DOI 10.36622/1682-7813.2024.27.1.002
УДК 004.056

АВТОМАТИЗИРОВАННЫЙ БАНК ЗНАНИЙ И КАЛЬКУЛЯТОР РИСКОВ РЕАЛИЗАЦИИ КИБЕРАТАК И УЯЗВИМОСТЕЙ (ЧАСТЬ II)

Г.А. Остапенко, А.П. Васильченко, А.А. Остапенко, Д.С. Нестеров, А.С. Дубов, В.А. Старцев

В порядке развития инструментария риск-анализа в работе предложены оригинальные алгоритмы оценки рисков, учитывающие нарушение качеств информации и работоспособности защищенных систем и сетей под воздействием разнообразных векторов атак и используемых ими уязвимостей. Осуществлена алгоритмизация разработанных методик в виде калькуляторов риска частичной, либо полной утраты ценности, конфиденциальности и доступности информации с учетом специфики защищаемого объекта и множества используемых уязвимостей. Предложена серия алгоритмов формирования банка знаний кибератак и уязвимостей в виде агрегированных регламентов различных стадий противодействия вторжениям, включая текущее реагирование и ликвидацию последствий в отношении зарегистрированных инцидентов. Алгоритмизация позволяет пользователю в диалоговом режиме получить из банка практические рекомендации по борьбе с многообразием сценариев атак и брешей, используемых злоумышленниками.

Ключевые слова: система, безопасность, риск, ущерб, вероятность, регламент, база знаний и данных, алгоритм.

Введение

Для того, чтобы осуществлять алгоритмизацию для программного инструментария, важно определиться с основными целями, задачами и требованиями. Четкое определение требований, целей и задач помогает разработчикам найти более эффективные и надежные решения, что также помогает в планировании и организации работы, а также в управлении ресурсами и рисками.

Функциональные требования определяют, что именно должен делать инструментарий, в то время как нефункциональные требования определяют, как он должен работать. При этом, цели и задачи определяют, чего именно нужно достичь и какие шаги нужно предпринять для достижения этих целей. Они служат направляющим принципом для всего процесса разработки.

Перед тем, как реализовывать алгоритмическое обеспечение инструментария, необходимо определиться с основными информационными потоками. Это связано с тем, что алгоритмы и программное обеспечение в значительной степени зависят от структуры и потоков данных, которые обрабатывает инструментарий.

Определение основных информационных потоков устанавливает, какие данные будут использоваться, как они будут передаваться и как они будут обрабатываться. Это также помогает в проектной деятельности, включая выбор подходящих технологий и методов обработки данных.

Первостепенной задачей в любом проекте обеспечения кибербезопасности является агрегация данных о многообразии векторов атак и уязвимостей, что важно для создания эффективной системы защиты, которая может адаптироваться к постоянно меняющимся угрозам [1-16].

Агрегация данных – это процесс сбора, организации и анализа большого объема информации. В контексте кибербезопасности, это может включать в себя сбор данных о различных типах атак, их источниках, целях и последствиях. Также важно собирать данные о уязвимостях в системах и программном обеспечении, которые могут быть использованы злоумышленниками [1-7].

Эта задача требует глубоких знаний в области кибербезопасности, а также навыков работы с большими данными. Результатом

этого будет ценная информация, которую можно использовать для разработки стратегий и политик безопасности, а также для обучения персонала [1].

Для выполнения задачи агрегации данных о многообразии векторов атак и уязвимостей необходимо соблюдать следующие основные требования:

- получение данных. Сначала необходимо получить данные о всех известных векторах атак (CAPEC) и уязвимостях (CVE). Это может включать в себя сбор данных из открытых источников, таких как базы данных NIST National Vulnerability Database (NVD), или через API, предоставляемые этими базами данных [1-6],
- редактирование и обновление данных. Полученные данные должны быть тщательно проверены и отредактированы для удаления лишних или некорректных записей. Кроме того, данные должны быть регулярно обновлены, чтобы отражать новые угрозы и уязвимости [7],
- организация данных. Данные должны быть организованы таким образом, чтобы было легко их анализировать и использовать. Это может включать в себя категоризацию данных по типу атаки или уязвимости, а также создание связей между различными записями,
- анализ данных. После того как данные были собраны, отредактированы и организованы, они должны быть проанализированы. Это может включать в себя идентификацию общих тенденций или паттернов, а также определение областей, которые требуют дополнительного внимания,
- использование данных. Наконец, полученные данные должны быть использованы для создания эффективных стратегий и политик безопасности, а также для обучения персонала. Это может включать в себя создание отчетов, обучающих материалов или инструментов для мониторинга и управления угрозами [8-16].

Вторая задача реализуемого проекта – построение риск-ландшафта для пар вектор атаки – уязвимость, который поможет пользователю понять, какие угрозы существуют и как могут быть минимизированы риски их реализации [8-14].

Риск-ландшафт представляет собой визуальное представление рисков реализации всех возможных атак и уязвимостей заданного вида. Он показывает, как различные векторы атак могут быть использованы для эксплуатации уязвимостей, и помогает нам определить, какие меры противодействия нужно принять, чтобы минимизировать риск [8-14].

Построение риск-ландшафта включает в себя несколько ключевых этапов. Во-первых, мы должны определить все возможные векторы атак и уязвимости, которые могут существовать в нашей системе. Затем мы должны оценить риск, связанный с каждой парой вектор атаки – уязвимость, и представить эти данные в виде графика или диаграммы.

Для успешного выполнения задачи построения риск-ландшафта для пар вектор атаки – уязвимость необходимо соблюдать следующие требования:

1) определение векторов атак и уязвимостей: Первый шаг – это точное определение множества всех возможных векторов атак и уязвимостей, которые могут существовать в нашей системе,

2) оценка риска: Каждая пара вектор атак - уязвимость должна быть оценена по уровню риска, включая в себя учет таких факторов, как вероятность атаки и потенциальный ущерб от атаки,

3) визуализация риск-ландшафта: после того как все риски были оценены, они должны быть представлены в виде риск-ландшафта, что может быть сделано с помощью различных видов графиков или диаграмм, которые позволят пользователю понять структуру рисков и определить области ландшафта, требующие особого внимания,

4) анализ и интерпретация риск-ландшафта: Риск-ландшафт должен быть тщательно проанализирован и интерпретирован, что может включать в себя идентификацию наиболее значительных рисков и выработку мер для их минимизации, включая реагирование на возможные атаки,

5) поддержка и обновление риск-ландшафта: Риск-ландшафт должен быть поддерживаемым и регулярно обновляемым,

чтобы отражать изменения в сфере обеспечения кибербезопасности.

Архитектура программного инструментария

Планирование архитектуры инструментария перед реализацией — это ключевой шаг в процессе разработки, который поможет создать программный инструментарий, который будет удовлетворять всем требованиям и поставленным задачам.

В целях решения поставленных задач и разработки алгоритмов для реализации настоящего проекта, была предложена следующая архитектура:

1) база данных. База данных хранит все необходимые данные, включая информацию о векторах атак, уязвимостях и рисках. Она обеспечивает быстрый и эффективный доступ к данным, используемым для анализа и визуализации риск-ландшафта,

2) скрипты для наполнения базы данных. Эти скрипты используются для автоматического заполнения базы данных начальными данными о векторах атак и уязвимостях. Они должны быть способны обрабатывать большие объемы данных,

3) сервис агрегат. Этот компонент служит центральным узлом продукта, так как он обрабатывает запросы от других сервисов, объединяет данные из разных источников и предоставляет единую точку доступа к информации,

4) сервис обновления данных. Этот сервис отвечает за обновление базы данных. Он может автоматически собирать новую информацию о векторах атак и уязвимостях, а также обновлять оценки риска на основе актуальных данных открытого доступа,

5) фронт-сервис. Данный сервис предоставляет пользовательский интерфейс для взаимодействия с системой. Он позволяет пользователям просматривать риск-ландшафт, анализировать риски и принимать решения о мерах реагирования на киберугрозы,

Предложенная архитектура обеспечивает гибкость и масштабируемость, позволяя легко расширять и модифицировать систему в соответствии с меняющимися требованиями и условиями (рис. 1).

