World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:18, No:1, 2024

Static Analysis of Security Issues of the Python Packages Ecosystem

Adam Gorine, Faten Spondon

***Abstract*—**Python is considered the most popular programming language and offers its own ecosystem for archiving and maintaining open-source software packages. This system is called the Python Package Index (PyPI), the repository of this programming language. Unfortunately, one-third of these software packages have vulnerabilities that allow attackers to execute code automatically when a vulnerable or malicious package is installed. This paper contributes to large-scale empirical studies investigating security issues in the Python ecosystem by evaluating package vulnerabilities. These provide a series of implications that can help the security of software ecosystems by improving the process of discovering, fixing, and managing package vulnerabilities. The vulnerable dataset is generated using the NVD, the National Vulnerability Database, and the Snyk vulnerability dataset. In addition, we evaluated 807 vulnerability reports in the NVD and 3900 publicly known security vulnerabilities in Python Package Manager (Pip) from the Snyk database from 2002 to 2022. As a result, many Python vulnerabilities appear in high severity, followed by medium severity. The most problematic areas have been improper input validation and denial of service attacks. A hybrid scanning tool that combines the three scanners, Bandit, Snyk and Dlint, which provide a clear report of the code vulnerability, is also described.

***Keywords*—**Python vulnerabilities, Bandit, Snyk, Dlint, Python Package Index, ecosystem, static analysis, malicious attacks.

I. Introduction

PYTHON's popularity has skyrocketed in the last 16 years. Its programming methodology is to construct third-party extensions rather than build functionality by developers directly.

Python packages are becoming more prevalent and extensively utilised in applications, and most of the code generated and software applications used today rely on them. These packages are accessible through online repositories known as package managers. For example, NPM is the package management for Node.js apps, while PyPI is the package manager for Python projects. PyPI now has 419,968 packages uploaded, which is growing daily [1]. However, the simplicity of reusing third-party software comes with security vulnerabilities that endanger millions of users. These internal functionalities allow programmers a wide range of dynamic techniques, while attackers may easily download malicious code from the Internet and execute it instantly. Furthermore, it is challenging for existing Python system security to identify all possible security flaws and privacy threats in third-party extensions [2]. So, security vulnerabilities are one of the most critical challenges affecting these products.

This study discusses the security concerns in the Python ecosystem by examining the triviality of package reuse concerning publicly known security problems. This research paper includes 807 vulnerability reports in the NVD from 2002 to 2022, as well as 3900 publicly known security problems in Python Package Manager (Pip) from the Snyk database [3], [4]. The main conclusion is that the security flaws in Python package management are significant and could be misused. This study also presented an automated hybrid scanning tool that combines three scanners, Bandit, Snyk, and Dlint. It would function as a static analysis tool and deliver a clear vulnerabilities report.

II. Background

A. Python Package

A package is a folder that contains a collection of several modules. In a nutshell, many functions are saved in modules, which are put into packages. A package allows the structuring of modules in a more organised way, whereas a package is the same as a directory. Within the directory, sub-packages and modules can be created in a structured way, as shown in Fig. 1. Moreover, it will make the sub-packages and modules easy to access and understand. So, packages help keep other sub-packages and modules used by the other user when necessary [1].

The first step in creating a Python package is to write the __init__.py file. This file must be maintained in a package and signal to the Python interpreter that a package has been formed.

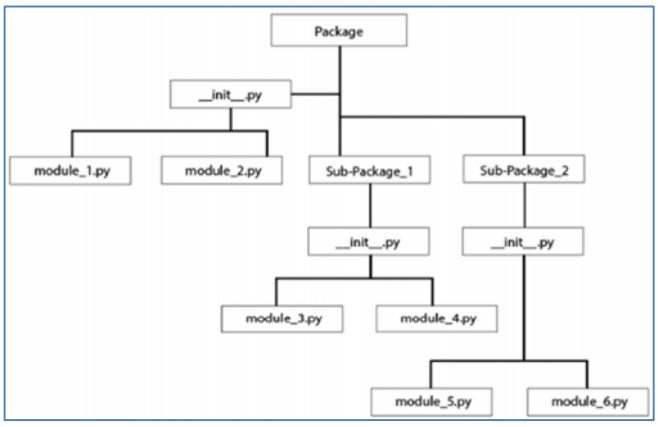

Fig. 1 Structure of a Python package

Dr Adam Gorine is a Senior Lecturer in Cyber Security at the University of the West of England, UWE Bristol, UK (corresponding author, e-mail: adam.gorine@uwe.ac.uk).

