ISSN 2658-5014 (Print) СИИТ ISSN 2686-7044 (Online)
http://siit.ugatu.su 2026. Т. 8, № 3(27)
СИСТЕМНАЯ ИНЖЕНЕРИЯ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ
Научная статья

УДК 004.056 DOI: 10.54708/SIIT-2026-no3-p80 EDN: VEKTHF

Метод использования данных кибербразведки для поддержки принятия решений при управлении уязвимостями программного обеспечения

Р. Р. Сабитов • К. В. Миронов

Уфимский университет науки и технологий

В данной статье рассматривается разработка системы управления уязвимостями программного обеспечения с поддержкой принятия решений на основе сбора данных из открытых источников кибербразведки. Разработан прототип подхода для приоритизации уязвимостей VPSS (Vulnerability Propagation Scoring System), разработан механизм подтверждения информации с различных источников для снижения ложных срабатываний, а также система рекомендаций для управления обновлениями версий. Реализован прототип в виде бота, который демонстрирует возможность автоматизации анализа уязвимостей для различных экосистем таких как PyPI, npm и Maven.

Управление уязвимостями; кибербразведка; агрегация данных; автоматизация.

Введение

Как правило, при разработке программного обеспечения используются сторонние библиотеки и фреймворки [Над24, Орл23, Pru26]. Так, согласно данным, представленным в отчете Black Duck Open Source Software and Risk Analysis (OSSRA) [Суд26], среднее количество уязвимостей в открытом исходном коде за последний год удвоилось. Отчет Black Duck также показал, что внедрение программного обеспечения с открытым исходным кодом продолжает ускоряться, в то время как масштабы и сложность программного обеспечения с открытым исходным кодом, а также связанные с ним риски увеличиваются по всем показателям [Pap25]. Как видно из таблицы, открытый исходный код в настоящее время почти повсеместно используется в коммерческом программном обеспечении.

Таблица

Использование открытого исходного кода в коммерческих ПО [Суд26]

	Доля открытого исходного кода, используемого в программном компоненте	Среднее количество компонентов OSS для каждого приложения	Среднее количество файлов на кодовую базу	Максимальное количество проектов на кодовую базу
OSSRA 2025	97%	911	48415	21
OSSRA 2026	98%	1180	84499	50

Только 2% протестированных кодовых баз не содержали компонентов с открытым исходным кодом, и даже эта цифра, скорее всего, относится к специализированным устаревшим системам, а не к новым разработкам. С практической точки зрения это означает, что при

Сабитов Р. Р., Миронов К. В. Метод использования данных кибербразведки для поддержки принятия решений при управлении уязвимостями программного обеспечения // СИИТ. 2026. Т. 8, № 3(27). С. 80-90. DOI: 10.54708/SIIT-2026-no3-p80. EDN: VEKTHF.

Sabitov R. R., Mironov K. V. “A method of using cyber intelligence data to support decision-making in software vulnerability management” // SIIT. 2026. Vol. 8, no. 3(27), pp. 80-90. DOI: 10.54708/SIIT-2026-no3-p80. EDN: VEKTHF. (In Russian).

создании программного обеспечения активно используется открытый исходный код. Это влияет на безопасность приложений.

При обеспечении безопасности приложений специалисты все чаще используют инструменты анализа состава программного обеспечения (Software Composition Analysis, SCA), чтобы отслеживать уязвимые зависимости, однако у них есть ряд существенных ограничений. Рассмотрим эти ограничения:

1. *Различие источников данных.* Популярные инструменты, такие как OWASP (DC)[1], Snyk[2], GitHub Dependabot[3], npm[4], Eclipse Steady[5], используют разные базы уязвимостей, что приводит к значительным расхождениям в результатах сканирования одного и того же проекта.
2. *Высокий уровень ложных срабатываний.* Так в статье [Imt21] предоставлено исследование, согласно результатам которого девять популярных SCA инструментов демонстрируют расхождения от 17 до 332 уязвимостей для одного проекта Maven.
3. *Отсутствие приоритизации.* Представленные отчеты не демонстрируют уязвимости, которые необходимо устранить в первую очередь.
4. *Устаревание баз данных.* Базы уязвимостей имеют задержки при обновлении.

В работах [Imt21, Dan22, Zha19] производится анализ различных SCA инструментов, каждый из которых использует одну определенную базу уязвимостей, что считается существенным недостатком.

В данной работе была поставлена цель разработать многобазовое средство. Вся информация об уязвимостях собирается из технических баз данных, а также данных киберразведки, которые впоследствии формируют единый индекс риска для пользователя. В связи с вышеизложенным, данная работа направлена на создание системы управления уязвимостями на базе бота с функцией поддержки принятия решений на основе анализа данных из источников киберразведки. Для достижения этой цели необходимо решить следующие задачи:

- Реализовать агрегатор уязвимостей с таких источников, как OSV.Dev, GitHub, Synk, NVD и механизм достоверности данных.
- Определить метрику для выставления приоритета устранения.
- Разработать систему рекомендаций для управления версиями.
- Разработать систему рекомендаций на основе данных киберразведки.
- Протестировать работоспособность бота на тестовых конфигурационных файлах.

В условиях применения различных инструментов SCA и использование различных баз, возникают проблемы в точности распознавания значимых уязвимостей, они могут устраняться с опозданием, а менее значимым может уделяться больше времени.

