УДК 004.056

Давыдов В. В.

Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича, студент, 6 курс, Россия, г. Санкт-Петербург

Егорова П.Ю.

Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича, студент, 4 курс, Россия, г. Санкт-Петербург

Козырев В.С.

Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича, студент, 6 курс, Россия, г. Санкт-Петербург

МЕТОДИКА КОНТЕКСТНОГО ОБОГАЩЕНИЯ ПРОГРАММНОЙ ВЕДОМОСТИ ДАННЫМИ ОБ УЯЗВИМОСТЯХ И ФОРМИРОВАНИЯ VEX-ЗАЯВЛЕНИЙ

Аннотация: *В статье рассмотрена проблема ограниченной практической применимости традиционных подходов к использованию программной ведомости в процессах управления уязвимостями. Показано, что программная ведомость (SBOM) обеспечивает машиночитаемое описание состава программного продукта, однако в типовом виде не позволяет надежно определить, какие из выявленных уязвимостей действительно применимы и эксплуатируемы в контексте конкретного приложения, его*

архитектуры и конфигурации. Проведен анализ форм представления SBOM, ее связи с документами класса VEX, а также ограничений современных инструментов генерации и обогащения программных ведомостей. На этой основе предложена методика контекстного обогащения SBOM, включающая генерацию ведомости, сопоставление компонентов с внешними источниками сведений об уязвимостях, построение векторного представления кодовой базы, извлечение релевантного контекста и формирование VEX-совместимых утверждений с использованием генеративной модели. Для апробации методики развернут экспериментальный стенд на основе Qdrant, Dependency-Track, генераторов SBOM и локального контура векторизации. Проверка выполнена на проектах с воспроизводимыми сценариями уязвимостей, включая демонстрационный проект с Log4Shell и OWASP Juice Shop. Полученные результаты показывают, что предложенный подход позволяет связывать уязвимости прямых зависимостей с конкретными фрагментами исходного кода и формировать обоснованные VEX-заявления, однако испытывает затруднения при анализе транзитивных зависимостей, для которых требуется более глубокое восстановление контекста использования. Сделан вывод о практической применимости методики как средства снижения доли ложноположительных срабатываний и повышения информативности программной ведомости в процессах безопасной разработки.

Ключевые слова: *программная ведомость, SBOM, VEX, уязвимости, программная цепочка поставок, эксплуатабельность уязвимостей, векторные базы данных, генеративные модели, информационная безопасность*

Davydov V. V.

St. Petersburg State University
of Telecommunications named after Prof. M. A. Bonch-Bruevich,
student, 6th year, Russia, St. Petersburg
Egorova P.Y.
St. Petersburg State University
of Telecommunications named after Prof. M. A. Bonch-Bruevich,
student, 4th year, Russia, St. Petersburg
Kozyrev V.S.
St. Petersburg State University
of Telecommunications named after Prof. M. A. Bonch-Bruevich,
student, 6th year, Russia, St. Petersburg

THE METHODOLOGY OF CONTEXTUAL ENRICHMENT OF THE PROGRAM STATEMENT WITH VULNERABILITY DATA AND THE FORMATION OF VEX STATEMENTS

Anotation: *The paper addresses the limited practical applicability of traditional approaches to using a software bill of materials in vulnerability management processes. It is shown that SBOM provides a machine-readable description of software composition, but in its typical form it does not reliably determine which identified vulnerabilities are actually applicable and exploitable in the context of a particular application, its architecture, and configuration. The study analyzes SBOM representation forms, its relationship with VEX-class documents, and the limitations of current tools for generating and enriching software bills of materials. On this basis, a methodology for context-aware SBOM enrichment is proposed. It includes SBOM generation, component matching against external vulnerability sources, construction of a vector representation of the code base,*

retrieval of relevant context, and generation of VEX-compatible assertions using a generative model. To validate the methodology, an experimental testbed was deployed using Qdrant, Dependency-Track, SBOM generators, and a local vectorization pipeline. The methodology was tested on projects with reproducible vulnerability scenarios, including a Log4Shell demonstration project and OWASP Juice Shop. The results show that the proposed approach can link vulnerabilities of direct dependencies to specific source-code fragments and generate reasoned VEX statements, but encounters difficulties when analyzing transitive dependencies, for which deeper reconstruction of the usage context is required. The paper concludes that the methodology is practically applicable as a means of reducing false positives and increasing the informativeness of software bills of materials in secure software development processes.