Faten Spondon is a Research Student at the University of the West of England, UWE Bristol, UK (email: faten.spondon@live.uwe.ac.uk).

B. National Vulnerability Database

The United States government developed the NVD to assist people and organisations in developing the automation of vulnerability assessments and other security mechanisms [3]. This database contains a list of vulnerabilities. The National Institute of Standards and Technology (NIST), which has sponsored NVD since 2015, researches Common Vulnerabilities and Exposures (CVEs). It defines a Common Vulnerability Scoring System (CVSS), which provides basic information about each vulnerability and updates the score as new data become available. Constant evaluation and analysis assist NVD users in assessing the severity of each vulnerability and prioritising risk management activities. For example, when a vulnerability in Python is discovered, MITRE assigns a unique CVE identification, a brief description, and external references.

C. Common Vulnerability Scoring System

CVSS is a general framework for rating software security vulnerabilities' severity. A CVSS score is composed of three metrics: Base, Temporal, and Environmental; each of which has an underlying scoring component.

- Base metrics are the fundamental characteristics of a vulnerability that do not change in time and across user environments [5].

The base metric produces a score ranging from 0 to 10, which can be modified by scoring the Temporal and Environmental metrics.

- Temporal metrics are metrics that change over the lifetime of a vulnerability. In addition, these metrics measure the current exploitability of the vulnerability.
- Environmental metrics represent vulnerability characteristics that are impacted by the user's environment. These are essentially modifiers to the Base metric group.

D. Common Weakness Enumeration

CWE is a community-maintained listing of various types of vulnerabilities in software and hardware published by the MITRE Corporation. The vulnerabilities are categorised and given uniquely identifiable IDs [6].

E. CVE of Heartbleed Vulnerability

The Heartbleed vulnerability (CVE-2014-0160) is a flaw in the implementation of the protocol used by the Secure Socket Layer (SSL) and Transport Layer Security (TLS) that update the connection between the Client and Server. Heartbleed vulnerability enables an attacker to retrieve data from the victim's computer memory without authorisation [7].

CVE-2014-0160 for Heartbleed Vulnerability is the 160th vulnerability listed in the NVD in 2014. It has a CVSS Base Score of 7.5 (High), as shown in Fig. 2.

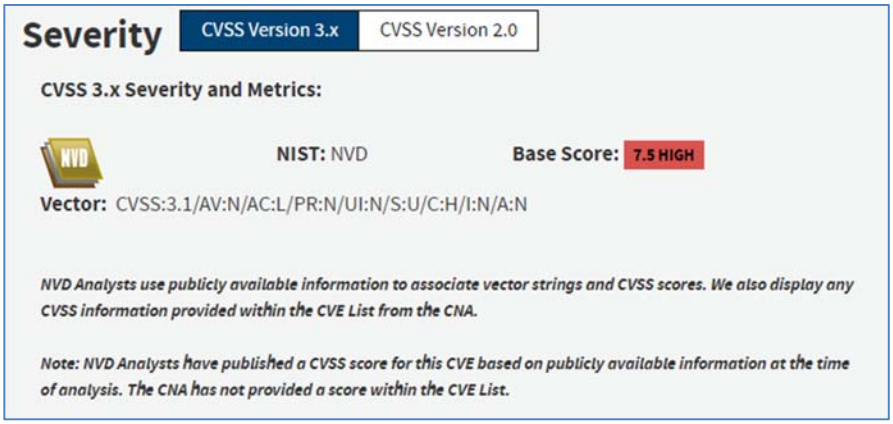

Fig. 2 Metric score of the Heartbleed vulnerability

III. Related Work

A. Static Analysis

This section concentrates on three significant areas of linked study, allowing us to define the scope of the work better.

Due to new vulnerabilities emerging daily, the analysis will change from past years based on these data. As a result, staying current is vital. Making a tool that can interpret current data can provide accurate and exact results.

Reference [8] collects data from PyPI.org, Package download statistics, and Safety DB. They provided a large-scale examination of the Python ecosystem based on PyPI. They demonstrated how easily the ecosystem might be attacked, analysed the impact of attacking a package, and explained how attackers deceive users into downloading harmful packages. First, they examine their data with Safety DB, where they can improve some of their functionalities.

