УДК 004.056:004.043
DOI 10.17513/snt.39913

ДЕТЕКТИРОВАНИЕ УЯЗВИМОСТЕЙ В ИМПОРТИРУЕМЫХ БИБЛИОТЕКАХ ЯЗЫКА PYTHON С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ СТАТИЧЕСКОГО АНАЛИЗА

Швыров В.В., Капустин Д.А., Сентяй Р.Н.

ФГБОУ ВО «Луганский государственный педагогический университет», Луганск, e-mail: slsh@i.ua

Использование дополнительных модулей и библиотек значительно повышает эффективность разработки программ на Python. Однако большинство из библиотек являются проектами с открытым исходным кодом, которые зачастую разрабатываются энтузиастами и могут содержать потенциальные уязвимости или вредоносный код. В связи с этим возникает необходимость разработки эффективных программных средств для проверки импортируемых библиотек на предмет наличия в них известных дефектов и уязвимостей, которые описаны в различных открытых каталогах. Методы статического анализа программного кода могут быть эффективно использованы для детектирования различных уязвимостей и повышения качества программ на языке Python. Анализ количества публикаций по данной тематике показывает существенный интерес авторов к проблемам разработки безопасного программного обеспечения. В статье рассматривается проблема обнаружения уязвимостей в сторонних библиотеках Python, используемых в проектах с открытым исходным кодом. Представлено описание общей схемы разработки, а также реализация детектора уязвимостей, который обеспечивает обнаружение потенциальных проблем на основании данных открытых каталогов уязвимостей и повышает безопасность приложений на Python. Кроме того, в работе представлен анализ уязвимостей наиболее популярных библиотек с открытым исходным кодом на Python.

Ключевые слова: статический анализ, уязвимость, CVE, Python, импортирование библиотек в Python

DETECTING VULNERABILITIES IN IMPORTED PYTHON LIBRARIES USING STATIC ANALYSIS METHODS

Shvyrov V.V., Kapustin D.A., Sentyay R.N.

Luhansk State Pedagogical University, Luhansk, e-mail: slsh@i.ua

The use of additional modules and libraries significantly increases the efficiency of developing programs in Python. However, most of the libraries are open source projects that are often developed by enthusiasts and may contain potential vulnerabilities or malicious code. In this regard, there is a need to develop effective software tools for checking imported libraries for the presence of known defects and vulnerabilities, which are described in various open catalogs. Static code analysis methods can be effectively used for detecting various vulnerabilities and improving the quality of Python programs. An analysis of the number of publications on this topic shows the authors' significant interest in the problems of developing secure software. This article addresses the problem of vulnerability detection in third-party Python libraries used in open-source projects. It provides an overview of the general development scheme and the implementation of a vulnerability detector that identifies potential issues based on data from open vulnerability repositories, thus enhancing the security of Python applications. Additionally, the paper presents an analysis of vulnerabilities in the most popular open-source Python libraries.

Keywords: static analysis, vulnerability, CVE, Python, importing libraries in Python

Язык программирования Python устойчиво занимает лидирующие позиции в различных рейтингах [1]. Например, согласно данным рейтинга TIOBE (рис. 1) по состоянию на октябрь 2023 г., Python сохраняет первую позицию, опережая языки C, C++ и Java.

Современной тенденцией в области разработки программного обеспечения является формирование развитых экосистем для различных языков программирования, включающих, помимо форумов и сообществ программистов, различные репозитории дополнительных пакетов, которые существенно расширяют возможности конкретного языка, экономят время и в целом делают работу разработчиков значительно эффективнее. В частности, наиболее известным репозиторием пакетов для языка JavaScript считается npm, для Java – Maven, NuGet для языка C#, PyPI – репозиторий библиотек для Python. Следует отметить, что анализ безопасности для экосистемы npm проводился в работе [2], для PyPI – в работе [3].

По данным рейтинга OWASP Top 10 (Open Web Application Security Project [4]), использование уязвимых и устаревших компонент занимает шестую позицию. Данные рейтинга свидетельствуют об актуальности разработок, связанных с анализом используемых библиотек и с повышением безопасности разрабатываемого программного обеспечения в целом.