Key words: *software bill of materials, SBOM, VEX, vulnerabilities, software supply chain, vulnerability exploitability, vector databases, generative models, information security*

Введение

Рост доли внешних библиотек, фреймворков и транзитивных зависимостей в современных программных продуктах привел к тому, что задача прозрачности программного состава стала одним из центральных направлений обеспечения безопасности цепочки поставок программного обеспечения. В этой логике программная ведомость рассматривается как базовый артефакт, фиксирующий компоненты, версии, поставщиков и зависимости, входящие в состав программного продукта [1–3]. Однако практическое использование SBOM быстро выявляет принципиальное ограничение: наличие ведомости еще не означает, что организация получает достаточные основания для корректной оценки риска.

Основная проблема состоит в том, что типовое обогащение SBOM сведениями из банков данных уязвимостей опирается преимущественно на формальную корреляцию идентификаторов компонентов и версий. Такой подход позволяет получить перечень потенциально затронутых уязвимостей, но не дает надежного ответа на вопрос о том, действительно ли соответствующая уязвимость применима к конкретному приложению, достигается ли уязвимый функционал в его кодовой базе и существует ли реалистичный путь эксплуатации. В результате значительная часть найденных уязвимостей требует дополнительной ручной интерпретации, а сам процесс управления рисками перегружается ложноположительными срабатываниями.

В современной практике для преодоления этого ограничения SBOM все чаще рассматривается совместно с документами класса VEX, которые предназначены для фиксации статуса уязвимости применительно к конкретному продукту [3–5]. Если SBOM описывает состав, то VEX выражает контекстную применимость уязвимости и тем самым делает результаты анализа существенно более пригодными для практического использования. Следовательно, центральной задачей становится переход от статического перечня компонентов к методике, способной связать уязвимость с реальным использованием соответствующего компонента в кодовой базе и на этой основе сформировать обоснованное VEX-заявление.

Цель статьи состоит в разработке и апробации методики контекстного обогащения программной ведомости данными об уязвимостях и оценками их эксплуатабельности. Научная новизна работы заключается в объединении средств генерации SBOM, платформ компонентного анализа, векторного представления кодовой базы и генеративной модели в единый воспроизводимый конвейер, ориентированный на формирование VEX-совместимых утверждений.

Программная ведомость и VEX в архитектуре анализа уязвимостей

С методической точки зрения программная ведомость представляет собой не просто перечень компонентов, а структурированное описание состава программного продукта, пригодное для автоматизированного анализа [1–3]. При этом для корректного использования ведомости необходимо учитывать, что она может существовать в разных формах представления и относиться к различным стадиям жизненного цикла программного обеспечения.

По структуре представления целесообразно различать плоскую, иерархическую и реляционную формы SBOM. Плоская форма удобна для обмена и ручного анализа, но ограничена в выражении зависимостей, прежде всего транзитивных. Иерархическая форма позволяет отразить дерево или граф зависимостей и тем самым точнее привязать уязвимость к цепочке достижения компонента внутри продукта. Реляционная форма делает возможной агрегацию, аналитическую обработку и последующий экспорт данных в более сложные контуры управления рисками. Следовательно, богатство структурного представления напрямую влияет на то, насколько обоснованно можно связать сущности уровня «продукт — компонент — уязвимость».