- The open-source Safety DB is only updated once a month.
- The premium versions are expensive.
- The output does not show the severity of the vulnerability, simply the ID.

Their investigation comprises a limited set of data, which needs to be more comprehensive to analyse the entire ecosystem. Therefore, the analysis must obtain data from the sound source.

Reference [9] used NVD data to analyse third-party susceptible data in their study. The *completeness* of some project repositories is one of their dataset's limitations. There are old project versions that do not exist in the project's current repository. As a result, if the NVD contains entries for these

versions, the mapping process will fail.

Reference [10] talks about the security issues of the Python packages in detail. Descriptive package security concerns with severity and issue categories were acquired using the Bandit tool. Bandit is a well-known Python utility for detecting security flaws. However, it occasionally produces false positives and is unable to identify critical assaults like SQL injections, which can lead to inaccurate analysis.

Safety DB is also used in [11] to track vulnerabilities. They examined the common vulnerability exposures (CVE) listed in NVD and provided statistical results based on severity, CWE id, and publication year. However, as previously stated, the Safety DB requires specific enhancements, and a centralised tracking system for software vulnerabilities is needed.

In [2], researchers empirically assess security vulnerabilities in the PyPi ecosystem, examining approximately 500 package vulnerabilities affecting 252 packages. They investigated vulnerability propagation, discovery, and remedies in particular and compared the findings to the NPM ecosystem. Libraries.io and Snyk.io provided the dataset. Essentially, the emphasis is on the timeliness of vulnerable package discovery rather than discussing specifics on other sectors such as severity, CVE, CVE, or attack types in general.

Reference [12] investigates the similarities and differences between reported software vulnerabilities for interpreted programming languages. The CVSS and the CWE are used to compare vulnerabilities across four software repositories not only in PyPI but also in Maven, npm, and RubyGems based on a sample of vulnerabilities from each repository. They discussed the most common flaws across all four repositories: cross-site scripting and input validation. This research needs to clearly explain which repositories they used. Therefore, more study is needed to address this assumed restriction; one potential road forward would be to compare the results to different databases, such as the commercial Snyk database.

The mentioned study [12] lacks clarity in its data collection technique, impacting the reliability and reproducibility of findings. Our suggested solution would manually parse the necessary data using Python and JavaScript, providing a clear comprehension for statistical analysis. We can collect updated data using this strategy because new vulnerable packages are published every day, and predictions might alter from year to year. Selecting the correct database to collect all the information is also critical. Our study technique collects data from both the NVD and the Snyk Database.

B. Security Issues

Many assessments of software ecosystems are discussed in the literature; nonetheless, it is essential to concentrate on security-related concerns. While these efforts provide helpful information for interpreting these complicated objects, they fail to address the issue of attackers spreading malicious software by leveraging holes in the PyPI ecosystem.

The most critical part of the analysis is discussing security concerns after obtaining the vulnerability. The sorts of security issues, how they occur, and the mitigation will assist in preventing the installation of malicious packages in the future, or at the very least make you aware of it. Unfortunately, the preventative strategy needs to be included in these publications [13], [14], which presented a quick overview of typical problem kinds.

On both sides of the socio-technical research paradigm, either known vulnerabilities (based on the CVEs framework) or abstract weaknesses (based on the equivalent CWE framework) are employed [15], [16]. However, CVEs are more beneficial on the technical side of the study, whereas CWEs are more valuable on the social side [17]. CWEs, for example, have been used to improve understanding of common programming errors in Python packages, to offer dynamic information sources for software developers using static analysis tools, and so on.

Despite the benefits of such methodologies for systematic empirical research, the approach utilised in this work is based on the issue categories offered by the static analysis tool. Although the program can map concerns to CWEs [18], such mapping produces fewer fine-grained categories and must be done manually.

Our research outlines the most prevalent forms of attacks that can occur and the mitigation method. After analysing the data, we explained the most common forms of attacks and CWE for the vulnerability.

C. Scanning Tools

Static vulnerability detection techniques are often used in a programming language that does not run applications. The source code is the primary analysis object. It can also have deep-buried vulnerabilities, although it requires user assistance and has a significant false positive rate.

