УДК 004.056, 004.043
DOI: 10.17587/prin.13.394-403

Д. А. Капустин, канд. техн. наук, доц., kap-kapchik@mail.ru,
В. В. Швыров, канд. физ.-мат. наук, доц., slsh@i.ua,
Т. И. Шулика, ассистент, shulika-tatyana@mail.ru,
Луганский государственный педагогический университет

Статический анализ корпуса исходных кодов Python-приложений

Одним из популярных методов анализа кода программного обеспечения является метод статического анализа. Такой метод позволяет не только проверять код на соответствие спецификации языка, но и находить в коде потенциальные уязвимости. В работе выполнен статический анализ корпуса листингов приложений на Python с открытыми исходными кодами. С использованием библиотеки Bandit найдены статистические показатели различных категорий потенциальных уязвимостей, построена рейтинговая таблица уязвимостей, найденных в исследуемом наборе данных. Проведен качественный анализ угроз на основе данных каталога CWE.

Ключевые слова: *анализ безопасности, большие данные, статический анализ, уязвимость, угроза, bandit, CWE, OWASP, linters, source code analysis, python dataset*

Введение

Вопросы написания безопасного кода, в частности, отсутствия уязвимостей и ошибок в приложениях являются актуальной темой исследования в последние годы. Это связано с растущими вызовами и требованиями эффективной и надежной работы приложений в условиях агрессивной конкуренции между разработчиками с одной стороны и растущего числа хакерских атак с другой стороны [1, 2].

Одним из эффективных методов анализа программного кода является статический анализ кода, который получил свое развитие в работах Кузо [3] и Аллена [4]. Одним из первых приложений, которое выполняло проверку кода языка С с помощью статического анализа, было приложение Lint [5]. Впоследствии класс программных средств, осуществляющих статический анализ, стали называть линтерами.

Методы статического анализа программного кода активно развивались в последнее десятилетие. Об этом свидетельствует значительное число публикаций и обзорных работ по данному направлению [6—11]. Существенный прогресс в развитии и использовании данных методов связан с возросшими вычислительными способностями современных систем, а также необходимостью поиска потенциальных угроз и уязвимостей в программном обеспечении (ПО). Современные подходы в области анализа безопасности веб-приложений представлены в проекте OWASP (*Open Web Application Security Project*) [12].

Следует отметить, что кроме статического анализа используются также методы динамического анализа и методы формальной верификации программ. Динамический метод анализа предполагает мониторинг выполнения программы преимущественно на этапе тестирования, на некотором наборе входных данных. Основная идея фаззинга состоит в том, чтобы "выявить ошибку до тех пор пока она не стала уязвимостью". Выделяют фаззинг-анализаторы, которые используют специально сгенерированные наборы входных данных для тестирования приложения, например, путем мутационного или генеративного фаззинг-тестирования. В качестве примера можно привести утилиту Burp Suite, которую можно использовать для тестирования веб-приложений методом фаззинга. Каждый из методов анализа программного кода имеет свои преимущества и недостатки, в связи с этим на практике желательно использование различных методов.

Значительную роль в развитии методов статического анализа для исследования проблем безопасности приложений сыграли каталоги уязвимостей и дефектов ПО. Одними из наиболее известных таких каталогов являются полученные в рамках

проектов CWE (*Common Weakness Enumeration*) и CVE (*Common Vulnerabilities and Exposures*) [13, 14]. Последний, по состоянию на начало мая 2022 г., насчитывает более 175 000 описанных уязвимостей. С помощью списков CWE и CVE составлен рейтинг OWASP Top 10 [15]. Данный рейтинг приведен в табл. 1.

В связи со стремительным ростом популярности языка Python [16] в различных проектах с открытым исходным кодом и упомянутыми ранее каталогами CVE и CWE, возникает естественный вопрос построения рейтинга уязвимостей и дефектов аналогичного OWASP Top 10 для Python-приложений.

Подобные исследования проводились в ряде работ зарубежных авторов, например, в направлении анализа безопасности кода во фреймворке Django [17], в контексте анализа динамической типизации в Python [18]. В работе [19] проводился анализ уязвимостей в Python-пакетах для разработки веб-приложений. Комплексный и системный анализ безопасности пакетов Python Package Index (PyPI) [20] выполнен в исследовании [21]. Для проверки кода пакетов в контексте настоящей статьи авторы используют статический анализатор Bandit [22]. В отличие от аналогичных работ, в статье представлен рейтинг потенциальных уязвимостей согласно каталогу CWE для проектов с открытым исходным кодом на языке Python, кроме того, полученные результаты сопоставляются с данными рейтинга OWASP Top 10, также выполнен частотный анализ жестко вписанных в программный код паролей.

Целью работы, результаты которой представлены в настоящей статье, являются анализ корпуса листингов Python-приложений с открытыми исходными кодами на предмет наличия потенциальных уязвимостей с использованием статического анализатора и составление рейтинга наиболее распространенных угроз на основании найденных потенциальных уязвимостей.

Таким образом, исследование должно дать ответ на перечисленные далее вопросы.