Необходимо разработать бот, который мог бы производить анализ уязвимостей в зависимостях и иметь поддержку принятия решения. Бот должен принимать файлы с разных экосистем, извлекать пакеты и по ним производить агрегированный поиск уязвимостей. Найденные записи должны объединяться по идентификаторам CVE и GHSA, также должен учитываться уровень консенсуса источников и формирование профиля уязвимости.

Для повышения ценности, необходимо реализовать механизм приоритизации, позволяющий определить не только базовую критичность, но и использование зависимости [npm26].

Дополнительно боту необходимо обогащать результаты данными с киберазведки. Это позволит сформировать более точные рекомендации по срочности обновления.

---

[1] Dependency-check. https://owasp.org/www-project-dependency-check (дата обращения 12.05.2026).
[2] Snyk Open Source Security Management. https://support.snyk.io/hc/enus/articles/360000925438-What-does-Snyk-access-and-store-when-scanninga-project- (дата обращения 12.05.2026).
[3] Github Advisory Database. https://github.com/advisories (дата обращения 12.05.2026).
[4] npm Security Advisories. https://www.npmjs.com/advisories (дата обращения 12.05.2026).
[5] Eclipse Steady 3.1.14 (incubator project). https://eclipse.github.io/steady/about (дата обращения 12.05.2026).

Основной эффект заключается в сокращении времени анализа уязвимостей, увеличение полноты обнаружения и улучшения качества рекомендаций при планировании обновлений.

Дальнейший текст статьи организован следующим образом. В следующем разделе переходим к разбору существующих баз уязвимостей, а также проблему идентификации различных уязвимостей и приводим в нашей работе решение данной проблемы с помощью агрегации баз уязвимостей. После идентификации уязвимостей, чтобы определить, какую угрозу устранять первой переходим к реализации упрощенной метрики приоритета исправления. После того, как уязвимостям были выставлены приоритеты, переходим к разработке системы рекомендаций по управлению версиями. Также, чтобы подтвердить опасения по той или иной угрозе, вводим систему рекомендаций на основе данных кибербразведки, которая в свою очередь сможет указать нам на популярные и обсуждаемые уязвимости в интернет-пространстве с целью дальнейшего устранения.

Известная реализация агрегатора уязвимостей и достоверности данных

Современные экосистемы программного обеспечения сопровождаются множеством независимых каталогов уязвимостей, которые можно разделить на три группы: открытые базы уязвимостей, такие как OSV[6] и NVD[7]; коммерческие, как Snyk[8]; платформенные, как GitHub Security Advisories[9]. Одна и та же программная ошибка может быть зарегистрирована под разными глобальными идентификаторами. Так в статье [Imt21] показано, что инструменты (SCA) анализа состава ПО по-разному сообщают об уязвимостях для одних и тех же зависимостей, что в конечном итоге затрудняет «картину» риска.

Исходя из вышесказанного, в рассматриваемом боте эта проблема решается методом агрегации. Внешние API-интерфейсы опрашиваются последовательно, результаты нормализуются, затем сравниваются по идентификаторам, после чего формируется окончательный список объектов со списком источников и производных показателей, который включает уровень консенсуса и текстовые рекомендации.

Термины и определения. Для того, чтобы продолжить ознакомление с реализацией агрегации, разберем понятия:

Уязвимость – в контексте каталогов это зарегистрированное описание дефекта (ошибка в коде или конфигурации), который при определенных условиях позволяет нарушить ожидаемые свойства безопасности в системы, использующий затронутый компонент.

Пакет – это файл, который представляет собой какой-то инструмент или библиотеку

Экосистема – это комплекс инструментов, которые обеспечивают управление файлами. Экосистемами являются, например, Python [Par23], npm, Maven.

Иде*нтификаторы*. На практике одна логическая уязвимость может иметь:

- *CVE*[10] – это уникальный, общий идентификатор для публично известных уязвимостей в программных пакетах, находящихся в публичном доступе в формате «CVE-год-номер»
- *GHSA-идентификатор* – это уникальный идентификатор рекомендации по безопасности в базе данных GitHub Advisory Database.
- *Идентификатор в формате OSV* — это уникальный код, который используется для идентификации конкретной уязвимости в базе данных.

Агрегация – это процесс объединения разрозненных данных из различных источников в единый, более значимый набор.

---

[6] https://osv.dev (дата обращения 12.05.2026).
[7] https://nvd.nist.gov (дата обращения 12.05.2026).
[8] https://docs.snyk.io/scan-with-snyk/snyk-open-source/manage-vulnerabilities/snyk-vulnerability-database (дата обращения 12.05.2026).
[9] https://github.com/advisories (дата обращения 12.05.2026).
[10] What is a CVE? (2022). https://www.securitylab.ru/analytics/532385.php (дата обращения 12.05.2026).

Архитектура модуля представлена на рис. 1.