Reference [19] offered a Python security analysis framework. The suggested method is built into the Python interpreter. When users run a Python script, the interpreter automatically searches for the backup file and verifies the script's integrity. If the script's integrity is compromised, the bandit with a taint module detects the vulnerabilities. Experiment findings suggest that the Python security analysis framework performs well and that the bandit with taint can decrease false positives.

The research paper [18] used a qualitative approach to generate security smells from Python Gists and syntactic contexts to identify scent occurrences in the environment automatically. They utilised two static analysers to locate scent occurrences. Bandit is the first static analyser used for security smells in Python programs. Another is their tool which uses Python as a module to collect some occurrences while traversing the Python Gists abstract syntax tree. Some static analysis tools specialise in locating specific frameworks. For example, flask vulnerabilities can be found using static analysis tools such as PyT.

In our research paper, we created a static analyser by combining popular scanning technologies. These are Bandit, Snyk, and Dlint. The shell script will download the scanner, scan the code, and generate a report automatically. It will lower the number of false positives, make the script accessible to everyone, and work for all forms of attacks.

IV. Methodology

This section explains the methodology for collecting and preparing the data to achieve our objectives. It is divided into two parts.

A. Data Collection from NVD and Snyk.io

JSON is a text-based data storage format. In other words, data structures are used to represent objects as text. Despite being inherited from JavaScript, it has become the de facto standard for object transfer.

Most common programming languages, including Python, support this format. APIs most typically use JSON to communicate data objects. An example of a JSON string is as shown in Fig. 3.

Fig. 3 JSON Format

B. Report Generation from the Combined Scanner

Not all scanners provide an individually accurate report for every type of vulnerability. Some scanners have their own set of restrictions. For example, some scanners provide false positive results, while others cannot detect specific attacks. In addition, some scanners focus on packages, while others go through line-by-line execution.

Combining many scanners to obtain every possible result as a report will aid in detecting and preventing code security measures while also saving time. First, the shell script establishes dependencies and configures the combined scanner. Then, the project folder undergoes scanning, with the results saved as a text file and an HTML report generated, as depicted in Fig. 4.

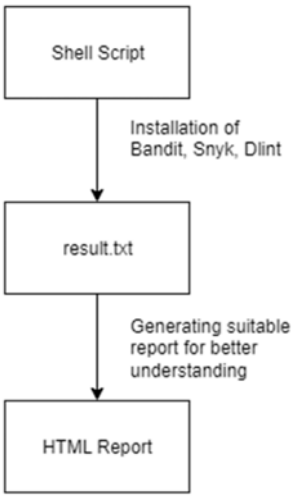

Fig. 4 Report generation from the combined scanner

V. Test and Results

A. Results from NVD

The NVD extraction involves Python vulnerability reports from 2002 to 2022, revealing approximately 807 vulnerabilities, as depicted in Fig. 5. Subsequently, the data review process commenced, leading to the confident assertion that the number of new exposures is increasing. However, the sheer quantity of vulnerabilities is not what we should be concerned about. If identifying a considerable number of potential attack pathways or low-impact exploits is possible, this might not be as frightening as it appears.

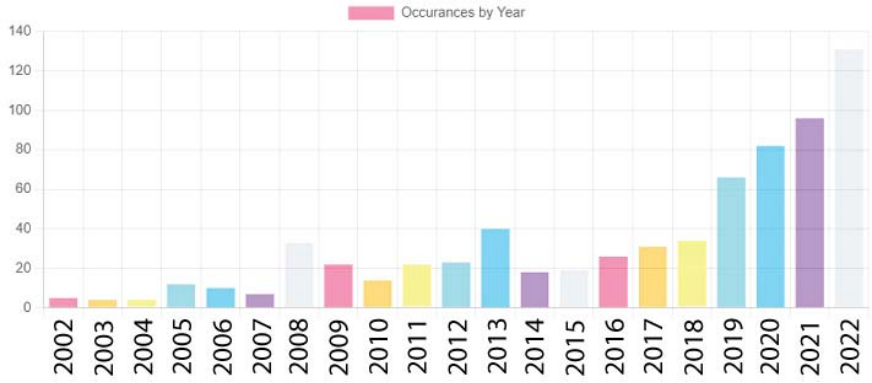

Fig. 5 Python vulnerability reports

1. CVSS Score

The data indicate a mix of version 3 and version 2, as shown in Fig. 6. There is a difference between the versions. But each version does not separately include the entire score from 2002 to 2022. After 2015, for example, there is no score for version 3, and the upgraded version needs scores for version 2. To obtain the whole result, the version 3 score was first prioritised. Those CVEs that do not have a version 3 score have their version 2 score taken. The vast majority are for version 6.

