platforms like Django or Flask have inbuilt security mechanisms that guard applications against various web attacks like cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL. Generally, the findings demonstrate that Python has an entire range of security facilities which when applied in the right way can greatly enhance the security of Python-based applications.

6. Conclusion

In this study, python's security provisions availability and importance of developing secure software application was analyzed. The results highlights that there are several built in security management tools dependency tools, cryptographic libraries and secure coding packages provided by python. Frameworks such as Django and Flask has additional protection feature against web vulnerabilities. The security issues arise due to improper coding practices and use of third-party libraries. So, the developer must follow secure programming practices and use the security tools available to reduce these vulnerabilities and improve overall security of python applications.

References

[1] Ruohonen, J., Hjerppe, K., &Rindell, K. (2021). Large-scale security-oriented static analysis of Python packages. Empirical Software Engineering. https://doi.org/10.1007/s10664-021-09996-y

[2] Wartschinski, L., et al. (2022). VUDENC: Vulnerability detection with deep learning on Python code. arXiv.

[3] Ruohonen, J. (2022). An empirical study of vulnerability handling in CPython. Software: Practice and Experience.

[4]Alam, M. I., & Singh, S. N. (2021). Designing and implementing cloud security using multi-layer DNA cryptography in Python. In M. Chakraborty, R. K. Jha, V. E. Balas, S. N. Sur, & D. Kandar (Eds.), *Trends in wireless communication and information security* (Lecture Notes in Electrical Engineering, Vol. 740). Springer. https://doi.org/10.1007/978-981-33-6393-9_38

[5] Kochhar, S., et al. (2020). Exploring security commits in Python projects. IEEE Software.

[6] Chen, X., et al. (n.d.). PyBugHive: A comprehensive database of reproducible Python bugs. In Proceedings of the IEEE/ACM Conference on Software Engineering.

[7] Al Atiiq, S., Gehrmann, C., & Dahlén, K. (2024). Vulnerability detection in popular programming languages with language models. arXiv. https://doi.org/10.48550/arXiv.2412.15905

[8]Bihari, S., & Alam, M. I. (2025). Leveraging recommender systems for course selection in higher education: A pathway to informed decision-making. In *Proceedings of the Recent Advances in Artificial Intelligence for Sustainable Development (RAISD 2025)*. Atlantis Press, Springer Nature. https://doi.org/10.2991/978-94-6463-787-8_27

[9] Hussain, A., et al. (n.d.). Adversarial evaluation of machine learning-based vulnerability detection systems. ACM Computing Surveys.

[10] Feng, P., et al. (n.d.). Secure coding with AI: Detection and repair. IEEE Security & Privacy.

[11] Li, H., et al. (n.d.). Conformer-based Python vulnerability detection. IEEE Access.

[12]Alam, I. (2020). Enhancing cloud security using multi-level DNA cryptography. *Splint International Journal of Professionals, 7*(1), 75–82.

[13] Zhou, Y., et al. (n.d.). Deep recurrent architectures for SQL injection detection. IEEE Transactions on Dependable and Secure Computing.

[14] Li, R., et al. (n.d.). Machine learning techniques for Python security. Journal of Information Security.

[15] Ruohonen, J., et al. (n.d.). Empirical study of Python package vulnerabilities. Empirical Software Engineering.

[16]Alam, M. I., & Priya, A. (2025). Computational approaches to revitalize Maithili literature: Bridging tradition and technology. In *Proceedings of the Recent Advances in Artificial Intelligence for Sustainable Development (RAISD 2025)* (Vol. 196, pp. 245–255). Atlantis Press, Springer Nature.

[17] Ahmed, M., et al. (n.d.). Hybrid security protocol using Python. International Journal of Computer Applications.

[18] Chen, Y., et al. (2023). DiverseVul: A new vulnerable source code dataset for deep learning-based vulnerability detection. arXiv. https://doi.org/10.48550/arXiv.2304.00409

[19] Mend.io. (n.d.). Most secure programming languages. https://www.mend.io/most-secure-programming-languages

[20]Phylum Research Team. (2023). ***Malicious Python packages in PyPI: Software supply chain security report.*** **Phylum Inc.**

.