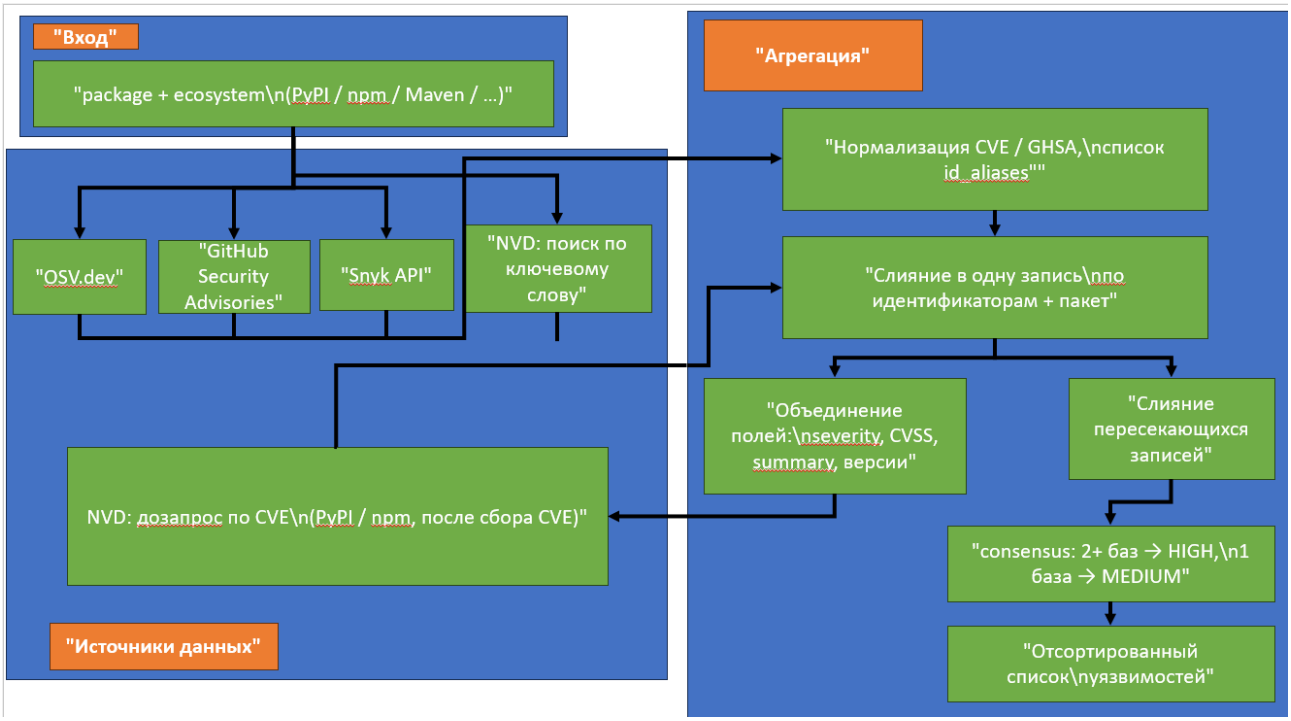

Рис. 1 Архитектура модуля

Агрегация данных. Когда на вход поступает имя пакета и экосистема, для начала определяется, какие базы уязвимостей стоит опрашивать. Для экосистем PyPI, npm последовательно опрашиваются OSV [Adi26], GitHub Security Advisories [Git26] и при наличии ключа Snyk [Sny26]. Для Maven в приоритете OSV и официальный каталог NVD [Nat26] с поиском по ключевым словам. Для прочих экосистем по умолчанию используется OSV. После отправки запросов получаем первоначальный набор записей с разных баз, которые описали разные, либо одинаковые уязвимости в своих форматах.

После каждый ответ преобразуется в единую модель. Происходит нормализация идентификаторов в основном CVE и GHSA. Это делается с целью сопоставить записи с разных баз.

Далее выполняется слияние. Если несколько записей относятся к одной уязвимости по совпадению нормализованных идентификаторов при том же пакете, то они объединяются.

При объединении список источников дополняется, также в основном берется информация, где уровень серьезности и CVSS имеет более высокий индекс и имеется более информативное описание.

После того, как с других баз были собраны записи с известными CVE, по этим идентификаторам производится точечный дозапрос в NVD. Данное действие позволяет подтянуть официальные метрики и формулировки по найденной уязвимости, а также снизить ложные срабатывания.

Далее производится слияние по второму кругу. Это необходимо для проверки записей, Bon25которые могли бы получить разные ключи, но иметь общие псевдонимы.

На выходе для каждой записи вычисляется уровень согласованности источников. Если одна и та же уязвимость подтверждена двумя и более независимыми источниками из списка, уровень помечается, как High, если одно, то Medium, иначе Low. Итоговый список сортируется сначала по серьезности (от критической до низкой), затем по числу источников по убыванию, чтобы в отчете выше оказались наиболее опасные уязвимости.

Метрика VPSS

В статье [Bon25] авторы подняли проблему отсутствия метрик для количественной оценки влияния уязвимостей на цепочки поставок программного обеспечения. И для решения данной проблемы авторы предложили систему оценки распространения уязвимостей (Vulnerability Propagation Scoring System, VPSS), теоретико-графический динамический индикатор, предназначенный для количественной оценки воздействия уязвимостей в цепочках поставок программного обеспечения. Данная система была разработана с учетом как широты, так и глубины распространения уязвимостей. VPSS имеет тот же диапазон оценок (от 0 до 10) и уровни влияния (низкий, средний, высокий и критический), что и CVSS, что делает его простым для понимания и использования.

Основываясь на данной статье, в нашей работе был использован упрощенный подход VPSS. Выглядит она следующим образом:

$$\mathrm{VPSS} = \mathrm{CVSS} \times (1 + (\mathrm{depth} - 1) \times 0.2) \times \mathrm{frequency},$$

где depth – глубина зависимости, $\mathrm{frequency}$ – частота использования.