В1. Какие категории потенциальных уязвимостей по каталогу CWE наиболее распространены в проектах с открытым исходным кодом на Python?

В2. Какими категориями тестов (проверок) библиотеки Bandit найдено больше всего уязвимостей?

В3. Существует ли корреляция полученных статистических данных с данными рейтинга OWASP?

Для достижения поставленной цели исследования необходимо решить следующие задачи:

- выполнить предварительную обработку набора данных для анализа (выбор корпуса листингов);
- оценить возможности и параметры модулей проверки выбранного статического анализатора;
- разработать скрипт для автоматизации обработки результатов (построить выборку по заданным критериям);
- выполнить анализ результатов с использованием статистических методов.

Данные для анализа

Несмотря на рост использования методов машинного обучения и технологий анализа больших данных в сфере программной инженерии и информационной безопасности, поиск подходящего для анализа набора данных остается актуальной

Таблица 1

Рейтинг OWASP Top 10

№	Код категории	Категория уязвимостей
1	A01:2021	Нарушение контроля доступа (*Broken Access Control*)
2	A02:2021	Неудачное Использование Криптографии (*Cryptographic Failures*)
3	A03:2021	Внедрение кода (*Injection*)
4	A04:2021	Небезопасный дизайн (*Insecure Design*)
5	A05:2021	Неправильная конфигурация (*Security Misconfiguration*), категория A04:2017 Внедрение внешних XML сущностей — на данный момент входил в категорию A05:2021
6	A06:2021	Уязвимые и устаревшие компоненты (*Vulnerable and Outdated Components*)
7	A07:2021	Ошибки идентификации и аутентификации (*Identification and Authentication Failures*)
8	A08:2021	Нарушение целостности данных и ПО (*Software and Data Integrity Failures*)
9	A09:2021	Журнал безопасности и сбои мониторинга (*Security Logging and Monitoring Failures*)
10	A10:2021	Подделка запросов со стороны сервера (*Server-Side Request Forgery*)

Таблица 2

Общие характеристики набора данных

Характеристика	Значение
Число файлов	150 000
Размер на диске, Гбайт	1,05
Средний размер файла, Кбайт	7,39

задачей. Проведенный анализ каталогов с наборами данных [23] показал, что для языка Python для решения поставленных задач подходит датасет PY150, который содержит 150 000 файлов листингов на языке Python. Данный корпус листингов был разработан в рамках проекта SRILAB [24]. Корпус исходных кодов был сформирован путем сбора данных с GitHub-проектов на датасет Python в открытом доступе, с последующим удалением веток версий и дубликатов. В табл. 2 приведены общие характеристики корпуса листингов.

Статический анализатор Bandit

На данный момент существует широкий спектр как универсальных статических анализаторов, например, Infowatch Appercut [25], АК-ВС 2 [26], так и специализированных линтеров для языка Python, таких как Pylint [27], MyPY [28]. Хорошая документация и большой спектр модулей для тестирования обусловили выбор ранее упомянутого статического анализатора Bandit.

Утилита выполняет проверку кода в указанной директории с помощью множества отдельных тестов, которые представлены как плагины. При необходимости возможна разработка пользовательских плагинов для других проверок. В табл. 3 представлены категории тестов утилиты Bandit. Число тестов варьируется в различных категориях. Общее число детекторов — 67.

Таблица 3

Категории плагинов для поиска уязвимостей

Код теста	Описание категории
B1xx	Детекторы общих уязвимостей, в том числе детектор жестко вписанных паролей в коде
B2xx	Детектор режима отладки веб-приложения
B3xx	Детекторы вызова различных функций, которые могут привести к проблемам безопасности
B4xx	Детекторы небезопасного импорта
B5xx	Детекторы проблем, связанных с сетевыми и криптографическими протоколами
B6xx	Детекторы инъекций
B7xx	Детекторы межсайтового скриптинга

После выполнения проверки формируется отчет в выбранном формате (csv, json, html, text, xml, yaml). Результирующий отчет содержит ряд показателей для каждой найденной ошибки. В частности, это может быть: степень доверия (*Confidence* — низкая, средняя, высокая); код теста, с помощью которого обнаружена потенциальная уязвимость; степень угрозы (*Severity* — низкая, средняя, высокая); краткое текстовое описание ошибки; детальная информация о коде с ошибкой и ее расположении. На рис. 1 представлен пример фрагмента отчета в формате .json. Степень угрозы определяется непосредственно детектором библиотеки Bandit на основании данных каталога CWE.

Следует отметить, что параметр степени доверия может быть полезен при оценке количества ложно отрицательных срабатываний и ложно положительных случаев [29].