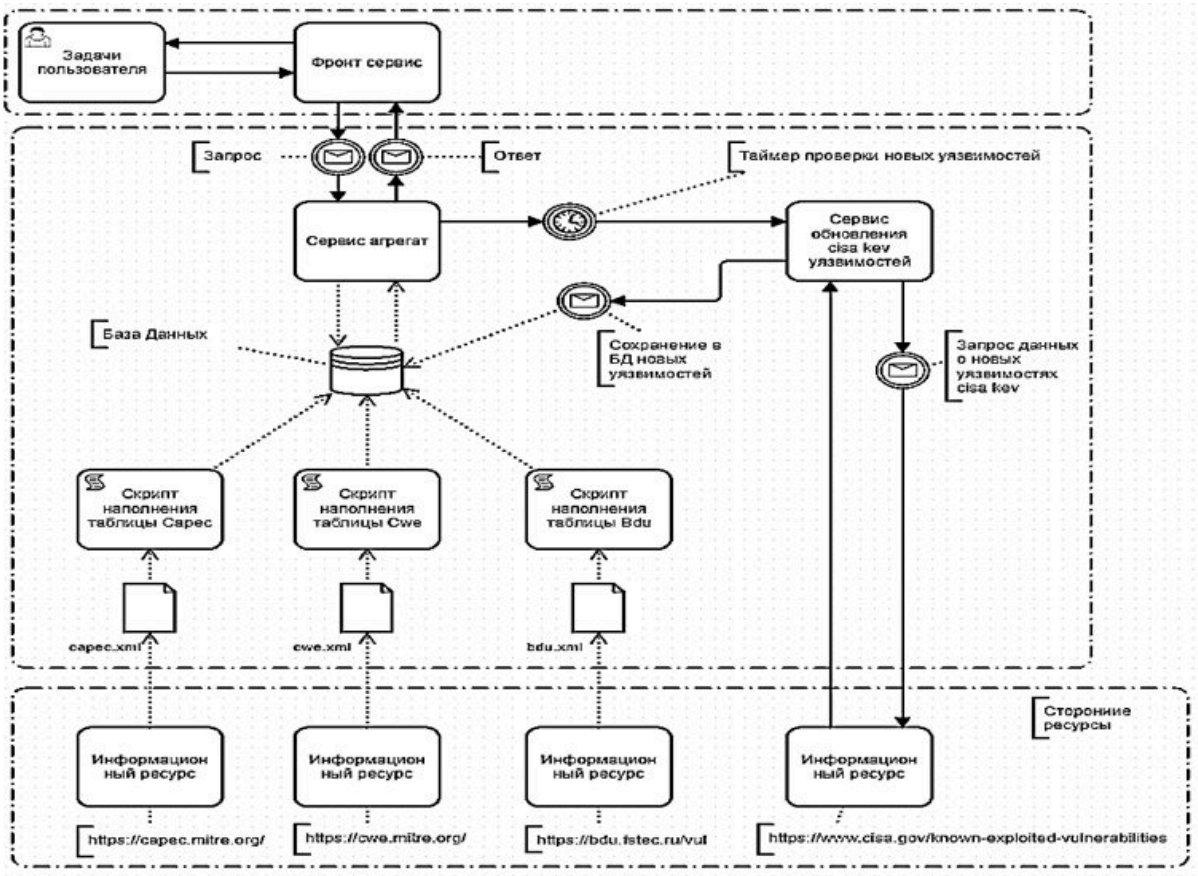

Рис. 1. Архитектура программного инструментария

Так как база данных выступает в качестве хранилища данных, формирование **алгоритмического обеспечения** для нее не уместно. Однако, взаимодействие с этим элементом фигурирует в алгоритмах остальных элементов архитектуры программного обеспечения. Подробное описание ее структуры будет изложено в программном обеспечении инструментария автоматизации.

Реализованный фронтенд-сервис, отображающий данные о многообразии уязвимостей и векторов атак, играет ключевую роль в обеспечении безопасности информационных систем.

Этот сервис служит мостом между пользователями сервисом агрегации данных, содержащей информацию о различных типах уязвимостей и векторах атак.

Основное назначение этого сервиса – предоставление пользователям доступа к актуальной и точной информации о потенциальных угрозах безопасности. Это позволяет пользователям принимать обоснованные решения о том, какие меры безопасности следует принять для защиты своих систем от возможных атак.

Основные алгоритмы фронт сервиса состоят в последовательном вызове предоставленных сервисом агрегатом данных в формате JSON.

1 Алгоритмизация сервиса агрегата данных о многообразии векторов атак и уязвимостей

Сервис-агрегат представляет собой структуру, которая обеспечивает различные функциональные возможности для работы с данными, включая следующие основные операции:

- CRUD операции. Позволяют создавать, читать, обновлять и удалять данные. Эти операции являются основой любого сервиса агрегата,
- получение связанных сущностей. Этот метод используется для извлечения связанных данных. Например, он должен по id CAPEC устанавливать соответствие с уязвимостями CVE и наоборот,
- поиск по ключевым словам. Этот метод позволяет искать данные по определенным ключевым словам, что необходимо реализовать для таких сущностей CVE и CAPEC,
- функция таймера на обновление списка уязвимостей. Автоматически обновляет список уязвимостей через определенный интервал времени. Это может быть полезно для поддержания актуальности информации о кибератаках и уязвимостях.

Все эти операции выполняются над данными, хранящимися в базе данных. Сервис-агрегат обеспечивает доступ к этим данным, обрабатывая запросы от клиентов и возвращая результаты в формате, который можно легко использовать (рис. 2).

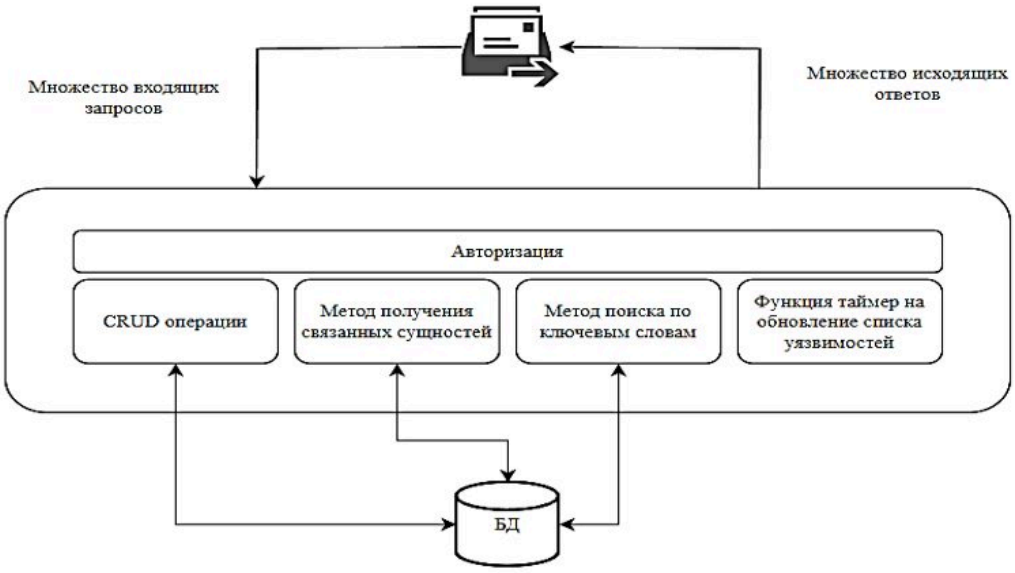

Рис. 2. Алгоритмическая структура сервиса агрегата.