Рассмотрим из чего формируется показатель VPSS. Для начала для каждой пары «уязвимость – зависимость» используются: оценка CVSS [Ben17] из агрегатора (при отсутствии значения выставляется нейтральное значение 5.0), глубина зависимости depth (для прямой зависимости выставляется значение 1, а для транзитивных это значение может увеличиваться) и коэффициент частоты использования frequency из модели зависимости (по умолчанию равняется единице). Система пытается понять, к какой зависимости из файла относится каждая найденная уязвимость. Для этого она сравнивает имена пакетов и, если имя из уязвимости и имя из списка зависимостей совпадают, то уязвимость «цепляется» к этой зависимости. Если не одна зависимость не подошла, все равно нужно просчитать риск. Тогда система притворяется, что есть зависимость с тем же именем пакета, что в уязвимости, а экосистему помечает, как «неизвестно». Это служит «заглушкой», чтобы расчет VPSS произвелся, пусть и менее точными данными.

Таким образом, при прямой зависимости глубинный множитель равен единице, а при увеличении транзитивной зависимости приоритет уязвимости плавно возрастает.

Ранжирование и представление. Все найденные уязвимости сортируются по убыванию VPSS. Для каждой уязвимости присваивается порядковый ранг приоритета, который покажет какую уязвимость необходимо будет устранить в первую очередь (рис. 2).

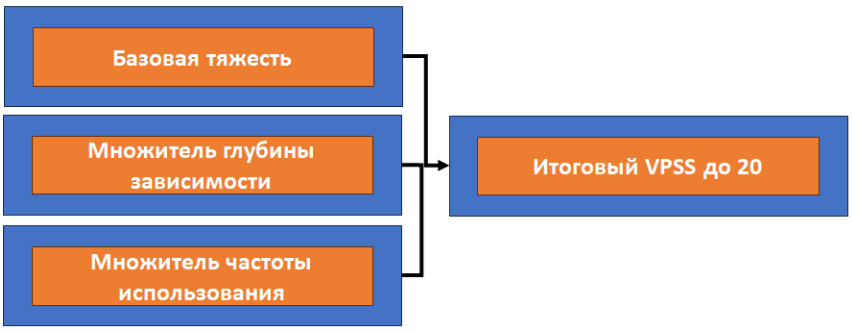

Рис. 2 Построение VPSS

Система рекомендаций для управления версиями

Современная разработка на Node.js, Python и Java опирается на декларативное описание зависимостей. Перечень пакетов и ограничений на допустимые версии передаются в менеджер

пакетов, который строит граф и подтягивает артефакты из репозиториев. В плане безопасности критичны два аспекта. Первым является наличие уязвимостей в выбранных версиях, вторым способ фиксации версии [Imr25].

В данном прототипе бота при анализе конфигурационного файла блок рекомендаций по версиям не выполняет работу менеджера зависимостей. Его задача иная по уже извлеченному из конфигурационного файла зависимостей предоставить человеку удобочитаемую следующую информацию: что считается рискованным закреплением, что приемлемой гибкостью и насколько логична формулировка как следующий шаг.

Таким образом, система рекомендаций работает на уровне эвристик над типом ограничения (constraint), а не на уровне полного семантического сравнения всех совместимых версий в реестре пакетов.

Связь с остальными подсистемами следующая. Модуль парсинга преобразует текст файла в список объектов «зависимость». Модуль оценки рисков для каждой уязвимости, сопоставленной с зависимостью, добавляет структурированную рекомендацию по версиям. Модуль форматирования отчета дополнительно формирует обзорный раздел по всем зависимостям с жестким pinning. В совокупности это дает и точечные подсказки в контексте с конкретной CVE и агрегированную картину по файлу целиком.

Классификация ограничений. Проект определяет зависимости с помощью структуры Dependency, включающей имя, версию (или её часть), экосистему и тип ограничения (constraint_type). Вместо полного разбора всех форматов пакетов, используется упрощенный подход: для requirements.txt — регулярные выражения по строкам, для package.json — анализ префиксов версий (^, ~, >=), а для Maven — выделение явных привязок и других случаев.

Constraint_type классифицирует способы указания версий, например:

- pinning: точное совпадение версии (например, requests==2.28.0);
- range: интервал версий;
- floating-min: нижняя граница без верхнего предела (аналогично >=);
- compatible: семантика совместимости (похоже на ~ в npm);
- floating-max, no-version, unknown (если строка не соответствует шаблонам).

Эта классификация намеренно упрощена, отражая политику объявления зависимостей.

В рамках данной статьи этого достаточно, поскольку: при pinning основной риск — использование уязвимой минорной версии без явного обновления. При использовании диапазонов и нижних границ, обновления в рамках политики менеджера пакетов могут безопасно подтягивать патчи, не затрагивая основную версию. Рекомендации в коде реализуют эту логику через простые условия (рис. 3).