Анализ данных

После подготовки и загрузки корпуса исходных кодов был выполнен его анализ утилитой Bandit в стандартной конфигурации с последующим вы-


```
{
  "code": "10 import stat\n11 from subprocess import Popen, PIPE\n12 from riak.util import deep_merge\n",
  "col_offset": 0,
  "filename": "w:\\py\\astexample\\ex\\test_server22311.py",
  "issue_confidence": "HIGH",
  "issue_cwe": {
    "id": 78,
    "link": "https://cwe.mitre.org/data/definitions/78.html"
  },
  "issue_severity": "LOW",
  "issue_text": "Consider possible security implications associated with the subprocess module.",
  "line_number": 11,
  "line_range": [
    11
  ],
  "more_info": "https://bandit.readthedocs.io/en/1.7.4/blacklists/blacklist_imports.html#b404-import-subprocess",
  "test_id": "B404",
  "test_name": "blacklist"
}
```


Рис. 1. Пример фрагмента отчета об ошибке

водом отчета в файл json. В табл. 4—6 приведены общие метрики анализа по различным показателям. В таблицах в колонке "Значение, %" указана доля в процентах от общего числа ошибок.

Полученные характеристики показывают относительно невысокое число критически важных уязвимостей в исследуемом наборе данных.

В табл. 7 представлена детальная информация по категориям ошибок с указанием их кода по каталогу CWE. Данные получены путем анализа файла отчета. Значения сгруппированы по ключам "issue_cwe", "id" с последующим подсчетом числа ошибок. Вспомогательные вычисления проводились средствами Python.

Таблица 4

Общие данные проверки корпуса

Характеристика	Значение
Обработанных файлов	132 217
Ошибок обработки (ошибка построения AST — абстрактного синтаксического дерева)	17 783
Обработанных строк	18 364 306
Общее число потенциальных уязвимостей (предупреждений)	178 328

Таблица 5

Показатели по уровню доверия

Характеристика	Найдено потенциальных уязвимостей	Значение, %
Низкий уровень доверия	1370	0,77
Средний уровень доверия	10435	5,85
Высокий уровень доверия	166 523	93,38

Таблица 6

Показатели по уровню опасности

Характеристика	Найдено потенциальных уязвимостей	Значение, %
Низкий уровень опасности	161 396	90,51
Средний уровень опасности	14 469	8,11
Высокий уровень опасности	2463	1,38

Полученные результаты дают ответ на вопрос В1. Детальный анализ показал, что на первом месте оказались показатели уязвимостей, которые связаны с использованием оператора assert(), который, в свою очередь, определяется детектором B101 (134 699 значения). В табл. 7 в последних двух колонках также приведены показатели без учета значений детектора B101. Эти данные визуально представлены в диаграмме на рис. 2.

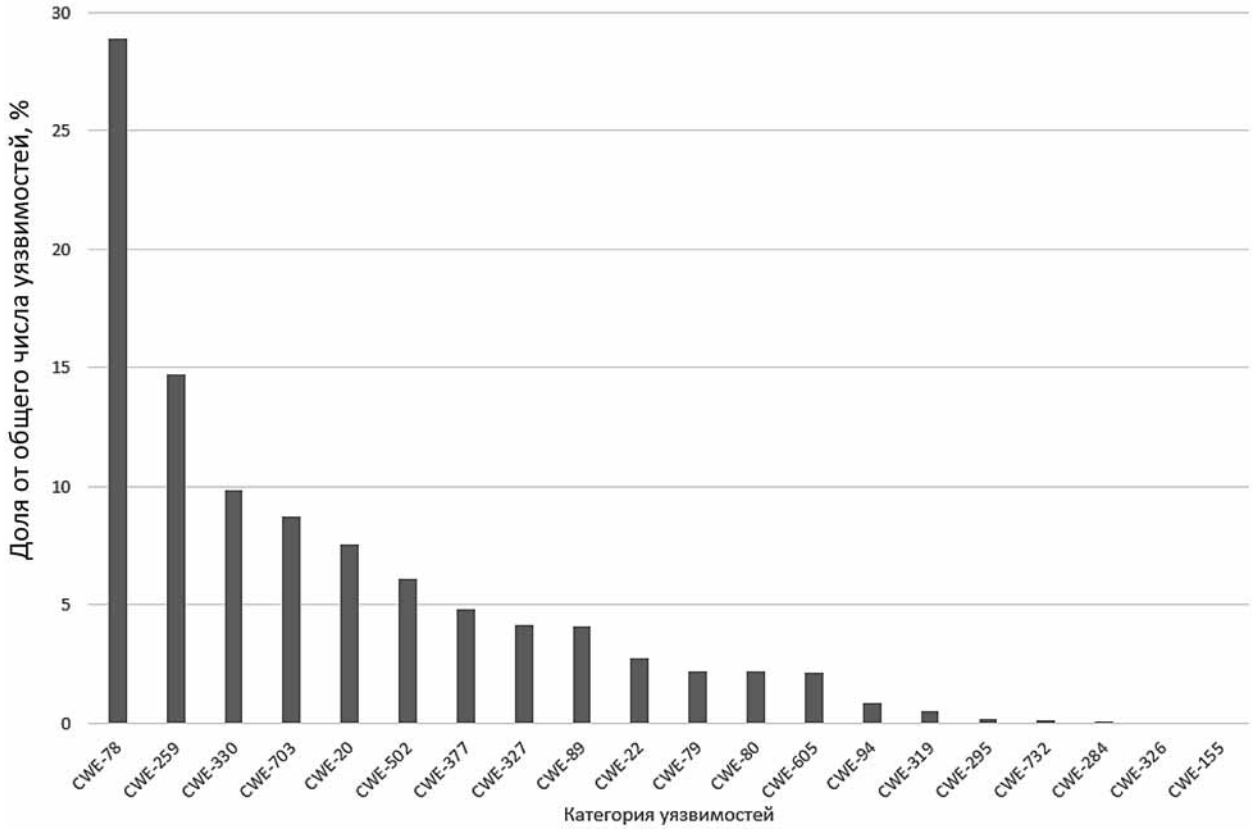

Рис. 2. Распределение частот найденных потенциальных уязвимостей без учета значений детектора B101