1.1 Унифицированные алгоритмы CRUD операций сервиса, агрегирующего данные о многообразии векторов атак и уязвимостей

Алгоритм получения данных по id

Данный алгоритм необходим для получения более детальной информации о целевом объекте. Такой алгоритм должен быть реализован для всех сущностей, которые будут храниться в базе данных, а именно информация о (рис. 3):

- векторе атаки CAPEC;
- уязвимости CVE;
- о уязвимостях зарегистрированных в БДУ ФСТЭК;
- слабостях и «дырах» в системах CWE.

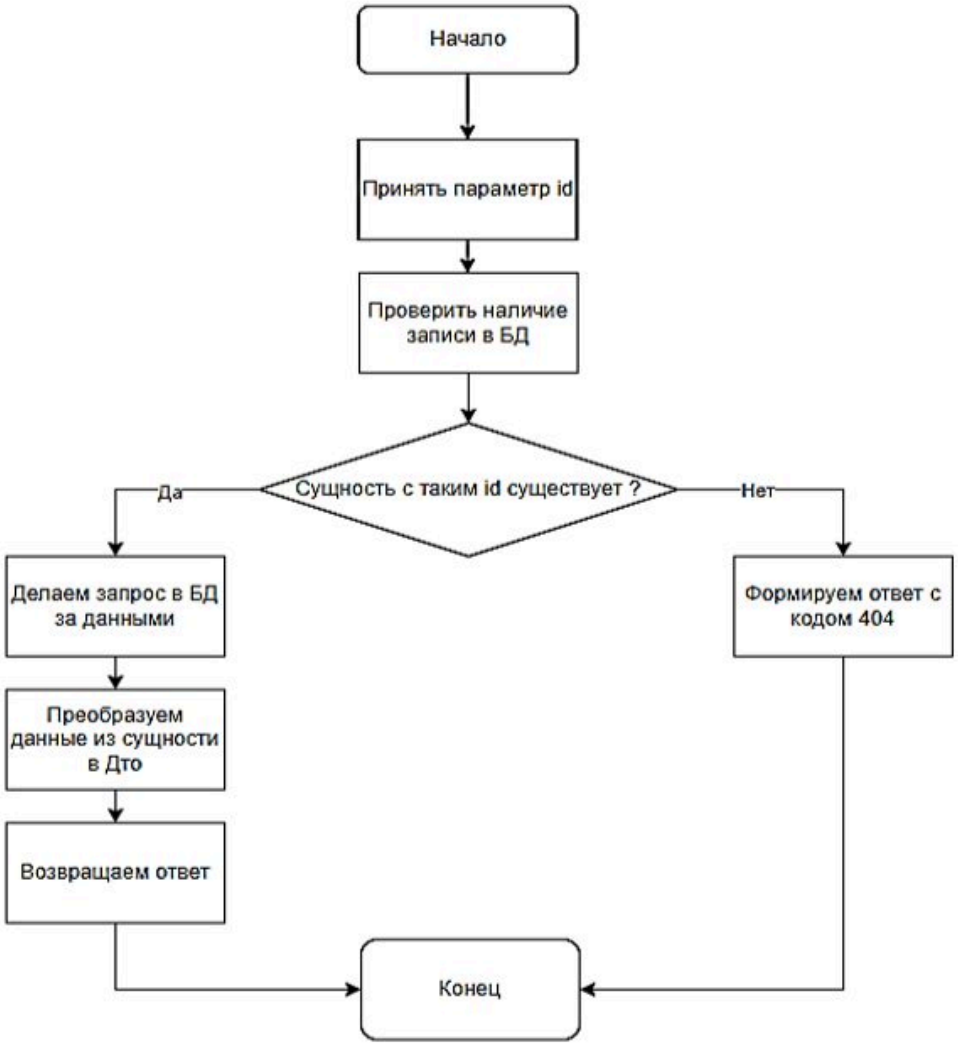

Рис. 3. Алгоритм получения данных по id

Для рис. 3 ниже предлагается подробное описание алгоритма получения данных по id:

1) принимаем параметр `id`. Это может быть любой уникальный идентификатор, который используется для поиска конкретной записи в базе данных,

2) проверяем наличие записи в базе данных с данным `id`. Это можно сделать, выполнив запрос к базе данных с использованием предоставленного `id`,

3) если сущность с таким `id` не существует, формируем ответ с кодом 404. Это стандартный HTTP-статус, который означает "Не найдено". В теле ответа можно включить сообщение об ошибке, которое поможет пользователю понять, что произошло,

4) если сущность с таким `id` существует, делаем запрос в базе данных за данными этой сущности. Это может включать в себя получение всех связанных данных или только определенных полей,

5) преобразуем данные из сущности в Data Transfer Object (DTO). DTO – это объект, который используется для инкапсуляции данных и передачи их между процессами или потоками. Он обычно содержит только те поля, которые необходимы клиенту,

6) возвращаем ответ. Ответ должен содержать статус 200 (OK), а также данные в формате JSON.

Алгоритм получения списка данных

Для более полного и качественного решения поставленных задач был разработан алгоритм получение списка данных для базы данных (БД), основываясь на технологии получения пагинирования списка данных. Получение пагинированного списка из БД — это технология, которая позволяет извлекать данные из БД по частям, вместо того чтобы извлекать все данные сразу.

Это особенно полезно, когда работа осуществляется с большими объемами данных. Вместо того чтобы загружать все данные за один раз, что может занять много времени и ресурсов, можно загрузить их по частям. Это также улучшает производительность, поскольку пользователь может начать просматривать данные, пока остальные части данных загружаются.

Рис. 4 иллюстрирует предлагаемый алгоритм, описание которого приводится ниже:

1) принимаем параметры для получения списка: `offset` и `pageSize`. Эти параметры используются для определения начального места в списке и количества элементов на странице соответственно,

2) делаем запрос в базе данных. Запрос может быть различным в зависимости от того, какие данные мы хотим получить. Например, мы можем захотеть получить все записи, или только те, которые удовлетворяют определенному условию,

Рис. 4. Алгоритм получения списка данных

3) формируем список данных. Это может включать в себя преобразование результатов запроса в нужный формат, добавление дополнительной информации и т.д.,

4) формируем данные для пагинации. Это может включать в себя подсчет общего количества страниц, определение текущей страницы и т.д.,

5) возвращаем ответ. Ответ должен содержать статус 200 (OK), список данных и данные для пагинации. Все эти данные обычно возвращаются в формате JSON или XML.

Алгоритм добавления сущности

В случае обновления данных в сторонних источниках появляется необходимость добавить недостающие данные в целевую БД. Для этого реализован алгоритм добавления сущности (рис. 5).