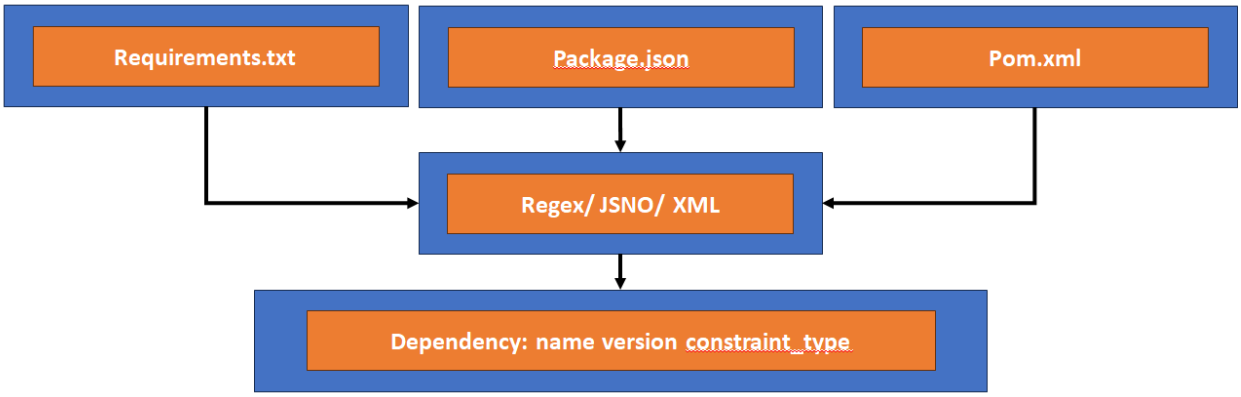

Рис. 3 Классификация ограничений

Логика рекомендаций в RiskEngine. Метод get_version_recommendation в классе RiskEngine отвечает за предоставление рекомендаций по управлению версиями зависимостей.

Суть метода не заключается в подборе безопасной версии пакета, ее задача заключается в осмотре зависимости, записанной в файле, и в выборе одной из предложенных веток поведения.

Если версии нет, метод выбирает статус, что информации недостаточно и выдает рекомендацию, что необходимо указать версию пакета. В данном выборе отражается идея, что без версии тяжело что-либо рекомендовать.

Если тип ограничения является «pinning» тогда совет формулируется как переход к новой версии, но в рамках следующей мажорной версии. То есть из текущей версии берется первый номер мажорной ветки, к нему прибавляется единица и в рекомендации получается шаблон «не ниже текущей, но строго ниже мажорного релиза»

Если тип ограничения является «floating-min» или «range», метод буде считать это хорошей практикой с точки зрения обновления. Рекомендация предложит «оставить все как есть» поскольку версия пакета находится в зоне стабильности.

Во всех остальных случаях применяется мягкая рекомендация, который предлагает не опускаться ниже указанной версии. Это является запасной веткой для тех, кого нельзя было отнести к «pinning» и «floating-min, range».

Таким образом, «система рекомендаций» в узком смысле — это набор if-elif правил над дискретным признаком (архитектура иллюстрируется на рис. 4).

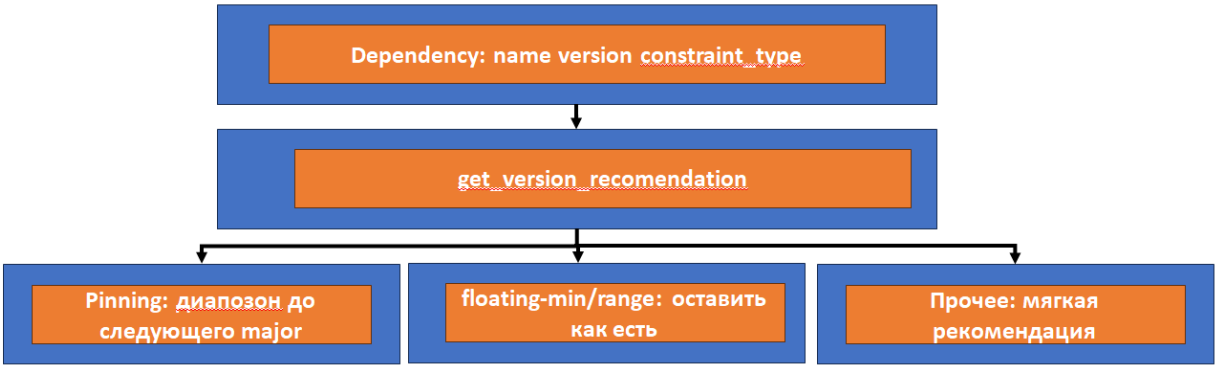

Рис. 4 Логика рекомендаций

СОЗДАНИЕ СИСТЕМЫ РЕКОМЕНДАЦИЙ НА ОСНОВЕ ДАННЫХ КИБЕРРАЗВЕДКИ

В данном боте кибербразведка считается дополнительным слоем, в котором осуществляется сбор информации из новостей, форумов по найденным CVE. Официальные базы уязвимости дают информацию по уязвимости, в каком пакете, какая у нее серьезность. Но кибербразведка дополняет иное. Она показывает, обсуждается ли уязвимость в индустрии, есть ли публичные экспойты, насколько популярна на форумах определенная CVE.

Для начала разберем, какие источники кибербразведки были задействованы:

- четыре RSS/Atom-ленты (The Hacker News, BleepingComputer, CVEFeed, SecurityLab.ru);
- Reddit (r/netsec);
- Computer Incident Response Center Luxembourg (CIRCL);
- Red Hat Security Data.

Также стоит сказать, что между запросами к внешним API есть небольшие паузы, это делается с целью не быть заблокированным.