Oct 2023	Oct 2022	Change	Programming Language	Ratings	Change
1	1		Python	14.82%	-2.25%
2	2		C	12.08%	-3.13%
3	4	▲	C++	10.67%	+0.74%
4	3	▼	Java	8.92%	-3.92%
5	5		C#	7.71%	+3.29%

Рис. 1. Рейтинг TIOBE

При выполнении анализа безопасности программного кода важную роль играют методы статического анализа [5-8], а также различные каталоги уязвимостей: каталог видов уязвимостей – CWE (Common Weakness Enumeration [9]), каталоги найденных уязвимостей (Банк данных угроз ФСТЭК России [10], CVE Common Vulnerabilities and Exposures [11]). Кроме того, одним из известных ресурсов является Snyk, который предоставляет информацию об уязвимостях в различных открытых библиотеках и проектах, а также классифицирует их по экосистемам для разных языков программирования.

Целью данной работы является разработка детектора уязвимостей в импортируемых библиотеках для языка Python с использованием методов статического анализа программного кода и открытых каталогов уязвимостей программного обеспечения, а также исследование его эффективности путем сканирования наиболее загружаемых библиотек Python, представленных в репозитории PyPI.

Для достижения поставленной цели необходимо решить ряд теоретических и практических задач:

- изучить основные тенденции и направления развития в области анализа программного кода, а также исследовать публикационную активность в области данной тематики;
- выполнить анализ различных вариантов получения блока импортируемых библиотек для кода Python;
- разработать методы анализа используемых библиотек с использованием различных открытых каталогов уязвимостей;
- оценить эффективность работы алгоритма детектирования путем исследования наиболее популярных библиотек Python.

Актуальность исследования обусловлена рядом факторов. Во-первых, в связи с различными санкционными ограничениями ряд зарубежных сервисов становится недоступен либо блокируется, в связи с этим возникает необходимость поиска новых доступных средств проверки программного кода. Во-вторых, в рамках общей тенденции импортозамещения зарубежного программного обеспечения возникает острая необходимость отечественных разработок в области безопасности и анализа программного кода.

Материалы и методы исследования

Для анализа программного кода используются различные методы, наиболее распространенными являются методы статического и динамического анализа, различные варианты тестирования программного кода, а также синтетические методы. Обзор ряда методов представлен в работе [12].

В связи с обширным числом публикаций возрастает важность обзорных работ, которые могут помочь ориентироваться в том или ином конкретном подходе к анализу программного кода. В частности, следует выделить работу, посвященную общему анализу использования методов статического анализа и машинного обучения для детектирования вредоносных программ [13].

Для анализа подключаемых библиотек и поиска уязвимостей в зависимостях в проектах на языке Python существует большое количество различных проектов, которые в большинстве случаев используют базы CVE, CWE, а также базу PyPa, которая интегрирована в проект Open Source Vulnerabilities. Среди таких проектов можно выделить проекты Bandit, pip-audit, Safety (использует собственную базу SafetyDB). Кроме того, существует множество универсальных коммерческих проектов для анализа безопасности программного кода, например Infowatch Appercut, АК-ВС 2. Существует также ряд инструментальных средств, которые используют нейросетевые методы для поиска уязвимостей Embold, DeepCode AI (используется в проекте Snyk).

Результаты исследования и их обсуждение

Исследование активности авторов в области методов статического анализа программного кода, путем поиска научных

публикаций по ключевому запросу «static analysis» на платформе ArXiv, показало, что в корпусе найденных публикаций содержится около 900 работ.

Среди русскоязычных ресурсов следует выделить портал библиотеки «Киберленинка», которая построена на парадигме открытой науки (Open Science). Поиск по ключевому запросу «статический анализ» показывает более 400 найденных публикаций за последние 10 лет.

Детальный анализ страницы с результатами поиска на ArXiv позволил установить, что гиперссылки на статьи в формате PDF имеют вид https://arxiv.org/pdf/YYMM.NNNNN, где первые четыре символа соответствуют году и месяцу публикации, таким образом, с использованием современных средств парсинга было определено количество публикаций по годам. Аналогичный анализ результатов поиска на «Киберленинке» дал возможность получить общие данные о публикационной активности как в англоязычном, так и в русскоязычном сегменте (рис. 2).

Суммируя показатели русскоязычных и англоязычных публикаций, можно увидеть общую тенденцию роста интереса авторов к тематике статического анализа программного кода и методам повышения качества и безопасности программного обеспечения (рис. 3).