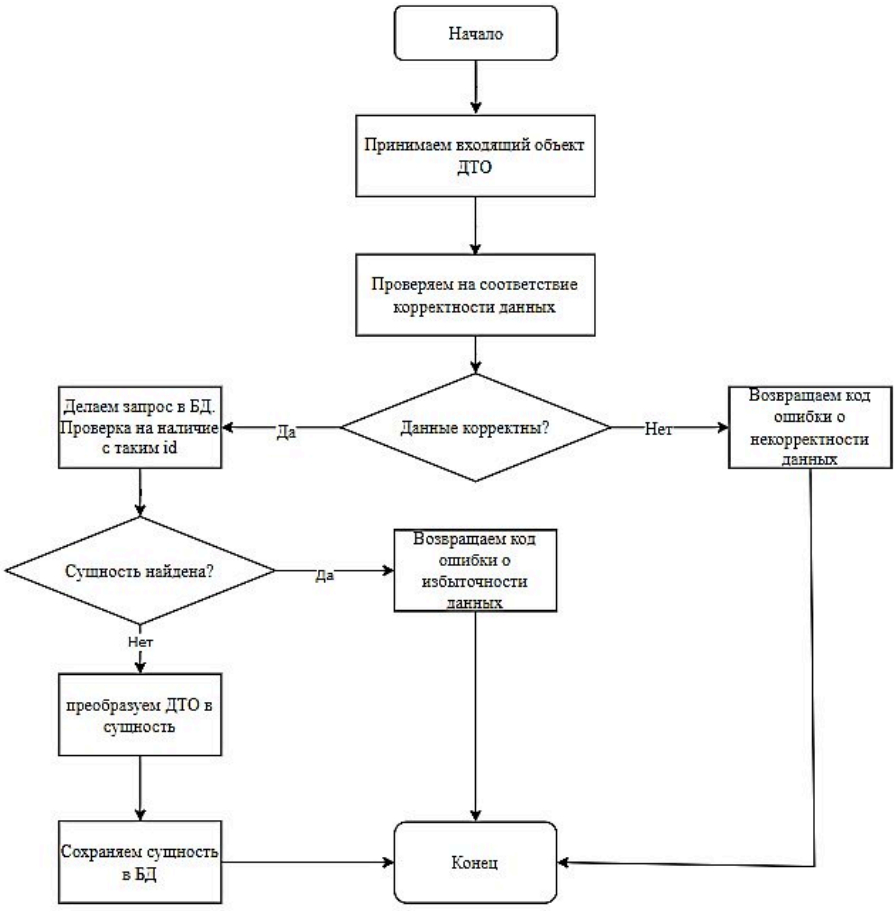

Рис. 5. Алгоритм добавление сущности в БД

Ниже приводится описание алгоритма добавления сущности в БД:

1) принимаем входящий объект DTO,

2) проверяем корректность данных в DTO. Это может включать в себя проверку на наличие обязательных полей, проверку формата данных, проверку диапазона значений и т.д.,

3) если данные некорректны, возвращаем код ошибки. Это может быть любой код ошибки, который соответствует типу ошибки,

4) если данные корректны, делаем запрос в базе данных на проверку наличия сущности с таким же `id`, как и в DTO,

5) если сущность с таким `id` уже существует, возвращаем код ошибки. Это может быть код 409 (Conflict), который указывает на то, что запрос не может быть выполнен из-за конфликта с существующими ресурсами,

6) если сущности с таким `id` не существует, преобразуем DTO в сущность. Это может включать в себя копирование данных из DTO в новый экземпляр сущности,

7) сохраняем новую сущность в базе данных. Это может включать в себя выполнение SQL-запроса INSERT или аналогичного,

8) возвращаем код статуса успешности. Это может быть код 201 (Created), который указывает на то, что запрос был успешно обработан и в результате была создана новая сущность.

Алгоритм обновления сущности

В реализованном программном инструментарии данные из большинства сторонних источников записаны на

английском языке. Для решения этой проблемы пользователю было предоставлена возможность редактировать данные из источников по алгоритму обновления сущности (рис. 6).

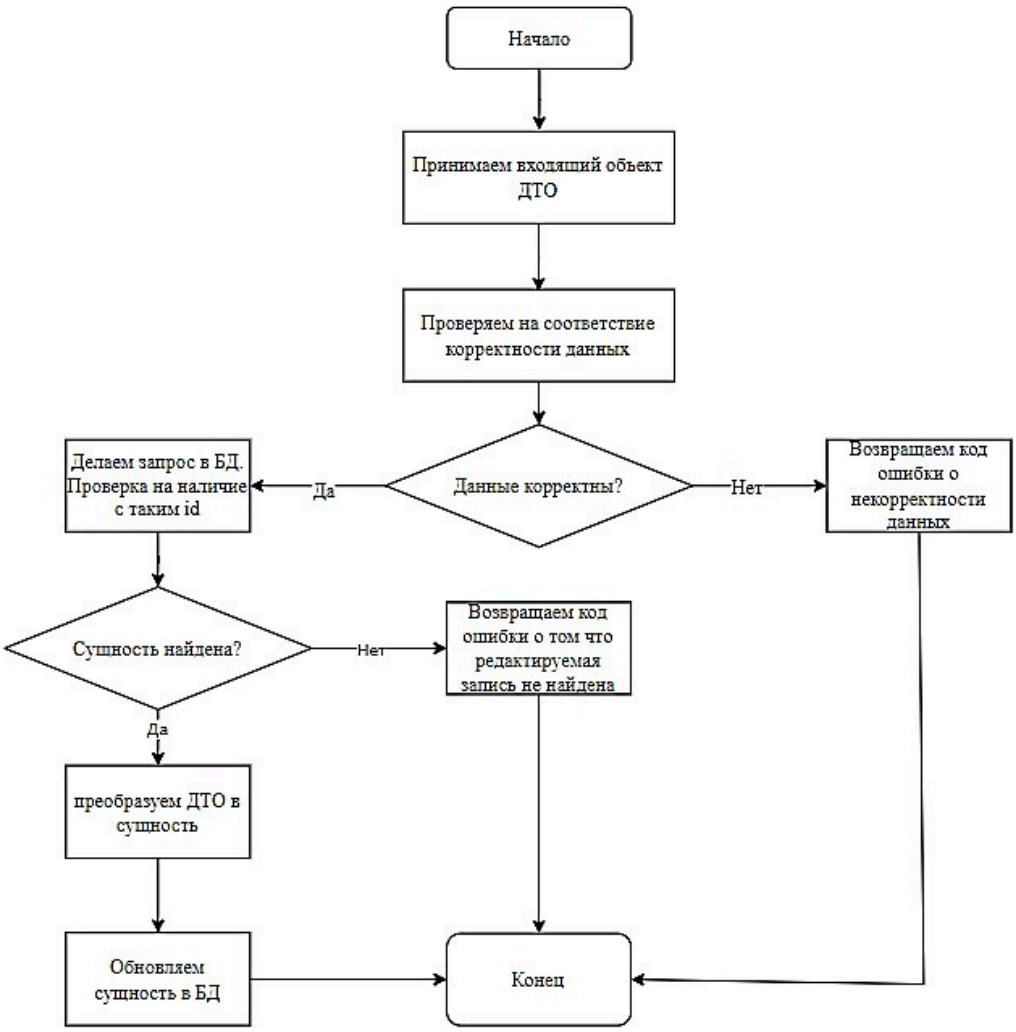

Рис. 6. Алгоритм обновления сущности

Ниже приводится описание алгоритма обновления сущности (рис. 6):

1) принимаем входящий объект DTO,

2) проверяем корректность данных в DTO. Это может включать в себя проверку на наличие обязательных полей, проверку формата данных, проверку диапазона значений и т.д.,

3) если данные некорректны, возвращаем код ошибки. Это может быть любой код ошибки, который соответствует типу ошибки,

4) если данные корректны, делаем запрос в базе данных на проверку наличия сущности с таким же `id`, как и в DTO,

5) если сущности с таким `id` не существует, возвращаем код ошибки, указывающий на то, что редактируемая сущность не найдена. Это может быть код 404 (Not Found),

6) если сущность с таким `id` существует, преобразуем DTO в сущность. Это может включать в себя копирование данных из DTO в существующий экземпляр сущности,

7) обновляем сущность в базе данных. Это может включать в себя выполнение SQL-запроса UPDATE или аналогичного,

8) возвращаем код статуса успешности. Это может быть код 200 (OK), который указывает на то, что запрос был успешно обработан и в результате была обновлена сущность.