Теперь перейдем к построению рекомендаций по результатам.

1. Рассмотрим более подробно логику, которая позволяет повысить заметность уязвимости. В нем реализованы два варианта.

- Первый вариант «Широта по типам источника». Имеется три семейства данных кибербразведки: RSS-группа, Reddit, VulnLookup. Если из этих трех семейств CVE упоминается

в двух и больше (например, в RSS и Reddit, или в Reddit и VulnLookup, или в RSS и VulnLookup, или все три). Это будет говорить о том, что данная CVE заметна на разных типов площадок и на нее стоит обратить внимание.

- Второй вариант это упоминание в новостях и Reddit. Здесь считается число ссылок по найденной CVE. И если таких ссылок больше двух, то это будет говорить о том, что CVE всплывает в потоке новостей и обсуждений.

2. Если только VulnLookup, то появляется текст, который говорит о дополнительном контексте в CIRCL и приоритет по CVSS/системам/наличию фикса.
3. Если по источникам OSINT не подтвердился, но при этом имеет высокий консенсус, то при высокой серьезности усиливается мотивация к обновлению.
4. Если запись не имеет CVE, то упор строится на консенсусе и наличии фикса.
5. В остальных случаях сочетается серьезность, CVSS, консенсус Medium и общие фразы про обновление.

Тестирование работоспособности бота

Для иллюстрации полного конвейера анализа был подготовлен конфигурационный файл зависимостей формата package.json. Данный файл является демопроектом, это можно понять по «version» где все значения нули. Также в нем можно увидеть пакеты, они указаны в кавычках, например, "body-parser", "ejs" и др. Теперь перейдем к рассмотрению префиксов, указанных в листинге. Знак “~” подразумевает в себя, что разрешены обновления только последней цифры, то есть получение патча, но в рамках одной минорной версии. Например, пакет "body-parser": "~1.13.2" допускает версии "~1.13.x", где x — любое выше число, но не допускает версию "~1.14.0". Знак “^” говорит о том, что разрешены обновления минорной версии и патч, но нельзя использовать мажорную версию. Например, в листинге представлен пакет "ejs": "^2.4.2". При обновлении пакет допускает версии 2.x.y, где x, y – любые числа, но не допускает версию "^3.0.0" Содержание файла показано на листинге.

Листинг

```
{
   "name": "vulnerable-node-source",
   "version": "0.0.0",
   "private": true,
   "scripts": {
      "start": "node ./bin/www"
   },
   "dependencies": {
      "body-parser": "~1.13.2",
      "cookie-parser": "~1.3.5",
      "debug": "~2.2.0",
      "ejs": "^2.4.2",
      "ejs-locals": "^1.0.2",
      "express": "~4.13.1",
      "express-session": "^1.13.0",
      "log4js": "^0.6.36",
      "morgan": "~1.6.1",
      "pg-promise": "^4.4.6",
      "serve-favicon": "~2.3.0"
   }
}
```

После проведения анализа можно ознакомиться с результатами. Вывод отчета представлен на рис. 5. Бот нашел уязвимости, построил их уровню важности, рассчитал показатель VPSS, предоставил источники, где была найдена уязвимость, рекомендации по исправлению, а также версию, где данная уязвимость устранена, также указал ссылки на источники по найденной

уязвимости, где можно увидеть, где упоминалась та или иная уязвимость. Также рекомендации по ограничениям версии.