Таблица 7

Показатели по найденным кодам уязвимостей

Категория потенциальной уязвимости	Код	Абсолютная частота	Доля от общего числа уязвимостей, %	Абсолютная частота без B101	Доля от общего числа уязвимостей, % (без B101)
Некорректная обработка исключений	CWE-703	138 508	77,670	3809	8,73
Некорректная фильтрация специальных элементов в командах ОС (Командные инъекции)	CWE-78	12 608	7,070	12 608	28,90
Жестко закодированный пароль	CWE-259	6417	3,598	6417	14,71
Недостаточное качество случайных значений	CWE-330	4291	2,406	4291	9,84
Неправильная проверка ввода	CWE-20	3309	1,856	3309	7,58
Десериализация непроверенных данных	CWE-502	2665	1,494	2665	6,11
Небезопасный временный файл	CWE-377	2101	1,178	2101	4,82
Использование скомпрометированных или слабых криптографических алгоритмов	CWE-327	1812	1,016	1812	4,15
SQL-инъекции	CWE-89	1781	0,999	1781	4,08
Неправильное ограничение пути к каталогу с ограниченным доступом ("Обход пути")	CWE-22	1198	0,672	1198	2,75
Неправильная нейтрализация ввода во время создания веб-страницы ("Межсайтовый скриптинг")	CWE-79	953	0,534	953	2,18
Неправильная нейтрализация HTML-тегов, связанных со сценариями, на веб-странице (базовый XSS)	CWE-80	953	0,534	953	2,18
Несколько привязок к одному и тому же порту	CWE-605	936	0,525	936	2,15
Неправильный контроль над генерацией кода ("Внедрение кода")	CWE-94	385	0,216	385	0,88
Передача конфиденциальной информации открытым текстом	CWE-319	235	0,132	235	0,54
Неправильная проверка сертификата	CWE-295	78	0,044	78	0,18
Неверное назначение разрешений для критического ресурса	CWE-732	56	0,031	56	0,13
Неправильный контроль доступа	CWE-284	29	0,016	29	0,07
Недостаточная сила шифрования	CWE-326	9	0,005	9	0,02
Неправильная нейтрализация подстановочных знаков или совпадающих символов	CWE-155	4	0,002	4	0,01

Напомним, что сериализацией называется процесс преобразования информации из приложения в бинарный формат, проблемы десериализации (процесс обратный сериализации) непроверенных данных зачастую связаны с наличием встроенных методов для вывода данных на диск и с возможностью удаленного выполнения кода. На практике, проблемы могут возникать в связи с использованием уязвимостей в библиотеках Pickle и Os.system, которые могут приводить к возможности просмотра содержимого файлов или выполнению других инструкций в процессе десериализации. Детальное описание эксплуатации уязвимостей библиотеки Pickle можно найти в обзоре [30]

Таким образом, по результатам анализа лидируют потенциальные уязвимости, связанные с вопросами фильтрации данных, которые могут приводить к выполнению командных инъекций (CWE-78). В частности, они связаны с использованием модулей os и subprocess. Кроме того, широко распространены проблемные вопросы с жестко вписанными паролями в программном коде, что соответствует классу уязвимостей CWE-259, общее

число таких вопросов составило 6417. В случае если пароль жестко вписан в программный код, такой пароль будет идентичен для каждой установки, не может быть изменен администратором без ручной правки программного кода. Потенциально жестко вписанные пароли могут привести к массированным атакам на различные организации, которые используют приложение. Одним из путей решения данного проблемного вопроса является использование защищенных конфигурационных файлов, либо требование к пользователям ввода нового пароля после использования "первого логина".

Рассмотренные проблемные вопросы фильтрации данных относятся к критически значимым угрозам с высокой опасностью, показатель числа таких ошибок занимает лидирующую позицию среди показателей уязвимостей с высокой степенью угрозы (табл. 8). Поскольку в отчете об ошибках статического анализатора Bandit присутствуют фрагменты кода, которые содержат найденные строки с жестко вписанными паролями, то на основании анализа частот таких строк можно получить представление о наиболее часто встречаемых паролях в программном коде. Эти данные приведены в табл. 9.

Интересно заметить, что пароль "123456" встречается почти в три раза чаще, чем "12345", что свидетельствует о психологическом его восприятии разработчиками как более "надежного".

Группировка по числу найденных потенциальных уязвимостей по категориям проверочных тестов позволяет найти суммарное число значений в каждой из категорий и дает ответ на вопрос B2 (табл. 10).