Не менее важным является различение форм SBOM по стадии жизненного цикла. Ведомости проектирования и разработки полезны для ранней фильтрации зависимостей и предварительной оценки рисков, но не отражают фактический состав поставляемого продукта. Наибольшее значение для VEX, как правило, имеет SBOM этапа сборки либо анализа собранного артефакта, поскольку именно она фиксирует реальный набор версий, вошедших в поставку. В условиях эксплуатации особую роль приобретают формы, описывающие развернутую и реально используемую конфигурацию,

так как они позволяют уточнить применимость уязвимости в конкретной среде.

В этой связи VEX выступает не альтернативой SBOM, а ее естественным дополнением. Если SBOM отвечает на вопрос о составе, то VEX отвечает на вопрос о статусе уязвимости применительно к конкретному продукту [3–5]. Практическая ценность такого разделения состоит в том, что формально найденная уязвимость перестает автоматически рассматриваться как равнозначный риск: ее статус может быть уточнен с учетом путей выполнения, конфигурации, способа использования библиотеки и других факторов, которые недоступны при чисто компонентном сопоставлении.

Ограничения существующих подходов к генерации и обогащению SBOM

Практическая реализуемость методик, основанных на программной ведомости, определяется качеством инструментов ее генерации и последующего обогащения. В настоящее время существует зрелая экосистема решений, поддерживающих стандартизованные форматы SBOM и интеграцию в жизненный цикл разработки. Однако сравнительные исследования и практические обзоры показывают, что результаты таких инструментов существенно различаются по полноте, корректности и воспроизводимости [9–13].

Ключевое ограничение генераторов SBOM состоит в том, что ни один из инструментов не обеспечивает универсально полное и безусловно точное описание состава для всех языков, пакетных менеджеров, контейнерных образов и сборочных сценариев. Ошибки и неполнота исходных метаданных, различия в алгоритмах анализа, неодинаковая глубина поддержки транзитивных зависимостей и неполная нормализация компонентных

идентификаторов приводят к тому, что одна и та же кодовая база может быть описана разными инструментами неэквивалентным образом. Из этого следует, что методика, опирающаяся на SBOM, должна быть устойчивой к частично неполному и частично некорректному входу.

Аналогичное ограничение проявляется и на этапе обогащения программной ведомости данными об уязвимостях. Современные решения в основном строятся вокруг схемы «SBOM → нормализация идентификаторов → корреляция с банками данных уязвимостей → добавление сведений о CVE, описаний и оценок серьезности» [6–8, 12, 13]. Такой подход технологически оправдан и необходим, однако он сохраняет фундаментальную зависимость от качества исходной ведомости и от корректности сопоставления идентификаторов. Более того, даже корректно найденное соответствие между компонентом и уязвимостью не отвечает на вопрос о том, используется ли уязвимый функционал в исследуемом приложении и существует ли в данной архитектуре реалистичный сценарий эксплуатации.

Особую сложность представляет анализ в российских условиях, где требуется учитывать не только международные источники вроде OSV и GitHub Advisory Database, но и национальные банки данных уязвимостей, прежде всего БДУ ФСТЭК [6–8]. Расхождения в описаниях, диапазонах затронутых версий и правилах идентификации уязвимостей между различными источниками дополнительно усложняют процедуру корреляции и повышают требования к нормализации данных.

Таким образом, существующие инструментальные подходы позволяют автоматизировать сбор состава и первичное обогащение SBOM, но в ограниченной степени решают задачу контекстной интерпретации уязвимости. Именно этот предел и определяет необходимость перехода к методике, в

которой компонентный анализ дополняется анализом фактического использования зависимостей в кодовой базе.

Методика контекстного обогащения программной ведомости

Предлагаемая методика ориентирована на переход от статического перечня компонентов и потенциально связанных с ними уязвимостей к формированию обоснованных VEX-заявлений, учитывающих реальный контекст использования зависимостей. В логическом отношении она представляет собой воспроизводимую последовательность из шести этапов.