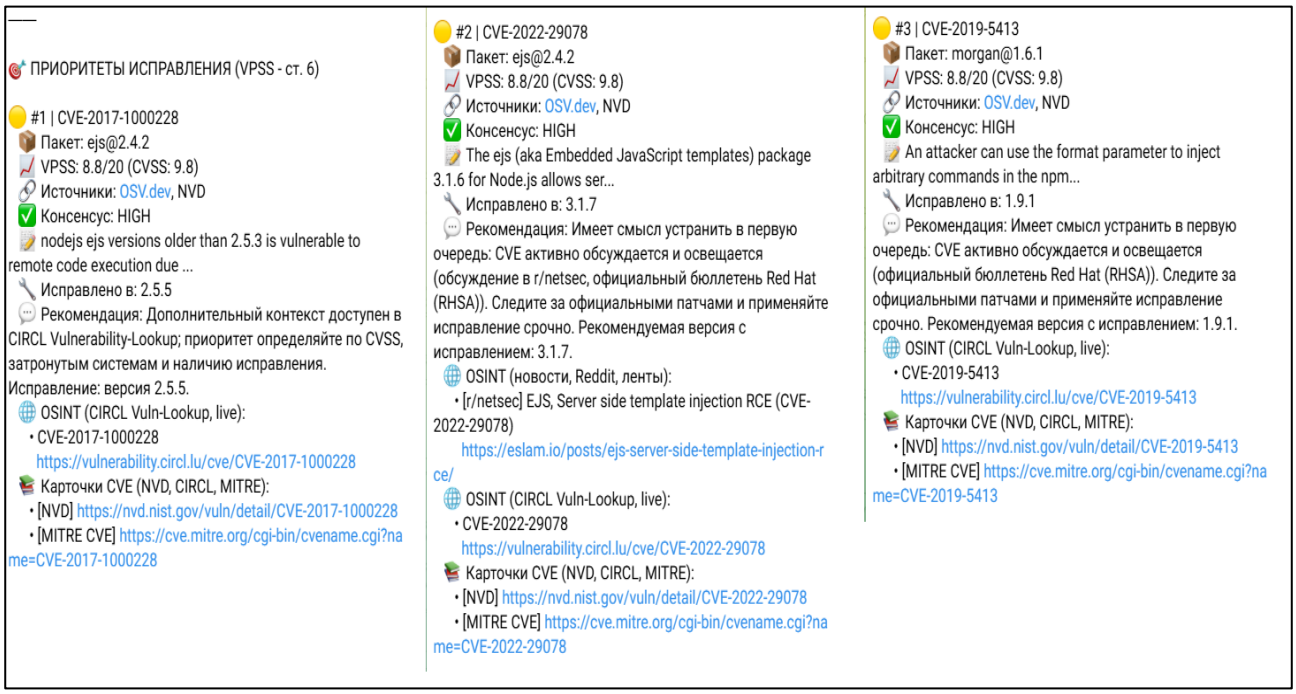

Рис. 5 Вывод отчета в боте

Пример рекомендации по обновлению представлен на рис. 6. В данном примере поскольку чуть выше были разобраны префиксы, которые подразумевает в себя обновления, бот основываясь на предоставленные данные, описал, что все зависимости являются безопасными.

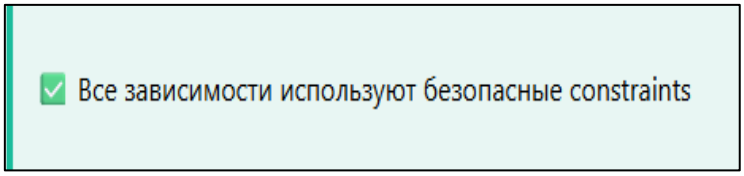

Рис. 6 Рекомендация по управлению версиями

Чтобы убедиться в работе рекомендаций по обновлению был загружен другой файл с фиксированными версиями. На рис. 7. Предоставлены результаты сканирования. В отчете можно увидеть, что были найдены зависимости с фиксированными версиями, и бот рекомендует их обновить.

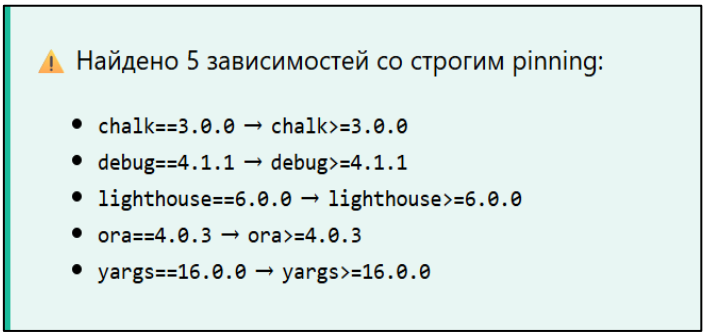

Рис. 7 Рекомендации по управлению версиями

Заключение

В данной работе была представлена система управления уязвимостями на основе бота с функцией поддержки принятия решений на основе анализа данных из источников киберразведки. Реализован агрегатор уязвимостей с таких источников, как OSV.Dev, GitHub, Synk, NVD и механизм достоверности данных. Добавлена метрика VPSS для выставления приоритета устранения. Создана система рекомендаций для управления версиями. Создана система рекомендаций на основе данных киберразведки. Протестирована работоспособность бота на тестовых конфигурационных файлах. Результаты тестирования показали, что бот позволяет отслеживать уязвимости, а именно выстраивать их по приоритету исправления, рассчитывать VPSS, находить и выводить источники по найденной уязвимости, также выводить версию, где данная уязвимость устранена и ссылки, где освещалась в интернете.

СПИСОК ЛИТЕРАТУРЫ | REFERENCES

[Adi26] A distributed vulnerability database for Open Source [Electronic resource]. 2026 https://osv.dev (accessed 04/11/2026).

[Ben17] Ben Othmane L., Chehrazi G., et al. Time for addressing software security issues: prediction models and impacting factors // Data Sci. Eng. 2, 107–124 (2017). 10.1007/s41019-016-0019-8. GCAVFS.

[Bon25] Bonan Ruan Zhiwei Lin Jiahao Liu Chuqi Zhang Kaihang Ji Zhenkai Liang. “Propagation-Based Vulnerability Impact Assessment for Software Supply Chains” 09.10.2025. arxiv.org/pdf/2506.01342.

[Dan22] Dann A., Plate H., et al. (2022). Identifying Challenges for OSS Vulnerability Scanners - A Study & Test Suite // IEEE Trans. Software Eng. 48. 3613-3625. 10.1109/TSE.2021.3101739.

[Git26] GitHub Advanced Security Database [Electronic resource]. 2026. https://docs.github.com/ru/get-started/learning-about-github/about-github-advanced-security (accessed 04/11/2026).

[Imr25] Imranur Rahman, Jill Marley, William Enck, Laurie Williams. ”Which Is Better For Reducing Outdated and Vulnerable Dependencies: Pinning or Floating?” 23.10.25. arxiv.org/html/2510.08609v2.

[Imt21] Imtiaz N., Thorn S., Williams L. (2021). A comparative study of vulnerability reporting by software composition analysis tools // ESEM'21: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 1-11. 10.1145/3475716.3475769.