Данные даже без учета теста B101 показывают, что наибольший процент выявленных проблем-

Таблица 8

Распределение числа уязвимостей по степеням угроз

Уязвимость	Код	Степень угрозы		
		Низкая	Средняя	Высокая
Некорректная обработка исключений	CWE-703	138 508	—	—
Некорректная фильтрация специальных элементов в командах ОС (Командные инъекции)	CWE-78	9290	2015	1303
Жестко закодированный пароль	CWE-259	6417	—	—
Недостаточное качество случайных значений	CWE-330	4291	—	—
Неправильная проверка ввода	CWE-20	1468	1724	117
Десериализация непроверенных данных	CWE-502	1337	1328	—
Небезопасный временный файл	CWE-377	—	2101	—
Использование скомпрометированных или слабых криптографических алгоритмов	CWE-327	85	1376	351
SQL-инъекции	CWE-89	—	1781	—
Неправильное ограничение пути к каталогу с ограниченным доступом ("обход пути")	CWE-22	—	1198	—
Неправильная нейтрализация ввода во время создания веб-страницы ("межсайтовый скриптинг")	CWE-79	—	953	—
Неправильная нейтрализация HTML-тегов, связанных со сценариями, на веб-странице (базовый XSS)	CWE-80	—	953	—
Несколько привязок к одному и тому же порту	CWE-605	—	936	—
Неправильный контроль над генерацией кода ("внедрение кода")	CWE-94	—	—	385
Передача конфиденциальной информации открытым текстом	CWE-319	—	52	183
Неправильная проверка сертификата	CWE-295	—	11	67
Неверное назначение разрешений для критического ресурса	CWE-732	—	32	24
Неправильный контроль доступа	CWE-284	—	—	29
Недостаточная сила шифрования	CWE-326	—	9	—
Неправильная нейтрализация подстановочных знаков или совпадающих символов	CWE-155	—	—	4

Таблица 9

Наиболее встречаемые пароли

№	Пароль	Абсолютная частота	Доля от общего числа паролей, %	№	Пароль	Абсолютная частота	Доля от общего числа паролей, %
1	Пароль отсутствует (пустой)	612	9,54	11	normaluser	64	1,00
2	password	525	8,18	12	123	59	0,92
3	secret	228	3,55	13	POST	55	0,86
4	pass	132	2,06	14	bar	52	0,81
5	test	130	2,03	15	foo	49	0,76
6	testpass	119	1,85	16	token	42	0,65
7	admin	108	1,68	17	user	40	0,62
8	123456	99	1,54	18	dummy	39	0,61
9	foobar	73	1,14	19	moo	39	0,61
10	cat	65	1,01	20	12345	31	0,48

ных вопросов связан с категорией тестов B1xx. Это обусловлено большим числом жестко вписанных паролей в коде. На втором месте находятся вопросы, связанные с использованием небезопасных функций. На третьем месте группа детекторов возможных инъекций в программном коде.

Заметим, что согласно рейтингу OWASP Top 10, проблемные вопросы использования инъекций также находятся на третьей позиции (A03:2021). В то же время вопросы использования устаревших или ненадежных компонентов (A06:2021) находятся на шестой строчке по рейтингу OWASP. Эта категория может быть связана с блоком импортируемых библиотек, за который отвечают детекторы группы B4xx, которые, исходя из данных табл. 10, находятся на четвертой позиции.

Сравнивая лидирующие позиции рейтингов A01:2021 и B1xx, можно отметить, что связь также присутствует, поскольку вписанные пароли в коде могут привести к нарушениям контроля доступа. Полученные числовые результаты свидетельствуют о наличии тесных связей между рассмотренными рейтингами и дают ответ на вопрос В3, поставленный в начале работы.

Среди российских ресурсов, которые могут быть использованы в качестве источника данных уязвимостей для анализа безопасности, можно отметить банк данных угроз безопасности информации [31], однако он также опирается на базу данных каталога CWE. Так, по результатам, представленным на данном ресурсе (рис. 3), наибольшее распространение получила уязвимость CWE-119, связанная с ошибками переполнения буфера, которая преимущественно связана с языками C и C++. Описание уязвимостей, связанных с ошибками переполнения буфера для языка Python, можно найти в отчете [32].

Распределение на рис. 3 носит более общий характер. Тем не менее, можно заметить, что также широко распространены уязвимости, обусловленные инъекциями и некорректной обработкой входных данных.