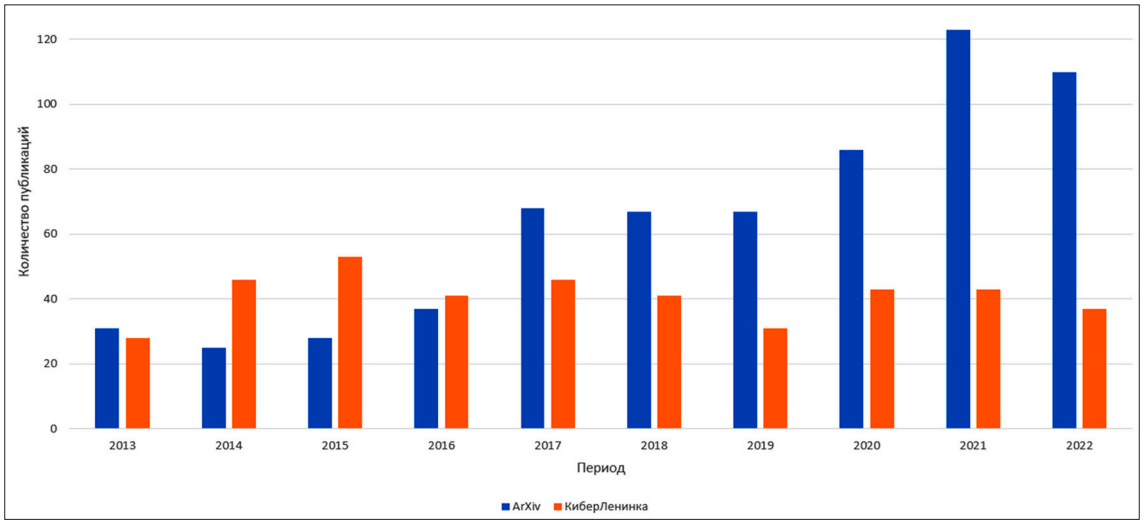

Рис. 2. Данные о количестве публикаций по тематике статического анализа

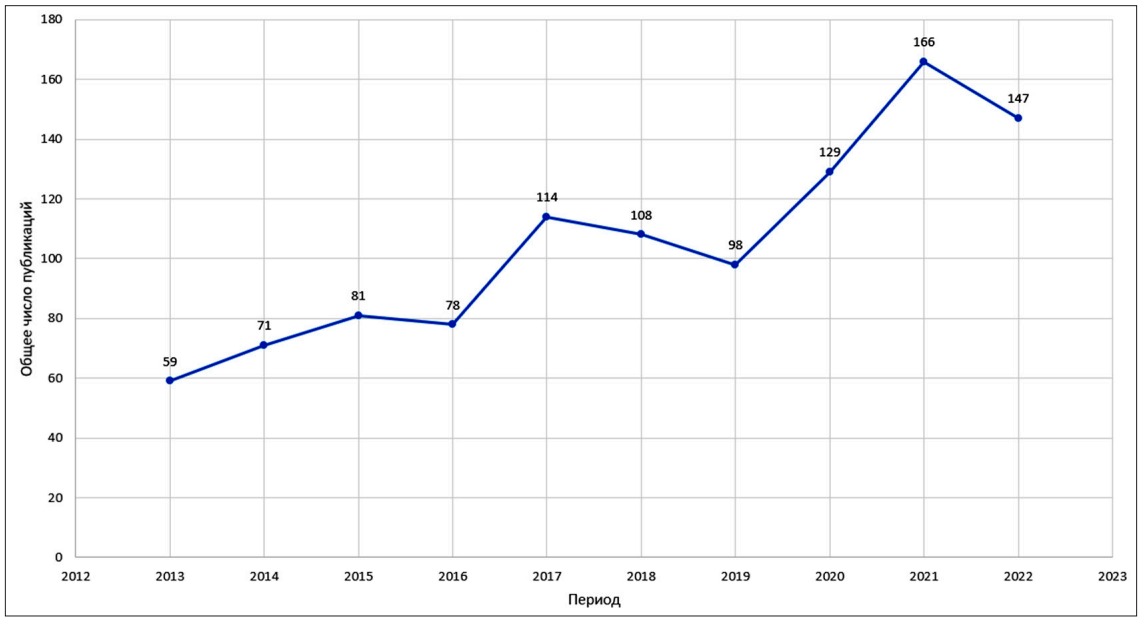

Рис. 3. Динамика изменения количества публикаций

Рассмотрим возможности использования методов статического анализа для разработки детектора уязвимостей в программном коде на Python. Для подключения библиотеки или модуля в коде Python могут использоваться несколько вариантов. Использование инструкции import является одним из наиболее распространенных. Например, строка импорта может принимать такие формальные формы записи:

1) import X
2) from X import Y
3) import X1, X2, …, Xn
4) import X as Name

Кроме того, функции, такие как importlib.import_module() и __import__(), также могут быть использованы для вызова механизма импорта.