[Nat26] National Vulnerability Database [Electronic resource]. 2026. https://nvd.nist.gov (accessed 04/11/2026).

[npm26] npm-audit [Electronic resource]. 2026. https://docs.npmjs.com/cli/v9/commands/npm-audit (accessed 04/11/2026).

[Pap25] Papotti A., Tuma K., Massacci F. (2025). On the effects of program slicing for vulnerability detection during code inspection // Empirical Software Engineering. 30. 10.1007/s10664-025-10636-y.

[Par23] Paramitha R., Massacci F. (2023). Technical leverage analysis in the Python ecosystem // Empirical Software Engineering. 28. 10.1007/s10664-023-10355-2.

[Pru26] Prutzkow A. V. “What is a good program? A space way” // SIIT. 2026. Vol. 8, no. 2(26), pp. 29 37. HGXVAJ.

[Sny26] Snyk Security Database [Electronic resource]. 2026. https://security.snyk.io (accessed 04/11/2026).

[Zha19] Zhang M., de Carné de Carnavalet X, et al. (2019). Large-scale empirical study of important features indicative of discovered vulnerabilities to assess application security // IEEE Transactions on Information Forensics and Security. 10.1109/TIFS.2019.2895963.

[Над24] Надеев С. А. Автоматизированный инструмент для рефакторинга логов в программном обеспечении // СИИТ. 2024. Т. 6, № 2(17). С. 72-77. OWVACS. [[Nadeev S. A. Automated tool for refactoring logs in software // SIIT. 2024. Vol. 6, No. 2(17). P. 72-77. (In Russian).]]

[Орл23] Орлов Г. О. Подход к обеспечению безопасности программного кода в веб-ориентированной среде // СИИТ. 2023. Т. 5, № 5(14). С. 68-77. HPCIMR. [[Orlov G. O. Approach to ensuring the security of software code in a web-oriented environment // SIIT. 2023. Vol. 5, No. 5(14). P. 68-77. (In Russian).]]

[Суд26] Сударев А. Уязвимости открытого исходного кода удваиваются по мере роста Ai [Элекр. ресурс]. 06.03.2026. https://itshaman.ru/news/security/uyazvimosti-otkrytogo-iskhodnogo-koda-udvaivayutsya-po-mere-rosta-kodirovaniya-ii. [[Sudarev A. Open Source Vulnerabilities Double as AI Grows. 03/06/2026. (In Russian).]]

ОБ АВТОРАХ | ABOUT THE AUTHORS

САБИТОВ Руслан Ринатович
Уфимский университет науки и технологий, Россия.
rusya.sabitov.03@mail.ru ORCID: 0009-0006-6117-1980.
Магистрант каф. вычислительной техники и защиты информации.

МИРОНОВ Константин Валерьевич
Уфимский университет науки и технологий, Россия.
mironovconst@gmail.com ORCID: 0000-0002-4828-1345.
Доц. каф. вычислительной техники и защиты информации. Дипл. спец. по защите инф-и (Уфимск. гос. авиац. техн. ун-т, 2012). PhD (Техн. ун-т Вены, 2016). Иссл. в обл. робототехники, применения ИИ в техн. системах.

SABITOV Ruslan Rinatovich
Ufa University of Science and Technology, Russia.
rusya.sabitov.03@mail.ru ORCID: 0009-0006-6117-1980.
Master's student of the Department of Computer Science and Information Security.

MIRONOV Konstantin Valeryevich
Ufa University of Science and Technology, Russia.
mironovconst@gmail.com ORCID: 0000-0002-4828-1345.
Assoc. Prof., Department of Computer Science and Information Security. Information Security Specialist (Ufa State Aviation Tech. Univ., 2012). PhD (Vienna University of Technology, 2016). Research in the field of robotics and intelligent control.

МЕТАДАННЫЕ | METADATA

Заглавие: Метод использования данных киберразведки для поддержки принятия решений при управлении уязвимостями программного обеспечения

Аннотация: В данной статье рассматривается разработка системы управления уязвимостями программного обеспечения с поддержкой принятия решений на основе сбора данных из открытых источников киберразведки. Разработан прототип подхода для приоритизации уязвимостей VPSS (Vulnerability Propagation Scoring System), разработан механизм подтверждения информации с различных источников для снижения ложных срабатываний, а также система рекомендаций для управления обновлениями версий. Реализован прототип в виде бота, который демонстрирует возможность автоматизации анализа уязвимостей для различных экосистем таких как PyPI, npm и Maven..

Ключевые слова: Управление уязвимостями; киберразведка; агрегация данных; автоматизация.

Язык: Русский.

Статья поступила в редакцию 15 июля 2025 г.

Title: A method of using cyber intelligence data to support decision-making in software vulnerability management.

Authors: Sabitov R. R., Mironov K. V.

Abstract: This article discusses the development of a software vulnerability management system with decision-making support based on data collection from open sources of cyber intelligence. A prototype approach for prioritizing VPSS vulnerabilities (Vulnerability Propagation Scoring System) has been developed, a mechanism for confirming information from various sources has been developed to reduce false positives, as well as a recommendation system for managing version updates. A prototype has been implemented in the form of a bot, which demonstrates the possibility of automating vulnerability analysis for various ecosystems such as PyPI, npm and Maven.

Key words: Vulnerability management; cyber intelligence; aggregation data; automation.

Language: Russian.

The article was received by the editors on 15 July 2025.