Таблица 10

Показатели по категориям тестов

Категория	Всего значений	Доля от общего числа по категории, %	Всего без значений детектора B101	Доля от общего числа по категории, %
B1xx	148 557	83,31	13 858	31,76
B2xx	143	0,08	143	0,33
B3xx	11 683	6,55	11 683	26,78
B4xx	5927	3,32	5927	13,59
B5xx	751	0,42	751	1,72
B6xx	10 072	5,65	10 072	23,09
B7xx	1195	0,67	1195	2,74

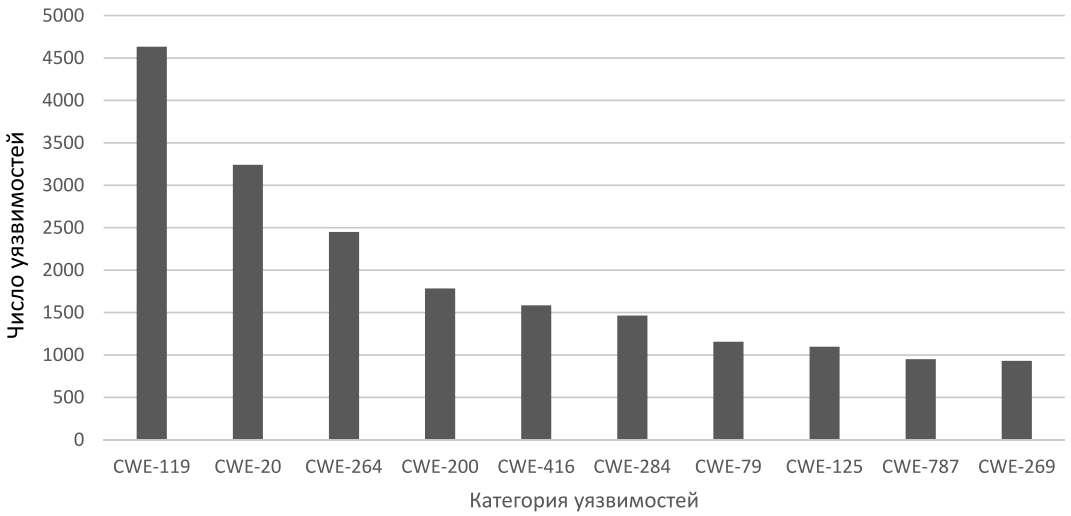

Рис. 3. Распределение по типам ошибок

Заключение

Представленные в статье результаты анализа свидетельствуют о том, что проблемные вопросы, связанные с написанием безопасного кода для приложений на Python, остаются открытыми. Многие библиотеки с открытым исходным кодом содержат достаточно важные уязвимости, множественные ошибки фильтрации данных, используют жестко вписанные пароли либо устаревшие версии криптографических библиотек.

Одним из путей решения задачи повышения качества программного кода и его безопасности может быть использование методов статического анализа и библиотек для поиска ошибок еще на этапе разработки. Эффективность таких анализаторов достаточно высока даже для больших проектов. В то же время, остается актуальным вопрос с ложно положительными срабатываниями модулей тестирования. Кроме того, необходимо дальнейшее развитие российских средств линтеринга и статического анализа, которые будут опираться на национальный каталог уязвимостей.

Отметим, что перспективным направлением исследования в области информационной безопасности является использование комбинированных методов. В частности, это относится к методам машинного обучения, статическим, латентно-семантическим методом и методам динамического анализа для проверки программного кода. Разработки в данных направлениях сопряжены с рядом трудностей, которые обусловлены отсутствием в открытом доступе качественных больших наборов данных для обучения.

Список литературы

1. **О техническом** регулировании. Федеральный закон Российской Федерации от 27 дек. 2002 г. № 184-ФЗ, ред. от 28.11.2018. URL: http://www.consultant.ru/document/ cons_doc_LAW_40241/

2. **О сертификации** средств защиты информации. Постановление Правительства Российской Федерации от 26 июня 1995 г. № 608, ред. от 21.04.2010. URL: http://www.consultant.ru/document/cons_doc_LAW_7054/

3. **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.

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

5. **Johnson S. C.** Lint, C Program Checker // COMP. SCI. TECH. REP. 1978. P. 78—90.

6. **Beller M., Bholanath R., McIntosh S., Zaidman A.** Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software // In Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016), Suita, IEEE, 2016. P. 470—481. DOI: 10.1109/SANER.2016.105.

7. **Chess B., McGraw G.** Static Analysis for Security // IEEE Security & Privacy, 2004. Vol. 2, Is. 6. P. 76—79. DOI: 10.1109/MSP.2004.111.

8. **Fromherz A.**, **Ouadjaout A., Mine A.** Static Value Analysis of Python Programs by Abstract Interpretation // In Proceedings of the 10th NASA Formal Methods International Symposium (NFM 2018), Springer, Newport News, 2018. P. 185—202. DOI: 10.1007/978-3-319-77935-5_14.

9. **Oyetoyan T. D., Milosheska B., Grini M., Cruzes D. S.** Myths and Facts about Static Application Security Testing Tools: An Action Research at Telenor Digital // In Proceedings of the 19th Conference on Agile Processes in Software Engineering and Extreme Programming (XP 2018), Porto, Springer, 2018. P. 86—103. DOI:10.1007/978-3-319-91602-6_6.

10. **Vassallo C.**, **Panichella S., Palomba F.** et al. How Developers Engage with Static Analysis Tools in Different Contexts // Empirical Software Engineering. 2020. Vol. 25, Is. 2. P. 1419—1457. DOI: 10.1007/s10664-019-09750-5.