На первом этапе выполняется генерация программной ведомости исследуемого программного продукта в стандартизированном машиночитаемом формате. В качестве целевого формата используется CycloneDX, поскольку он обеспечивает достаточную выразительность для описания состава и удобен для последующего включения сведений об уязвимостях и VEX-совместимых данных [2]. На выходе формируется SBOM, содержащая перечень прямых и транзитивных компонентов, версии, поставщиков и, при наличии, связи зависимостей.

На втором этапе выполняется обогащение сформированной ведомости данными из внешних источников сведений об уязвимостях. Для каждого компонента из SBOM производится сопоставление с записями в выбранных базах данных, после чего формируется расширенный документ, в котором фиксируются идентификаторы уязвимостей, краткие текстовые описания и дополнительные атрибуты. При этом на данном этапе формируется лишь «сырой» перечень потенциально релевантных уязвимостей, который еще не учитывает фактический контекст использования библиотек в кодовой базе.

На третьем этапе кодовая база исследуемого приложения преобразуется в семантическое представление. Исходный код разбивается на фрагменты, для

каждого фрагмента формируется краткое текстовое описание его назначения, роли в обработке данных и потенциально значимых с точки зрения безопасности свойств. Затем эти описания преобразуются в числовые векторы и сохраняются вместе с кодом, путем к файлу и диапазонами строк в векторном хранилище. Такая схема позволяет использовать векторную базу как семантический индекс репозитория, где поиск выполняется не по буквальному совпадению строк, а по смысловой близости текстового описания уязвимости и фрагмента кода [14–16].

На четвертом этапе для каждой пары «компонент — уязвимость» формируется текстовый запрос к векторной базе данных. Этот запрос строится из названия компонента, версии, идентификатора уязвимости и ее краткого описания. После преобразования запроса в вектор выполняется семантический поиск, в результате которого извлекается ограниченный набор наиболее релевантных записей, включающих текстовые описания и соответствующие фрагменты исходного кода. Тем самым формируется компактный и содержательный контекст, пригодный для дальнейшего анализа генеративной моделью.

Пятый этап реализует собственно контекстный анализ эксплуатабельности уязвимости. Генеративная модель получает сведения о компоненте, идентификатор и описание уязвимости, а также извлеченные на предыдущем шаге фрагменты кода и их краткие описания. На этой основе формируется вывод о статусе уязвимости применительно к рассматриваемому продукту и текстовое обоснование, связывающее вывод с конкретными участками реализации, способами использования библиотеки и возможными условиями активации уязвимости [14–16]. Важно подчеркнуть, что вывод модели рассматривается не как окончательное доказательство, а как

обоснованное предположение, подлежащее дальнейшей верификации в процессах безопасной разработки.

На заключительном этапе результаты анализа фиксируются в форме VEX-совместимого заявления, содержащего идентификатор уязвимости, сведения о компоненте и версии, статус применимости и краткое текстовое обоснование. В результате программная ведомость перестает быть только перечнем состава и превращается в основу более зрелого цикла управления уязвимостями, где статусы уязвимостей привязаны не только к версии компонента, но и к контексту его фактического использования.

Экспериментальная апробация

Для проверки практической реализуемости предложенной методики был развернут экспериментальный стенд, включающий векторное хранилище Qdrant, платформу Dependency-Track для обогащения SBOM данными об уязвимостях, локальный сервис вычисления эмбеддингов и промежуточный сервис для работы с генеративной моделью. Вычисление, хранение и обработка векторов, равно как и компонентный анализ, размещались на отдельном локальном сервере под управлением Linux с 16 ГБ оперативной памяти и графическим адаптером NVIDIA GeForce GTX 1080. Такая архитектура позволила изолировать контур анализа и одновременно сохранить воспроизводимость эксперимента.