Алгоритм удаления сущности

В случае, если некоторые данные, которые были добавлены пользователем являются некорректными, или некоторые данные из источников являются неактуальными, возникает необходимость удаления избыточных данных в БД. Для этого был разработан алгоритм удаления сущности из БД (рис. 7).

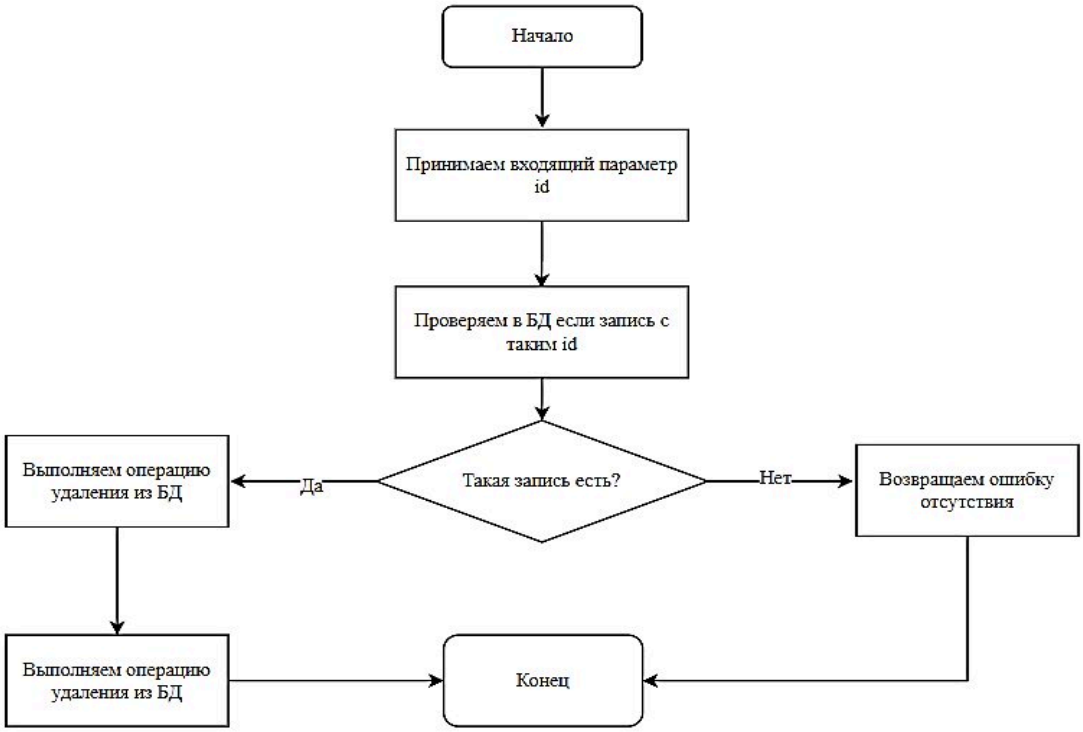

Рис. 7. Алгоритм удаления сущности из БД

Ниже приводится описание алгоритма удаления сущности из БД (рис. 7):

1) принимаем входящий параметр `id`. Это уникальный идентификатор сущности, которую мы хотим удалить,

2) проверяем в базе данных наличие записи с таким `id`. Это может включать в себя выполнение SQL-запроса SELECT или аналогичного,

3) если записи с таким `id` не существует, возвращаем ошибку, указывающую на то, что сущность не найдена. Это может быть любой код ошибки, который соответствует типу ошибки,

4) если запись с таким `id` существует, выполняем операцию удаления из базы данных. Это может включать в себя выполнение SQL-запроса DELETE или аналогичного,

5) возвращаем статус успешности. Это может быть код 200 (OK), который указывает на то, что запрос был успешно обработан и в результате была удалена сущность.

1.2 Алгоритмы агрегации связанных данных о многообразии векторов атак и уязвимостей

Алгоритм получения информации о связанных уязвимостях CVE по id вектора атаки CAPEC позволяет сопоставить, какие уязвимости CVE могут быть связаны с конкретным вектором атаки CAPEC.

Блок схема алгоритма представлена на рис. 8.

Алгоритм для получения уязвимостей CVE по id вектора атаки CAPEC выглядит следующим образом (рис. 8):

1) принимаем параметр `capecId`, который является уникальным идентификатором вектора атаки CAPEC,

2) проверяем наличие объекта с таким `capecId` в нашей базе данных. Если объект не найден, возвращаем код ошибки,

3) если объект найден, выполняем поиск по связанным слабостям CWE (Common Weakness Enumeration). Это может включать

в себя выполнение SQL-запроса или аналогичного, чтобы найти все слабости, связанные с данным вектором атаки,

4) затем для каждой найденной слабости CWE выполняем поиск по связанным уязвимостям CVE (Common Vulnerabilities and Exposures). Это может включать в себя выполнение другого SQL-запроса или аналогичного, чтобы найти все уязвимости, связанные с данной слабостью,

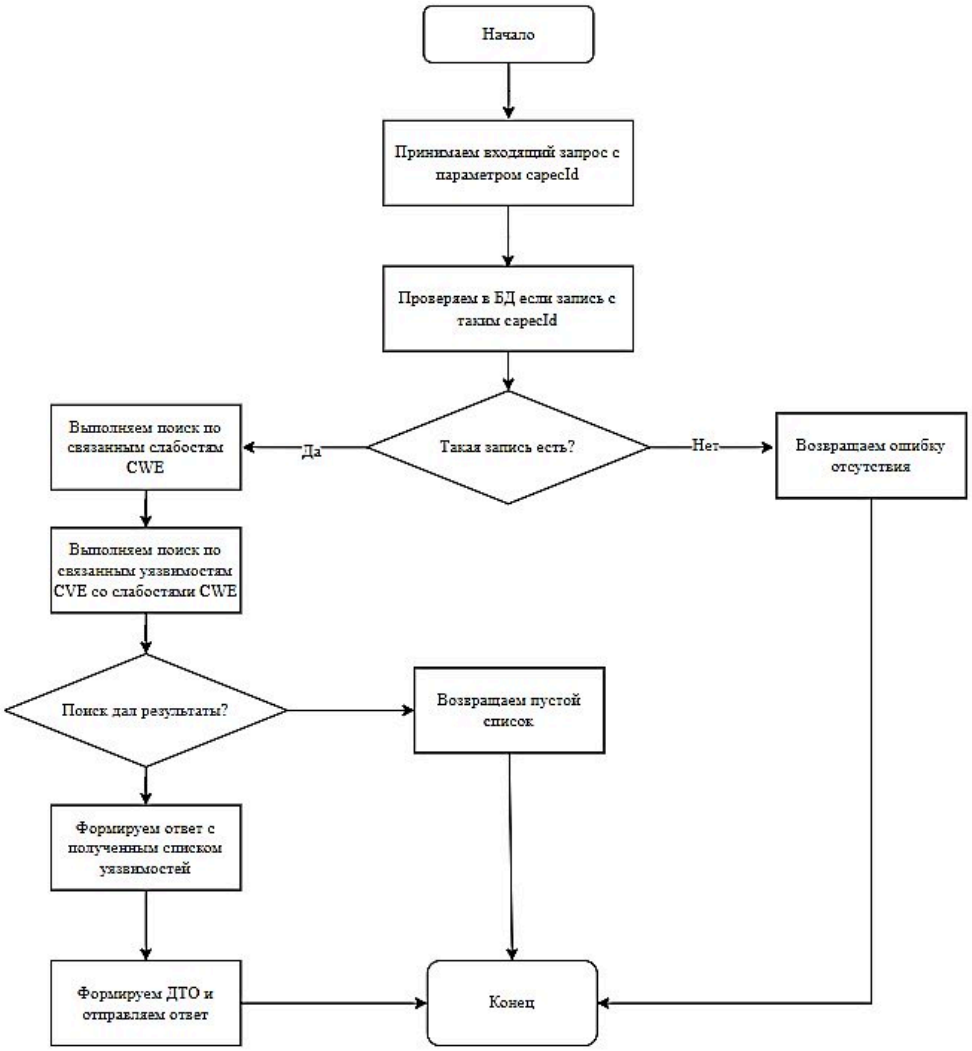

Рис. 8. Алгоритм получения информации о связанных уязвимостях CVE по id вектора атаки CAPEC

5) если поиск не успешен (то есть не были найдены связанные уязвимости CVE), возвращаем пустой список,

6) если поиск успешен, формируем ответ с полученным списком уязвимостей. Это может включать в себя создание нового объекта или структуры данных, которая содержит информацию о каждой уязвимости, включая ее идентификатор, название, описание, уровень риска и другие сведения,

7) наконец, формируем DTO и отправляем ответ. В данном случае, DTO может содержать список всех уязвимостей CVE, связанных с данным вектором атаки CAPEC.