11. **Smith J., Johnson B., Murphy-Hill E.** et al. How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis

Tool // IEEE Transactions on Software Engineering, 2019. Vol. 45. Is. 9. P. 877— 897. DOI: 10.1109/TSE.2018.2810116.
12. **OWASP** Web Security Testing Guide. URL: https://github.com/OWASP/wstg
13. **Common** Weakness Enumeration. URL: https://cwe.mitre.org/about/index.html
14. **CVE**. URL: https://cve.mitre.org/
15. **OWASP** Top 10 — 2021. URL: https://owasp.org/Top10/
16. **TIOBE** Index for March 2022. URL: https://www.tiobe.com/tiobe-index/
17. **Django** Software Foundation. Security in Django. URL: https://docs.djangoproject.com/en/3.0/topics/security/
18. **Xia X., He X., Yan Y.** et al. An Empirical Study of Dynamic Types for Python Projects // In Proceedings of the 8th International Conference on Software Analysis, Testing, and Evolution (SATE 2018), Springer, 2018. P. 85—100.
19. **Ruohonen J.** An Empirical Analysis of Vulnerabilities in Python Packages for Web Applications // In Proceedings of the 9th International Workshop on Empirical Software Engineering in Practice (IWESEP 2018). Nara, IEEE, 2018. P. 25—30.
20. **The** Python Package Index (PyPI) is a repository of software for the Python programming language. URL: https://pypi.org/
21. **Ruohonen J., Hjerpee 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 (online), IEEE, 2021. P. 1—10.
22. **Welcome** to the Bandit documentation! — Bandit documentation. URL: https://bandit.readthedocs.io/en/latest/
23. **A Collection** of Datasets for Big Code AnalysIs. URL: https://github.com/CUHK-ARISE/ml4code-dataset
24. **Secure,** Reliable, and Intelligent Systems Lab | SRI Group Website. URL: https://www.sri.inf.ethz.ch/
25. **Infowatch** Appercut**.** URL: https://www.infowatch.ru/products/appercut
26. **АК-ВС** 2. URL: https://npo-echelon.ru/production/65/4243
27. **pylint** 2.14.5 URL: https://pypi.org/project/pylint/
28. **Welcome** to mypy documentation! — Mypy 0.942 documentation. URL: https://mypy.readthedocs.io/en/stable/#
29. **Edmundson A., Holtkamp B., Rivera E.** et al. An Empirical Study on the Effectiveness of Security Code Review //In Proceedings of the 5th International Symposium on Engineering Secure Software and Systems (ESSoS 2013), Paris, Springer, 2013. P. 197— 212. DOI: 10.1007/978-3-642-36563-8_14.
30. **Hamann D.** Exploiting Python pickles. URL: https://davidhamann.de/2020/04/05/exploiting-python-pickle/
31. **Банк** данных угроз безопасности информации. URL: https://bdu.fstec.ru/vul
32. **Python**: List of security vulnerabilities. URL: https://www.cvedetails.com/vulnerability-list/vendor_id-10210/product_id-18230/opov-1/Python-Python.html

Static Analysis of the Source Code of Python Applications

D. A. Kapustin, kap-kapchik@mail.ru, **V. V. Shvyrov**, slsh@i.ua, **T. I. Shulika** shulika-tatyana@mail.ru, Lugansk State Pedagogical University, Lugansk, 91011, Lugansk People's Republic

Corresponding author:

Denis A. Kapustin, Associate Professor,
Lugansk State Pedagogical University, Lugansk, 91011, Lugansk People's Republic
E-mail: kap-kapchik@mail.ru

Received on July 08, 2022
Accepted on July 21, 2022

One of the popular methods of software code analysis is the static analysis method. This method allows not only to check the code for compliance with the language specification, but also to find potential vulnerabilities. The work performs a static analysis of a corpus of open source Python application. Using the Bandit library, statistical indicators of various categories of potential vulnerabilities are found, a rating table of vulnerabilities found in the studied data set is built. A qualitative analysis of threats is carried out according to their danger based on the CWE catalog data.

The purpose of this work is to analyze a corpus of open source Python listings for potential vulnerabilities using a static analyzer and rank threats based on the potential vulnerabilities found.

Thus, the study should answer the following questions:

Q1. What categories of potential vulnerabilities in the CWE catalog are most common in Python open source projects?

Q2. What categories of tests (checks) of the Bandit library found the most vulnerabilities?

Q3. Is there a correlation between the obtained statistical data and the OWASP rating data?

Keywords: *bandit, big data, CWE, linters, OWASP, python dataset, security analysis, static analysis, source code analysis, threat, vulnerability*

For citation:

Kapustin D. A., Shvyrov V. V., Shulika T. I. Static Analysis of the Source Code of Python Applications, *Programmnaya Ingeneria,* 2022, vol. 13, no. 8, pp. 394—403.