В качестве инструментальной основы прототипа использовались Qdrant для хранения векторного представления кодовой базы, генераторы SBOM в формате CycloneDX, Dependency-Track для корреляции компонентов с уязвимостями, модель эмбеддингов класса BGE для векторизации текстовых описаний и генеративная модель для формирования описаний кода, анализа уязвимостей и подготовки текстовых обоснований для VEX-заявлений. Кроме

того, были реализованы специализированные модули автоматизации: модуль индексирования кодовой базы, модуль анализа уязвимостей на основе обогащенного SBOM и модуль API для вычисления эмбеддингов.

Для апробации методики были выбраны два открытых проекта с воспроизводимыми сценариями уязвимостей и достаточно развитой структурой зависимостей. Первый объект представлял собой проект с демонстрацией Log4Shell, что позволяло проверить корректность анализа известной уязвимости в небольшом приложении на Java-стеке и сопоставить выводы с конкретными участками кода. Вторым объектом выступал OWASP Juice Shop, выбранный как пример приложения с большим объемом исходного кода, широким набором зависимостей и множеством поверхностей атаки. Выбор именно этих проектов обусловлен их распространенностью в исследовательской практике и наличием известных уязвимых зависимостей, пригодных для проверки роли семантического поиска при анализе эксплуатабельности.

Апробация выполнялась в соответствии с ранее сформулированной шестишаговой схемой. Для проекта с демонстрацией Log4Shell программная ведомость формировалась на основе контейнерного образа и экспортировалась в формате CycloneDX. Для OWASP Juice Shop ведомость строилась на основе файла package-lock.json, что позволяло получить перечень прямых и транзитивных зависимостей Node.js-проекта в машиночитаемом виде. После этого полученные SBOM-файлы загружались в Dependency-Track, где выполнялось их автоматическое обогащение сведениями об известных уязвимостях.

На следующем этапе кодовые базы проектов индексировались во векторном хранилище. Исходный код разбивался на семантически приемлемые фрагменты, для каждого из которых с помощью генеративной

модели формировалось краткое описание назначения, особенностей обработки данных и потенциально значимых с точки зрения безопасности операций. Затем эти описания преобразовывались в эмбеддинги, а результирующие записи сохранялись в Qdrant вместе с исходным кодом и метаданными, включая путь к файлу и диапазоны строк. Такая организация данных обеспечивала последующий переход от текстового описания уязвимости к наиболее релевантным участкам реализации.

Далее для каждой пары «компонент — уязвимость», извлеченной из обогащенной программной ведомости, формировался запрос к векторной базе. Запрос строился из названия компонента, версии, идентификатора уязвимости и ее краткого описания. По полученному вектору извлекались наиболее близкие записи из индекса кодовой базы, после чего соответствующие описания и фрагменты исходного кода передавались генеративной модели. На основании этого контекста формировалось VEX-заявление, включающее статус применимости уязвимости, текстовое обоснование и рекомендации по устранению.

По результатам анализа была получена показательная картина. В проекте с демонстрацией Log4Shell уязвимый компонент был корректно выявлен, а сформированное VEX-заявление содержало привязку к релевантным участкам кода и рекомендации по устранению, что позволило однозначно соотнести уязвимость с конкретной реализацией. В проекте OWASP Juice Shop были заявлены две уязвимые зависимости: sanitize-html версии 1.4.2 и express-jwt версии 0.1.3. По итогам анализа уязвимость в sanitize-html была определена как эксплуатируемая, причем в результирующем VEX-заявлении были приведены меры по ее митигации и устранению. Вместе с тем уязвимость, связанная с использованием express-jwt, не была подтверждена как эксплуатируемая. Наиболее вероятное объяснение этого результата состоит в недостаточном

объеме доступного контекста и в сложности восстановления условий эксплуатации для транзитивной зависимости, когда уязвимым является не сам основной пакет, а его вложенный компонент.