Алгоритм для получения информации о связанных векторах атаки CAPEC по id уязвимости CVE

Алгоритм для получения информации о связанных векторах атаки CAPEC по id уязвимости CVE выглядят следующим образом (рис. 9):

1) принимаем параметр `cveId`, который является уникальным идентификатором уязвимости CVE,

2) проверяем наличие объекта с таким `cveId` в нашей базе данных. Если объект не найден, возвращаем код ошибки,

3) если объект найден, выполняем поиск по связанным слабостям CWE (Common Weakness Enumeration). Это может включать в себя выполнение SQL-запроса или аналогичного, чтобы найти все слабости, связанные с данной уязвимостью,

4) затем для каждой найденной слабости CWE выполняем поиск по связанным векторам атаки CAPEC. Это может включать в себя выполнение другого SQL-запроса или аналогичного, чтобы найти все векторы атак, связанные с данной слабостью,

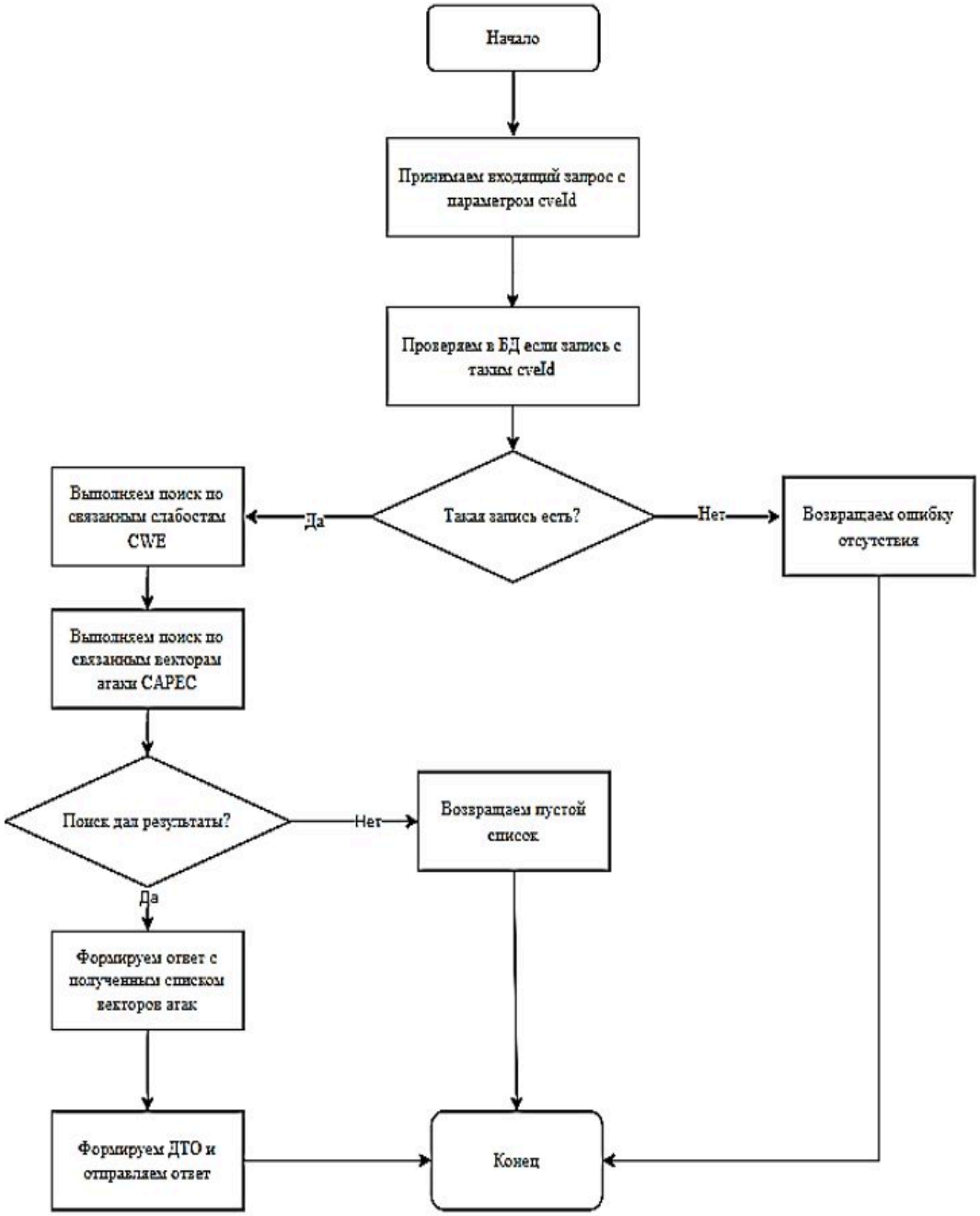

Рис. 9. Алгоритм получения информации о связанных векторах атаки CAPEC по id уязвимости CVE

5) если поиск не успешен (то есть не были найдены связанные векторы атаки CAPEC), возвращаем пустой список,

6) если поиск успешен, формируем ответ с полученным списком векторов атак. Это может включать в себя создание нового объекта или структуры данных, которая

содержит информацию о каждом векторе атаки, включая его идентификатор, название, описание, уровень риска и другие сведения,

7) наконец, формируем DTO и отправляем ответ. В данном случае, DTO может содержать список всех векторов атак CAPEC, связанных с данной уязвимостью CVE.

Как и в предыдущем случае, этот алгоритм требует наличия БД или другого источника данных, который содержит информацию о векторах атаки CAPEC, программных ошибках CWE и уязвимостях CVE.

1.3 Алгоритм получения данных по ключевым словам

Алгоритм поиска данных, по ключевым словам для многообразия векторов атак CAPEC и уязвимостей CVE обеспечивает эффективный способ быстро находить и анализировать информацию, связанную с конкретными векторами атак CAPEC и уязвимостями CVE (рис.10). Это может значительно ускорить процесс исследования кибератак и уязвимостей.