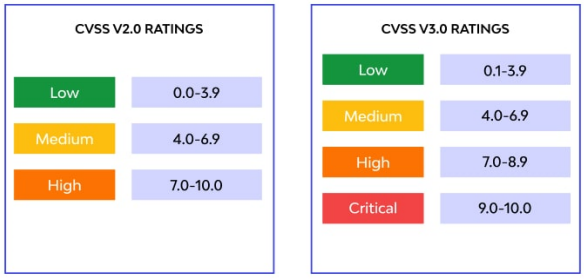

Fig. 6 A comparison of CVSS Scores v2 and v3

The occurrences of the base score are identified, and the resulting data are depicted in the pie chart in Fig. 7, which demonstrates the overall CVSS ratings and severity. Around 62.7% of the vulnerabilities are classified as high severity (CVSS >= 7), while 34.3% are classified as medium severity (CVSS 7 and CVSS >= 4). In a nutshell, 97% of publicly available exploits target medium or high-severity vulnerabilities.

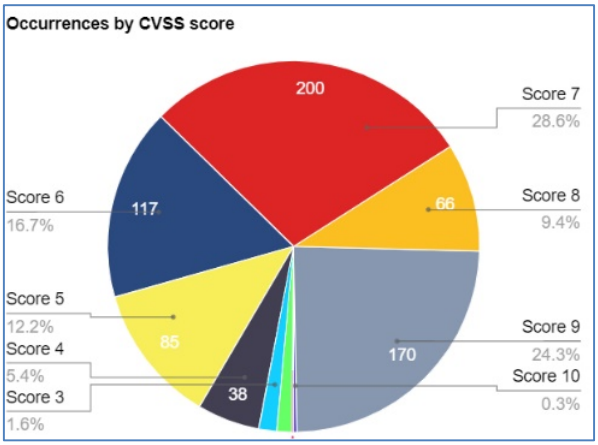

Fig. 7 The distribution of the base score occurrences

2. Common Weakness Enumeration (CWE)

After analysing the CWE data from NVD, it is possible to determine which CWE occurs most frequently.

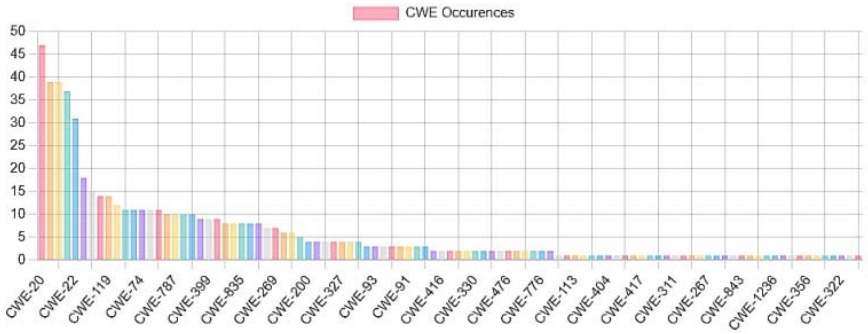

Fig. 8 CWE data from NVD

NVD implements CWE into CVE vulnerability scoring by giving a cross-section of the total CWE framework. CVEs are scored by NVD analysts using CWEs from multiple levels of the hierarchical structure, as shown in Fig. 8. Because of the varying levels of specificity offered by different CVEs, this cross-section of CWEs enables analysts to assess CVEs at both fine and coarse granularity. For example, Fig. 9 (a) represents CWE 2022's top 25 most dangerous software weaknesses [20] and Fig. 9 (b) illustrates the top five CWE that are frequently occurring.

B. Results from Snyk.io Database

The vulnerability reports available in the Snyk database have been taken, which is a platform for security. Python vulnerabilities are often published publicly after a more extended period. However, Snyk identifies many vulnerabilities before they are distributed in public databases.

Around 3900 vulnerability reports from Python's package manager, pip, are obtained. The Snyk report unveiled 1,391 distinct vulnerable packages spanning from 12 January 2010 to 25 November 2022. Notably, frequently encountered Python packages include TensorFlow, TensorFlow-CPU, TensorFlow-GPU, Django, Plone, Ansible, Pillow, Salt, Apache-Airflow, and RdiffWeb, as illustrated in Fig. 10.