Тем самым эксперимент показал, что предложенная методика работоспособна для сценариев, в которых требуется анализ уязвимостей прямых зависимостей и их связь с конкретными участками исходного кода. Из трех изначально обозначенных уязвимых компонентов два были обнаружены и корректно описаны, причем для них были сформированы связанные с кодовой базой обоснования и рекомендации по устранению. Одновременно эксперимент выявил содержательное ограничение метода: при анализе транзитивных зависимостей требуется более глубокое восстановление контекста использования уязвимого функционала, чем то, которое обеспечивается текущим сочетанием компонентного анализа, семантического поиска и генеративной интерпретации.

Обсуждение результатов

Полученные результаты позволяют сделать несколько принципиальных выводов.

Во-первых, сама идея контекстного обогащения программной ведомости оказывается практически реализуемой в виде воспроизводимого конвейера, объединяющего генерацию SBOM, ее обогащение уязвимостями, индексирование кодовой базы и последующее формирование VEX-заявлений. Это означает, что SBOM может использоваться не только как артефакт инвентаризации, но и как входной слой более сложной аналитической процедуры.

Во-вторых, подход показывает наибольшую эффективность в отношении прямых зависимостей, для которых связь между компонентом, описанием уязвимости и конкретными участками исходного кода

восстанавливается относительно прозрачно. В таких случаях генеративная модель, получая релевантный контекст из векторной базы, способна сформировать содержательное и инженерно полезное обоснование статуса уязвимости.

В-третьих, эксперимент выявил важное ограничение, связанное с анализом транзитивных зависимостей. В подобных сценариях одного семантического поиска по текстовому описанию уязвимости оказывается недостаточно: требуется более глубокая реконструкция пути достижения уязвимого функционала, учет вложенных цепочек вызовов и, вероятно, дополнительное использование методов статического или гибридного анализа. Следовательно, текущая методика должна рассматриваться не как универсальная замена экспертного анализа, а как средство его предварительного структурирования и ускорения.

Наконец, существенное значение имеет то, что вывод генеративной модели в явном виде интерпретируется как предположение, требующее верификации в рамках процессов безопасной разработки, особенно в критически значимых системах. Такая постановка сохраняет инженерную добросовестность подхода и не подменяет формальную экспертизу текстом модели, а использует ее как механизм повышения информативности и снижения нагрузки на аналитика.

Заключение

В статье показано, что традиционное использование программной ведомости в задачах управления уязвимостями сталкивается с принципиальным ограничением: формальная корреляция компонентов с внешними банками данных уязвимостей не позволяет надежно определить

фактическую применимость уязвимости к конкретному программному продукту. Основной дефицит таких подходов состоит в недостаточном учете архитектурного и кодового контекста, без которого многие найденные CVE остаются лишь потенциальными совпадениями и требуют дорогостоящей ручной интерпретации.

Для преодоления данного ограничения предложена методика контекстного обогащения SBOM, объединяющая средства генерации ведомости, платформу компонентного анализа, векторное представление кодовой базы и генеративную модель, используемую для формирования VEX-совместимых утверждений. В отличие от традиционного компонентного анализа, предложенный подход ориентирован не только на фиксацию состава и известных уязвимостей, но и на выявление признаков их фактической применимости в коде исследуемого приложения.

Апробация на открытых проектах с воспроизводимыми сценариями уязвимостей показала, что методика работоспособна и позволяет связывать уязвимости прямых зависимостей с конкретными фрагментами реализации, формируя содержательные VEX-заявления с текстовым обоснованием и рекомендациями по устранению. Одновременно выявлено ограничение, связанное с анализом транзитивных зависимостей, для которых требуется более глубокое восстановление контекста использования уязвимого функционала.

Теоретическое значение работы состоит в уточнении места программной ведомости в архитектуре анализа уязвимостей: SBOM рассматривается не как конечный результат инвентаризации, а как исходный слой для последующего контекстного анализа. Практическое значение заключается в возможности снижения доли ложноположительных срабатываний и повышения полезности

программной ведомости в процессах безопасной разработки, сопровождения и управления рисками программной цепочки поставок.