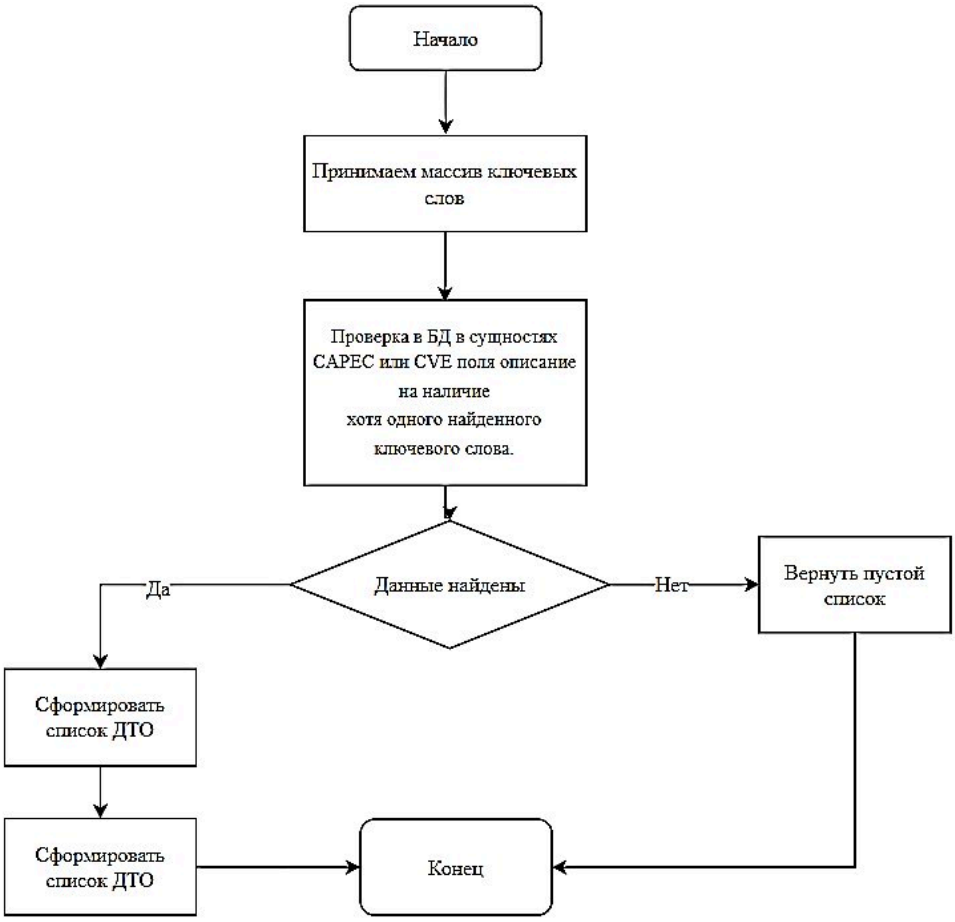

Рис. 10. Алгоритм получения сущности по ключевым словам

Алгоритм получения данных по ключевым словам (рис. 10) включает в себя следующие шаги:

1) прием входных данных. На вход алгоритму подается массив строк, каждая из которых является ключевым словом для поиска,

2) поиск в базе данных. Алгоритм проверяет наличие каждого ключевого слова в полях описания сущностей CAPEC или CVE в базе данных. Это может быть выполнено с помощью SQL-запроса, который ищет совпадения в текстовых полях,

3) формирование списка DTO. Если ключевое слово найдено в поле описания, соответствующая запись добавляется в список DTO. В этом случае DTO будет содержать информацию о найденной сущности CAPEC или CVE,

4) возвращение результатов. После того, как все ключевые слова были проверены, алгоритм возвращает сформированный список DTO. Если ни одно ключевое слово не было найдено, алгоритм возвращает пустой список.

Этот алгоритм позволяет эффективно искать данные по ключевым словам, сохраняя при этом структуру и целостность данных.

1.4 Алгоритм получения потока связанных пар данных

В процессе решения задачи по построению риск-ландшафта для пары вектор атаки – уязвимость, имеет место необходимость в реализации алгоритма получения потока связанных пар данных.

Риск-ландшафт представляет собой двумерное пространство, где ось X отражает вектора атак, а ось Y – уязвимости. Каждая точка на графике представляет собой конкретную уязвимость и вектор атаки, связанные с ней.

Алгоритм используется для обнаружения и связывания этих пар данных. Этот алгоритм включает в себя поиск по базе данных или другим источникам информации, чтобы найти все пары векторов атаки - уязвимость, которые имеют отношение друг к другу.

В контексте этой задачи, поток данных может быть использован для параллельного поиска и связывания пар векторов атаки - уязвимость. Каждый поток может обрабатывать отдельную пару данных, что позволяет ускорить процесс и улучшить производительность.

Важно отметить, что эффективность этого алгоритма зависит от качества и структуры исходных данных. Если данные хорошо организованы и структурированы, алгоритм сможет быстро и эффективно найти все необходимые пары. Если же данные являются неструктурированными или плохо организованными, алгоритм может занять больше времени и ресурсов.

Алгоритм получения потока связанных пар данных (рис. 11) является ключевым инструментом в процессе построения риск-ландшафта для пар вектор атаки - уязвимость. Он позволит быстро и эффективно находить и связывать все необходимые пары данных, что существенно упрощает процесс анализа и исследования.

Пошаговое описание алгоритма (рис. 11) следующее:

1) принимаем входящий поток пар идентификатор уязвимости - идентификатор вектора атаки. Это первый шаг, где мы получаем входные данные. Входные данные представляют собой поток пар, где каждая пара состоит из идентификатора уязвимости и идентификатора вектора атаки,

2) проверяем на соответствие корректности данных. Здесь мы проверяем, являются ли входные данные корректными. Если данные некорректны, мы переходим к следующему шагу,

3) возвращаем код ошибки о некорректности данных. Если данные некорректны, мы возвращаем код ошибки, указывающий на проблему с данными,

4) берем пару по индексу *i*. Если данные корректны, мы берем первую пару из входного потока,

5) делаем запрос в БД. Проверяем наличие вектора атаки в БД по id. Затем мы делаем запрос в базу данных, чтобы проверить, есть ли вектор атаки с данным идентификатором,

6) не добавляем пару в список ответа. Если вектор атаки не найден, мы не добавляем пару в список ответов,

7) преобразуем сущность вектора атаки в ДТО. Добавляем объект в результирующую пару ответа. Если вектор атаки найден, мы преобразуем его в объект DTO и добавляем его в результат,

8) делаем запрос в БД. Проверяем наличие уязвимости в БД по id. Далее мы делаем запрос в базу данных, чтобы проверить, есть ли уязвимость с данным идентификатором,

9) не добавляем пару в список ответа. Если уязвимость не найдена, мы не добавляем пару в список ответов,

10) преобразуем сущность уязвимости в ДТО. Добавляем объект в результирующую пару ответа. Если уязвимость найдена, мы преобразуем ее в объект DTO и добавляем ее в результат,