Для получения списка импортируемых библиотек существует достаточно много инструментов и подходов. В частности, можно использовать библиотеки ModuleFinder, listimports, snakefood.

Процесс интерпретации программного кода на Python включает в себя несколько этапов, включая анализ исходного кода, его преобразование в абстрактное синтаксическое дерево, создание графа управления выполнением, проведение различных проверок структуры дерева и генерацию байт-кода. Полученное абстрактное синтаксическое дерево может быть эффективно использовано для извлечения списка импортируемых библиотек и проведения статического анализа программного кода. Например, в работах [14; 15] анализ импортируемых библиотек проводится в контексте обнаружения устаревших программных интерфейсов в библиотеках Python.

Таким образом, в рамках решаемых в данной работе задач была использована библиотека ast (класс ast.NodeVisitor) для прохода по произвольному списку файлов *.py в заданной директории.

Общая схема работы детектора включает следующие этапы:

1. Получение списка файлов для проверки.

2. Получение списка импортируемых библиотек для каждого файла из списка.

3. Проверка отсутствия библиотеки в белом списке, который может быть задан пользователем.

4. Поиск по каждому значению из полученного списка библиотек в каталоге уязвимостей CVE.

5. Первичная фильтрация полученных результатов по заданным ключевым словам «Python», «library», «package» «framework», «module».

6. Дополнительная проверка результатов в базе CVE с помощью средств поиска по ID CVE посредством API-запросов. В частности, проверка совпадения полей vendor либо product с названием проверяемой библиотеки, получение данных о версиях, для которых актуальна описанная уязвимость, описание уязвимости и ее типа согласно классификации типов дефектов CWE.

7. Получение данных о текущей версии проверяемой библиотеки из репозитория PyPI.

8. Экспорт всех отфильтрованных результатов в файл отчета.

Указанная выше схема была реализована на языке Python, которая работает в виде консольного приложения.

В работе также был проведен анализ наличия потенциальных уязвимостей в наиболее загружаемых библиотеках Python (по состоянию на октябрь 2023 г.). Для получения данных о популярности библиотек использовались данные открытого рейтинга наиболее загружаемых библиотек Top PyPI Packages. В качестве данных для анализа был сформирован список из первых 1000 строк рейтинга Top PyPI, который позволил сформировать файл для последующей обработки с помощью детектора.

Проведенный с помощью детектора анализ, на основании данных открытой базы уязвимостей CVE, позволил определить ряд потенциальных уязвимостей, которые связаны использованием различных модулей и библиотек, подключаемых в инструкциях импорта.

Для исследуемых библиотек в каталоге CVE за последние три года было обнаружено около 80 записей о найденных уязвимостях. В таблице представлены уязвимости за 2023 год для соответствующих библиотек Python с указанием версий, для которых актуальна уязвимость. Также в таблице указана текущая версия библиотеки, данные о которой взяты из репозитория библиотеки в PyPI. Кроме того, в таблице представлено краткое описание уязвимости и ее тип согласно каталогу типов уязвимостей CWE.

Детальный анализ полученных данных за последние три года показал, что найденные уязвимости в наиболее загружаемых библиотеках соответствуют 28 различным типам CWE. В частности, можно выделить такие типы уязвимостей, как CWE-400 (неуправляемое потребление ресурсов), CWE-200 (конфиденциальная информация может быть доступна несанкционированным пользователям), CWE-20 (недостаточная проверка ввода).