Перспективы дальнейшей работы связаны с расширением набора тестовых проектов, введением количественных метрик качества формируемых VEX-заявлений, а также с углублением анализа транзитивных зависимостей за счет включения дополнительных методов статического и потокового анализа кода.

Использованные источники:

1. Software Bill of Materials (SBOM) // Cybersecurity and Infrastructure Security Agency (CISA). [Электронный ресурс] URL: https://www.cisa.gov/sbom (дата обращения: 12.11.2025).

2. ECMA-424: Software Bill of Materials (SBOM) // ECMA International. [Электронный ресурс] URL: https://ecma-international.org/publications-and-standards/standards/ecma-424/ (дата обращения: 06.11.2025).

3. SPDX Specification v3.0.1 (PDF) // SPDX. [Электронный ресурс] URL: https://spdx.dev/wp-content/uploads/sites/31/2024/12/SPDX-3.0.1-1.pdf (дата обращения: 13.11.2025).

4. Deciphering VEX and SPDX: a deep dive into software vulnerability analysis and reporting // SPDX. [Электронный ресурс] URL: https://spdx.dev/deciphering-VEX%20and-spdx-a-deep-dive-into-software-vulnerability-analysis-and-reporting/ (дата обращения: 07.12.2025).

5. CSAF (Common Security Advisory Framework) // csaf.io. [Электронный ресурс] URL: https://www.csaf.io/ (дата обращения: 10.12.2025).

6. OSV (Open Source Vulnerabilities) // osv.dev. [Электронный ресурс] URL: https://osv.dev/ (дата обращения: 13.11.2025).

7. GitHub Advisory Database (описание) // GitHub Docs. [Электронный ресурс] URL: https://docs.github.com/ru/code-security/security-advisories/working-with-global-security-advisories-from-the-github-advisory-database/about-the-github-advisory-database (дата обращения: 14.11.2025).

8. Регламент включения информации об уязвимостях программного обеспечения и программно-аппаратных средств в банк данных угроз безопасности информации ФСТЭК России // docs.cntd.ru. [Электронный ресурс] URL: https://docs.cntd.ru/document/564200972 (дата обращения: 22.11.2025).

9. Syft (anchore/syft) // GitHub. [Электронный ресурс] URL: https://github.com/anchore/syft (дата обращения: 26.11.2025).

10. ScanCode Toolkit // AboutCode. [Электронный ресурс] URL: https://aboutcode.org/scancode/ (дата обращения: 28.11.2025).

11. cdxgen // GitHub. [Электронный ресурс] URL: https://github.com/cdxgen/cdxgen (дата обращения: 10.11.2025).

12. Dependency-Track // Dependency-Track. [Электронный ресурс] URL: https://dependencytrack.org/ (дата обращения: 02.11.2025).

13. GUAC // OpenSSF. [Электронный ресурс] URL: https://openssf.org/projects/guac/ (дата обращения: 10.11.2025).

14. Radanliev P., De Roure D., Santos O. Generative Pre-Trained Transformers, Natural Language Processing and Artificial Intelligence and Machine Learning (AI/ML) in Software Vulnerability Management: automations in the Software Bill of Materials (SBOM) and the Vulnerability-Exploitability eXchange (VEX) // Preprints.org, 2023. [Электронный ресурс] URL: https://www.preprints.org/manuscript/202307.1303 (дата обращения: 13.11.2025).

15. DepsRAG: Towards Managing Software Dependencies using Large Language Models // arXiv, 2024. [Электронный ресурс] URL: https://arxiv.org/pdf/2405.20455v3 (дата обращения: 03.11.2025).

16. Leveraging MDS2 and SBOM data for LLM-assisted software vulnerabilities analysis and classification // PubMed Central (PMC). [Электронный ресурс] URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC12309023/ (дата обращения: 19.11.2025).