TensorFlow is Google's open-source machine learning framework, which was created to make machine learning models simpler and easier to apply [21]. It was initially released 16 years ago. TensorFlow has a very active community, which means that many essential building pieces for getting started with data sets, models, and other things are readily available. TensorFlow has also been translated into other languages; Python being unquestionably the most popular. Its adaptable design enables simple computational deployment over various platforms, including CPUs and GPUs.

TensorFlow is an extensive framework that relies heavily on third-party libraries, which is why it creates significant vulnerability reports and ranks first in vulnerability reports. According to the statistics, TensorFlow-CPU, TensorFlow-GPU, and TensorFlow are the most vulnerable to Denial-of-Service attacks (DoS) [22]. These three occur most in vulnerability reports.

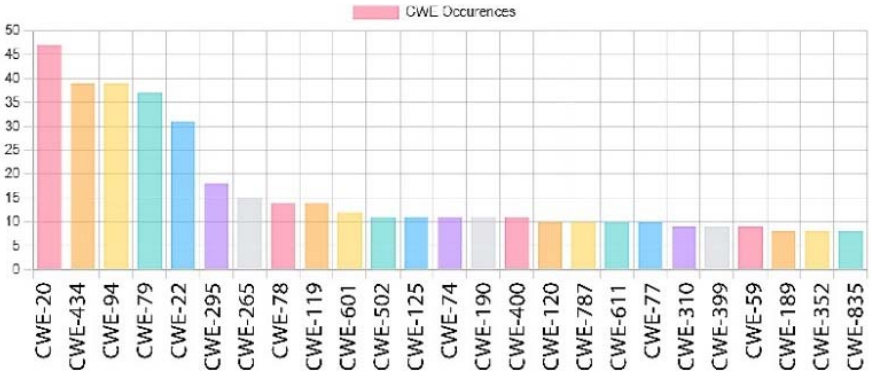

Fig. 9 (a) The most often occurring Python CWE

CWE-20	Improper Input Validation	The product accepts input or information, but it does not authenticate that the input has the attributes necessary to process the data securely and correctly or it verifies wrongly.
CWE-434	Unrestricted Upload of File with Dangerous Type	The program enables the attacker to upload or transfer malicious data. The file can be automatically executed within the package's ecosystem.
CWE-94	Improper Control of Generation of Code ('Code Injection')	The system will generate all or part of a code segment utilizing remotely driven inputs from an upstream component. It does not neutralize or neutralizes improperly special components that may affect the syntax or behaviour of the intended code segment.
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	This exploitation occurs when an attacker utilizes an online application to transfer malicious code. Once the malware has been inserted, the attacker is free to carry out a range of malicious behaviour.
CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	An attacker can use a path traversal exploitation to get access to data on your web application that they should not have. They achieve this by deceiving the web server or application that is executing on it into sending data that which are not in the web root directory.

Fig. 9 (b) The top five CWEs that are occurring frequently

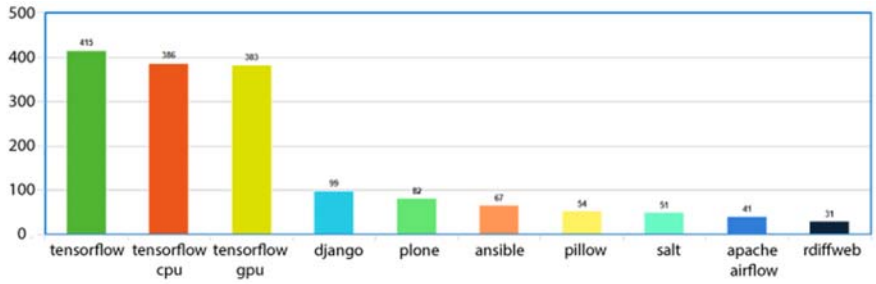

Fig. 10 Most often encountered Python packages

Django, recognized for its Object-Relational Mapper, has consistently held a position in the top 10 web development frameworks for multiple years, as indicated by [25]. It is built on Python, a candidate for the most popular and easiest learning coding language. However, Django contains high vulnerabilities, and the security update releases prevent remote attackers from exploiting these vulnerabilities like SQL injection.