Найденные уязвимости с указанием версий библиотек,
для которых уязвимость актуальна

Название библиотеки	Текущая версия	Версии, для которых найдены уязвимости в базе CVE	CVE ID BDU ID	Краткое описание уязвимости (тип уязвимости по базе CWE)
cryptography	41.0.5	>=1.8, <39.0.1	CVE-2023-23931 BDU:2023-02656	Уязвимость связана с методом Cipher. update_into, ее эксплуатация позволяла изменять неизменяемые объекты и приводила к повреждению выходных данных (CWE-754)
ipython	8.16.1	< 8.10	CVE-2023-24816	Уязвимость связана с инъекцией команд в случае выполнения ряда специфический условий (CWE-20)
werkzeug	3.0.1	< 2.2.3	CVE-2023-25577 BDU:2023-02343	Атакующий может вызвать отказ в обслуживании, отправляя сформированные данные на точку доступа, которая их разбирает (CWE-770)
gradio	3.50.2	< 3.13.1	CVE-2023-25823	При использовании функции share в Gradio приватный SSH-ключ отправляется каждому пользователю, который подключается к серверу Gradio (CWE-798)
configobj	5.0.8	Все версии	CVE-2023-26112	Уязвимость связана с атакой на регулярное выражение, которая может быть осуществлена через функцию validate, если разработчик включит вредоносное значение в конфигурационный файл на стороне сервера (CWE-1333)
pydash	7.0.6	6.0.0	CVE-2023-26145	Ряд методов в pydash принимает пути в виде строк, которые указывают на вложенные объекты Python. Эти пути могут использоваться для доступа к внутренним атрибутам класса и элементам словаря, чтобы извлекать, изменять или вызывать вложенные объекты Python (CWE-78)
langchain	0.0.325	<0.0.131	CVE-2023-29374	Уязвимость позволяет проводить атаки с внедрением команд, позволяющие выполнить произвольный код с использованием метода exec в Python (CWE не найден)
sqlparse	0.4.4	>= 0.1.15, < 0.4.4	CVE-2023-30608 BDU:2023-03345	Наличие регулярного выражения в SQL-парсере, которое подвержено атаке ReDoS (Denial of Service на основе регулярных выражений) (CWE-1333)
starlette	0.31.1	<0.25.0	CVE-2023-30798	Уязвимость позволяет неавторизованному удаленному атакующему указать любое количество полей формы или файлов, что может вызвать отказ в обслуживании HTTP-сервиса (CWE-400)
gradio	3.50.2	< 3.34.0	CVE-2023-34239	Уязвимость связана с отсутствием фильтрации путей, что позволяет пользователям получать неограниченный доступ к файлам (CWE-20)
aiohttp	3.8.6	< 3.8.5	CVE-2023-37276 BDU:2023-05462	Уязвимость связана с библиотекой llhttp (v6.0.6), используемой для асинхронного парсинга HTTP-запросов (CWE-444)
cryptography	41.0.5	<41.0.2	CVE-2023-38325 BDU:2023-05436	Уязвимость связана с некорректной обработкой SSH-сертификатов, содержащих критические опции

Окончание табл.

Название библиотеки	Текущая версия	Версии, для которых найдены уязвимости в базе CVE	CVE ID BDU ID	Краткое описание уязвимости (тип уязвимости по базе CWE)
gitpython	3.1.40	<= 3.1.32	CVE-2023-40590 BDU:2023-05185	Если пользователь запускает GitPython из директории с репозиторием, где есть исполняемый файл git.exe или git, этот исполняемый файл будет запущен вместо того, что находится в переменной окружения PATH пользователя. Злоумышленник может создать вредоносный git исполняемый файл в репозитории и убедить пользователя скачать этот репозиторий
gitpython	3.1.40	<= 3.1.34	CVE-2023-41040 BDU:2023-05476	В некоторых случаях имя файла, который должен быть прочитан, предоставляется пользователем, и GitPython не проверяет, находится ли этот файл за пределами директории .git. Это позволяет злоумышленнику заставить GitPython читать любой файл на системе
urllib3	2.0.7	>= 2.0.0, < 2.0.7	CVE-2023-43804	Заголовок HTTP Cookie не удаляется при перенаправлении между разными источниками (CWE-200)
urllib3	2.0.7	>= 2.0.0, < 2.0.6	CVE-2023-45803	Тело запроса не удаляется после перенаправления в urllib3 (CWE-200)
twisted	23.8.0	<23.10.0rc1	CVE-2023-46137	Уязвимость связана с асинхронной обработкой нескольких HTTP-запросов в одном TCP-пакете без гарантии порядка ответов (CWE-444)
pip	23.3.1	<23.3	CVE-2023-5752	Уязвимость в pip связана с установкой пакетов из репозитория Mercurial VCS (CWE-77)

Заключение

В работе было проведено исследование публикационной активности отечественных и зарубежных авторов в области различных методов статического анализа программного кода. Полученные данные свидетельствуют о значительном интересе авторов к данной тематике.