DOI: 10.17587/prin.13.394-403

References

1. **O tekhnicheskom** regulirovanii. Federal'nyj zakon Rossijskoj Federacii ot 27 dek. 2002 g. № 184-FZ, red. ot 28.11.2018, available at: http://www.consultant.ru/document/ cons_doc_LAW_40241/ (in Russian).
2. **O sertifikacii** sredstv zashchity informacii. Postanovlenie Pravitel'stva Rossijskoj Federacii ot 26 iyunya 1995 g. № 608, red. ot 21.04.2010, available at: http://www.consultant.ru/document/cons_doc_LAW_7054/ (in Russian).
3. **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, pp. 238—252.
4. **Allen F. E.** Control flow analysis, *ACM SIGPLAN Notices*, 1970, Vol. 5, Is. 7, July, pp. 1—19.
5. **Johnson S. C.** Lint, C Program Checker, *COMP. SCI. TECH. REP,* 1978, pp. 78—90.
6. **Beller M., Bholanath R., McIntosh S., Zaidman A.** Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software, *In Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016)*, Suita, IEEE, 2016, pp. 470—481. DOI: 10.1109/SANER.2016/105.
7. **Chess B., McGraw G.** Static Analysis for Security, *IEEE Security & Privacy*, 2004, vol. 2, is. 6, pp. 76—79. DOI: 10.1109/MSP.2004.111.
8. **Fromherz A.**, **Ouadjaout A., Mine A.** Static Value Analysis of Python Programs by Abstract Interpretation, *In Proceedings of the 10th NASA Formal Methods International Symposium (NFM 2018),* Springer, Newport News, 2018, pp. 185—202. DOI: 10.1007/978-3-319-77935-5_14.
9. **Oyetoyan T. D., Milosheska B., Grini M., Cruzes D. S.** Myths and Facts About Static Application Security Testing Tools: An Action Research at Telenor Digital, *In Proceedings of the 19th Conference on Agile Processes in Software Engineering and Extreme Programming (XP 2018)*, Porto, Springer, 2018, pp. 86—103, DOI: 10.1007/978-3-319-91602-6_6.
10. **Vassallo C., Panichella S., Palomba F.** et al. How Developers Engage with Static Analysis Tools in Different Contexts, *Empirical Software Engineering*, 2020. Vol. 25, is. 2, P. 1419—1457. DOI: 10.1007/s10664-019-09750-5.
11. **Smith J., Johnson B., Murphy-Hill E.** et al. How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis Tool, *IEEE Transactions on Software Engineering*, 2019, vol. 45, is. 9, pp. 877— 897. DOI: 10.1109/TSE.2018.2810116.
12. **OWASP** Web Security Testing Guide, available at: https://github.com/OWASP/wstg
13. **Common** Weakness Enumeration, available at: https://cwe.mitre.org/about/index.html
14. **CVE,** available at: https://cve.mitre.org/
15. **OWASP** Top 10 — 2021, available at: https://owasp.org/Top10/
16. **TIOBE** Index for March 2022, available at: https://www.tiobe.com/tiobe-index/
17. **Django** Software Foundation. Security in Django. available at: https://docs.djangoproject.com/en/3.0/topics/security/
18. **Xia X., He X., Yan Y.** et al. An Empirical Study of Dynamic Types for Python Projects, *In Proceedings of the 8th International Conference on Software Analysis, Testing, and Evolution (SATE 2018)*, Springer, 2018, pp. 85—100.
19. **Ruohonen J.** An Empirical Analysis of Vulnerabilities in Python Packages for Web Applications, *In Proceedings of the 9th International Workshop on Empirical Software Engineering in Practice (IWESEP 2018)*, Nara, IEEE, 2018, pp. 25—30.
20. **The Python** Package Index (PyPI) is a repository of software for the Python programming language, available at: https://pypi.org/
21. **Ruohonen J., Hjerpee 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 (online)*, IEEE, 2021, pp. 1—10.
22. **Welcome** to the Bandit documentation! — Bandit documentation, available at: https://bandit.readthedocs.io/en/latest/
23. **A Collection** of Datasets for Big Code Analysis, available at: https://github.com/CUHK-ARISE/ml4code-dataset
24. **Secure,** Reliable, and Intelligent Systems Lab | SRI Group Website, available at: https://www.sri.inf.ethz.ch/
25. **Infowatch** Appercut, available at: https://www.infowatch.ru/products/appercut (in Russian).
26. **AK-VS 2**, available at: https://npo-echelon.ru/production/65/4243 (in Russian).
27. **pylint** 2.14.5, available at: https://pypi.org/project/pylint/
28. **Welcome** to mypy documentation! — Mypy 0.942 documentation, available at: https://mypy.readthedocs.io/en/stable/#
29. **Edmundson A.**, **Holtkamp B., Rivera E.** et al. An Empirical Study on the Effectiveness of Security Code Review, *In Proceedings of the 5th International Symposium on Engineering Secure Software and Systems (ESSoS 2013)*, Paris, Springer, 2013, pp. 197— 212. DOI: 10.1007/978-3-642-36563-8_14.
30. **Hamann D.** Exploiting Python pickles, available at: https://davidhamann.de/2020/04/05/exploiting-python-pickle/
31. **Bank** dannyh ugroz bezopasnosti informacii, available at: https://bdu.fstec.ru/vul (in Russian).
32. **Python:** List of security vulnerabilities, available at: https://www.cvedetails.com/vulnerability-list/vendor_id-10210/product_id-18230/opov-1/Python-Python.html

ИНФОРМАЦИЯ