Plone contains a bug allowing unauthorised users to access the restricted Python ecosystem. As a result, users who can edit PloneFormGen templates and site admins with ZMI access are vulnerable.

Ansible has a bug in which the secret information in async files is exposed when the job dir is changed to a world-accessible directory. As a result, an unauthorised user on that system will be able to view confidential information in an async status file.

The Python Imaging Library (PIL expansion) pillow is the image processing package for the Python programming language. It includes minimal image processing applications

that assist with picture editing, creation, and storage. This package's known vulnerabilities include DoS attacks, out-of-bounds read attacks, buffer overflow, etc.

```
In [20]: df=data['Packages'].value_counts()
         df.head(10)

Out[20]: tensorflow        415
         tensorflow-cpu    386
         tensorflow-gpu    383
         django             99
         plone              82
         ansible            67
         pillow             54
         salt               51
         apache-airflow     41
         rdiffweb           31
         Name: Packages, dtype: int64
```

Fig. 11 Most often encountered Python packages counts

Salt is a unique infrastructure management technique based on a dynamic communication network. Salt also is used for data-driven automation, remote execution for infrastructure, app stack configuration management, etc. The affected version of this package leads to Arbitrary Code Execution. The ClearFuncs class in the salt-master process does not correctly verify method calls [23]. This enables a remote user to access authorisation. It is also useful for retrieving user tokens from the salt master and running arbitrary instructions on salt minions.

Apache airflow is a Python-based open-source platform for developing, analysing, and scheduling batch-oriented processes. It comes with a default security key that is used to sign authentication information. This results in a security misconfiguration. A session cookie is generated when a user logs in with their authentication information in JSON format. Using a key, the JSON can identify the user's login history. This JSON file is signed with a string defined in the airflow.cfg config file [24].

An interface to rdiff-backup repositories is provided by rdiffweb. Because of missing set timeouts, affected versions of this software are vulnerable to Insufficient Session Expiration, which allows an attacker to steal the user session while utilising a shared computer. Additionally, an attacker can read sensitive data even when they are not signed into an account due to poor cache management.

These are the attacks that occurred within this ecosystem.

C. Results from Hybrid Data Scanning Tool

The report generated by the hybrid data scanning tool gives an overall solution after scanning a project. It identifies areas of vulnerability in projects, allowing them to take actions to minimise exposures and lower the risk of an attack as shown in Fig. 12.

Bandit Result	
Issue # 1	
Issue	[B105:hardcoded_password_string] Possible hardcoded password: 'aaaaaaa'
Severity	Low
Confidence	Medium
CWE	CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
Location	a/vulpy.py:16:11
Issue # 2	
Issue	[B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkzeug debugger and allows the execution of arbitrary code.
Severity	High
Confidence	Medium
CWE	CWE-94 (https://cwe.mitre.org/data/definitions/94.html)
Location	a/vulpy.py:55:0
Snyk Result	
Issue # 1	
Issue	[SNYK-PYTHON-BABEL-1278589] Directory Traversal
Severity	Medium
CWE	Package Name: babel
Package Name	CWE: CWE-22
Version	2.8.0
Fixed Version	2.9.1
Issue # 2	
Issue	[SNYK-PYTHON-RDIFFWEB-3114406] Insufficient Session Expiration
Severity	Medium
CWE	Package Name: rdiffweb
Package Name	CWE: CWE-613
Version	2.5.0a7
Fixed Version	2.5.0a8
Dlint Result	
Issue # 1	
Issue	" DUO130 insecure use of "hashlib" module

Fig. 12 Generated report in HTML format

VI. Conclusion

This research examined the security flaws in Python packages. The attacks that occur due to vulnerability cannot be neglected. From developing a small project to working in a large corporation, it is critical to understand how to safeguard a system. Although this research could not mitigate the problem, and new vulnerabilities are being discovered daily, the revised analysis provided an overall understanding. From this, we may infer the severity of these exploits, the most prevalent attacks in systems, the most impacted packages, and various ways to leverage them. Also, if we know how the most common threats occur, we may avoid them by implementing security measures. Finally, it is essential to ensure security remains in the code after constructing the code. The automatic scanner report will protect against any vulnerability attack.