Кроме того, в работе представлена разработка детектора уязвимостей в импортируемых библиотеках и модулях в программном коде на языке Python. С помощью разработанного приложения был проведен анализ списка из 1000 наиболее загружаемых Python-библиотек различных проектов с открытым исходным кодом. В частности, выполнялась проверка наличия библиотеки в открытом каталоге уязвимостей CVE, а также извлечение дополнительных данных о найденных уязвимостях в случае нахождения совпадений.

Следует отметить, что, несмотря на растущую популярность проектов на Python, разработчиками зачастую не уделяется должного внимания вопросам безопасности, в связи с этим многие из малоизвестных библиотек могут быть потенциально опасны.

Огромное количество уязвимостей, которые представлены в различных открытых каталогах, делает затруднительным ручную проверку всех используемых модулей или библиотек в проектах на Python. Кроме того, в базе CVE фильтрация уязвимостей по языкам программирования отсутствует, а поиск по вендору или продукту не всегда дает корректные результаты. Разработанный детектор может быть использован для автоматизации ряда рутинных проверок, а также как дополнение к существующим способам тестирования и контроля качества программного кода.

Результаты исследования показали, что для наиболее популярных библиотек существует значительное количество найденных уязвимостей. В то же время данные уязвимости в большинстве случаев оперативно

устраняются разработчиками. Однако проекты, которые используют устаревшие версии, могут быть уязвимы.

Список литературы

1. TIOBE Index for March 2022 [Электронный ресурс] URL: https://www.tiobe.com/tiobe-index/ (дата обращения: 08.11.2023).

2. Decan A., Mens T., Constantinou E. On the Impact of Security Vulnerabilities in the npm Package Dependency Network // Proceedings of the 15th International Conference on Mining Software Repositories (MSR 2018). Gothenburg: ACM. 2018. P. 181–191.

3. Ruohonen, J., Hjerppe, K., Rindell, K. A Large-Scale Security-Oriented Static Analysis of Python Packages in PyPI // Proceedings of the 18th Annual International Conference on Privacy, Security and Trust (PST 2021), Auckland, IEEE, 2021. P. 1-10.

4. OWASP Web Security Testing Guide. [Электронный ресурс]. URL: https://github.com/OWASP/wstg (дата обращения: 08.11.2023).

5. Cousot P. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints // Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. 1977. P. 238–252.

6. Allen F.E. Control flow analysis // ACM SIGPLAN Notices. 1970. Vol. 5, Is. 7. P. 1–19.

7. Аветисян А.И., Белеванцев А.А., Бородин А.Е., Несов В. Использование статического анализа для поиска уязвимостей и критических ошибок в исходном коде программ // Труды ИСП РАН. 2011. Т. 21. С. 23-38.

8. Smith J., Johnson B., Murphy-Hill E.R., Chu B., Lipford H.R. How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis Tool // IEEE Transactions on Software Engineering. 2019. Vol. 45(9). P. 877–897.

9. Common Weakness Enumeration. [Электронный ресурс]. URL: https://cwe.mitre.org/about/index.html (дата обращения: 08.11.2023).

10. Банк данных угроз безопасности информации [Электронный ресурс]. URL: http://www.bdu.fstec.ru/ (дата обращения: 08.11.2023).

11. CVE. [Электронный ресурс]. URL: https://cve.mitre.org/ (дата обращения: 08.11.2023).

12. Мерзлякова Е.Ю., Янченко Е.В. Обзор методов верификации и оценки качества программного обеспечения // Вестник СибГУТИ. 2023. Т. 17, № 1. С. 92–106.

13. Shalaginov A., Banin S., Dehghantanha A., Franke K. Machine Learning Aided Static Malware Analysis: A Survey and Tutorial. ArXiv, abs/1808.01201. 2018.

14. Vadlamani A., Kalicheti R., Chimalakonda S. APIScanner – Towards Automated Detection of Deprecated APIs in Python Libraries // 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). 2021. P. 5-8.

15. Wang J., Li L., Liu K., Cai H. Exploring how deprecated python library apis are (not) handled // Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020. P. 233–244.