References

[1] "PyPI · the Python Package Index," PyPI. (Online). Available: https://pypi.org/. (Accessed: 26-Dec-2022).
[2] M. Alfadel, D. E. Costa, and E. Shihab, "Empirical analysis of security vulnerabilities in python packages," in 2021 IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2021.
[3] "NVD - home," Nist.gov. (Online). Available: https://nvd.nist.gov/. (Accessed: 26-Dec-2022).
[4] "Snyk vulnerability database," Find detailed information and remediation guidance for vulnerabilities. Accessed on 26-Dec-22 at: https://security.snyk.io/
[5] "Common Vulnerability Scoring System version 3.1, Specification Document, Revision 1". Accessed on 12-Jan-2023 at:

https://www.first.org/cvss/v3-1/cvss-v31-specification_r1.pdf
[6] "Common Weakness Enumeration," Mitre.org. Accessed on 26-Dec-2022 at : https://cwe.mitre.org/index.html
[7] Wenliang Du, "Computer & Internet Security: A Hands-on Approach 2nd Edition", Independently published, May 2019.
[8] A. Bagmar, J. Wedgwood, D. Levin, and J. Purtilo, "I Know What You Imported Last Summer: A study of security threats in the python ecosystem," arXiv (cs.CR), 2021.
[9] A. Gkortzis, D. Mitropoulos, and D. Spinellis, "VulinOSS: A dataset of security vulnerabilities in open-source systems," in Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18, 2018.
[10] J. Ruohonen, K. Hjerppe, and K. Rindell, "A large-scale security-oriented static analysis of python packages in PyPI," in 2021 18th International Conference on Privacy, Security and Trust (PST), 2021.
[11] J. Ruohonen, "An empirical analysis of vulnerabilities in python packages for web applications," in 2018 9th International Workshop on Empirical Software Engineering in Practice (IWESEP), 2018.
[12] J. Ruohonen, "The similarities of software vulnerabilities for interpreted programming languages," in 2021 IEEE International Conference on Progress in Informatics and Computing (PIC), 2021.
[13] G. Antal, M. Keleti, and P. Hegedűs, "Exploring the security awareness of the python and JavaScript open source communities," in Proceedings of the 17th International Conference on Mining Software Repositories, 2020.
[14] S. Turner, "Security vulnerabilities of the top ten programming languages: C, Java, C++, Objective-C, C#, PHP, Visual Basic, Python, Perl, and Ruby," Journal of Technology Research.
[15] J. Garbajosa, X. Wang, and A. Aguiar, Eds., Agile Processes in Software Engineering and Extreme Programming: 19Th International Conference, XP 2018, Porto, Portugal, May 21-25, 2018, Proceedings, 1st ed. Cham, Switzerland: Springer International Publishing, 2018.
[16] J. Lopez and Y. Wu, Eds., Information security practice and experience: 11Th international conference, ISPEC 2015, Beijing, China, May 5-8, 2015, proceedings, 2015th ed. Basel, Switzerland: Springer International Publishing, 2015.
[17] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford, "How developers diagnose potential security vulnerabilities with a static analysis tool," IEEE Trans. Soft. Eng., vol. 45, no. 9, pp. 877–897, 2019.
[18] M. R. Rahman, A. Rahman, and L. Williams, "Share, but be aware: Security Smells in Python Gists," in 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2019.
[19] S. Peng, P. Liu, and J. Han, "A python security analysis framework in integrity verification and vulnerability detection," Wuhan Univ. J. Nat. Sci., vol. 24, no. 2, pp. 141–148, 2019.
[20] "CWE-2022 CWE top 25 most dangerous software weaknesses," Mitre.org. (Online). Accessed on 26-Dec-2022 at: https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html.
[21] G. Blok Dyk, TensorFlow: A Complete Guide. 5starcooks, 2018.
[22] "Denial of Service guidance, National Cyber Security Centre. Accessed on 26-Dec-2022 at: https://www.ncsc.gov.uk/collection/denial-service-dos-guidance-collection.
[23] "Red Hat Bugzilla-Bug 1832472 – (CVE-2020-11651) CVE-2020-11651 salt: salt-master process ClearFuncs class does not properly validate method calls," Redhat.com. Accessed on 26-Dec-2023 at: https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2020-11651
[24] "Python API reference – airflow documentation," Apache.org. Accessed on 19-Dec-2023 at: https://airflow.apache.org/docs/apache-airflow/stable/python-api- ref.html.
[25] Django. (2019). The web framework for perfectionists with deadlines | Django. djangoproject.com.2019 (Online). Available at https://www.djangoproject.com/.