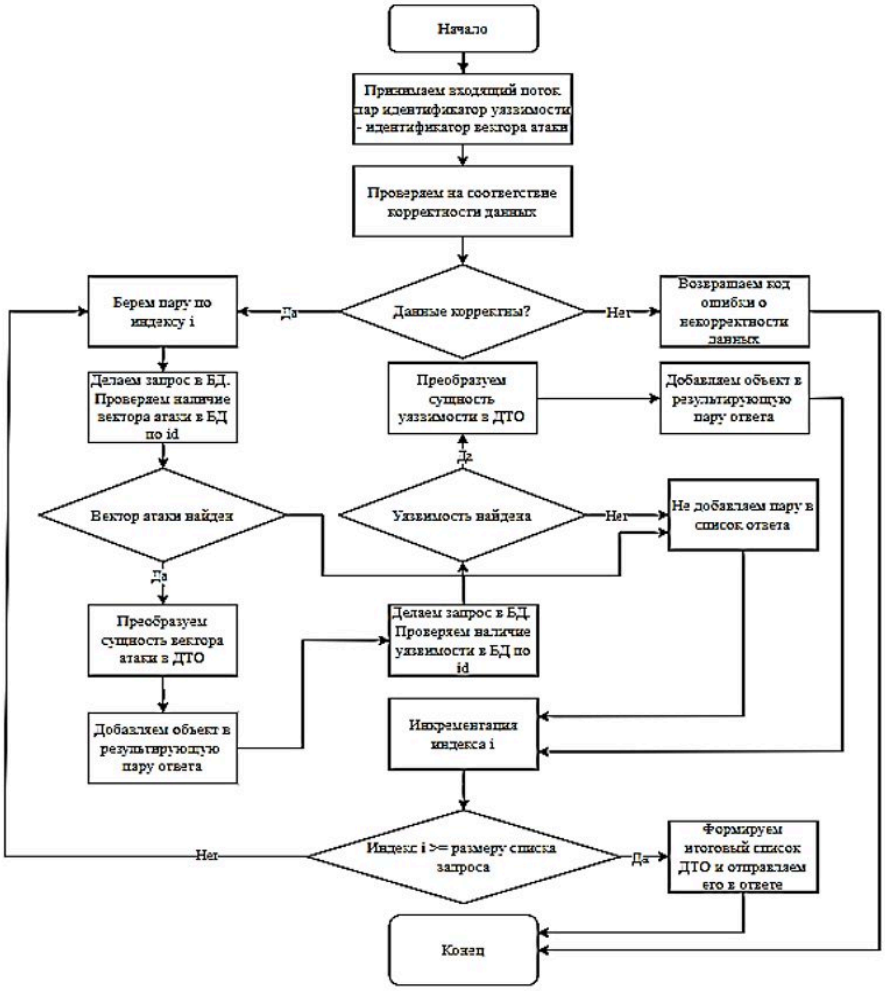

Рис. 11. Алгоритм получения потока пар данных

11) инкрементация индекса *i*: Мы увеличиваем индекс, чтобы перейти к следующей паре в входном потоке,

12) если *i* меньше размера списка запроса, то конец. Если мы достигли конца входного потока, мы завершаем алгоритм,

13) если мы еще не достигли конца входного потока (если *i* не меньше размера списка запроса), мы возвращаемся к шагу 4 и продолжаем процесс.

1.5 Алгоритмы авторизации и регистрации пользователей в системе

Авторизация, регистрация и разграничение по ролям являются критически важными функциями любого программного обеспечения, так как они обеспечивают безопасность и контроль над тем, кто имеет доступ к определенным функциям и данным:

- разграничение по ролям помогает управлять доступом и обеспечивает, что каждый пользователь имеет только те права, которые необходимы для выполнения своих задач,
- авторизация помогает защитить конфиденциальные данные и предотвратить несанкционированный доступ.

Алгоритм регистрации новой учетной записи в системе выглядит следующим образом (рис. 12):

1) получение входных параметров. Это первый шаг, где система собирает необходимые данные от пользователя. Входные параметры включают логин, электронную почту, пароль и роль пользователя. Эти данные обычно предоставляются через форму регистрации или API,

2) проверка на наличие зарегистрированного пользователя с таким логином. Система проверяет базу данных или другой источник данных на наличие пользователя с тем же логином. Если такой

пользователь найден, процесс регистрации останавливается,

3) если логин уже существует, возвращается ошибка наличия логина. Если система обнаруживает, что логин уже используется, она возвращает сообщение об ошибке, информируя пользователя о том, что этот логин уже занят,

4) если логин не существует, выполняется проверка на наличие электронной почты. Если логин доступен, система переходит к следующему шагу и проверяет, не зарегистрирована ли уже электронная почта,

5) если электронная почта уже существует, возвращается ошибка наличия электронной почты. Если система обнаруживает, что электронная почта уже используется, она возвращает сообщение об ошибке, информируя пользователя о том, что этот адрес электронной почты уже зарегистрирован,

6) если электронная почта не существует, сохраняется новая учетная запись пользователя в системе. Если ни логин, ни электронная почта не используются, система создает новую учетную запись пользователя с предоставленными данными и сохраняет ее в базе данных.

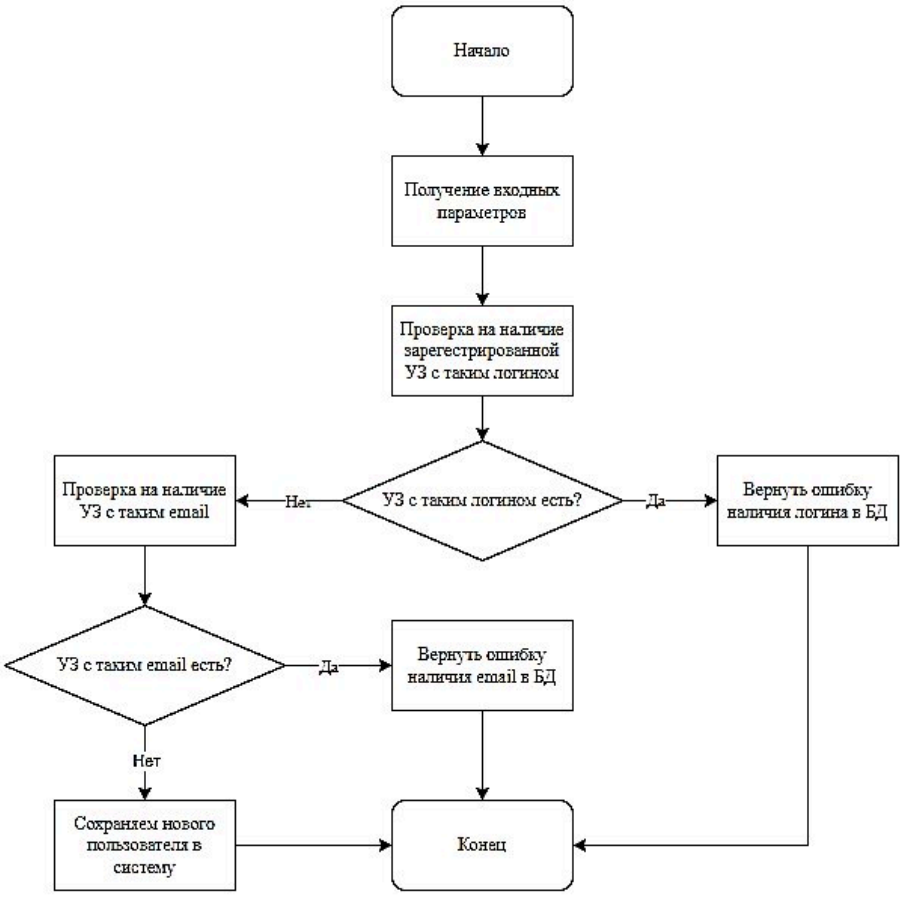

Рис. 12. Алгоритм регистрации и разграничения по ролям учётной записи

Этот алгоритм обеспечивает эффективное управление учетными записями пользователей, предотвращая дублирование логинов и электронных адресов.

В системе могут быть определены различные роли пользователей, каждая из которых предоставляет определенный уровень доступа и функциональности. В разработанной системе каждая учетная запись будет иметь одну из ролей:

- пользователь. Эта роль предоставляет основные возможности для взаимодействия с системой. Пользователи с этой ролью имеют право на получение данных из агрегата. Они могут просматривать информацию, но не могут ее изменять или удалять,
- модератор. Модераторы имеют более широкие права доступа по сравнению с обычными пользователями. Они могут добавлять, изменять и удалять данные из

базы данных. Это делает их идеальными кандидатами для управления контентом и обслуживания пользователей,

- администратор. Администраторы имеют наиболее широкий набор прав доступа. Они имеют доступ ко всем функциям, которые предоставляют модераторы, а также могут создавать новые учетные записи пользователей. Это делает их наиболее мощной ролью в системе, поскольку они могут полностью контролировать ее работу.

Каждая из этих ролей играет важную роль в обеспечении безопасности и эффективного управления системой. Выбор правильной роли для каждого пользователя зависит от его задач и обязанностей в рамках системы.

Алгоритм авторизации

Алгоритм авторизации (рис. 13) состоит из следующих шагов:

1) получение входных параметров для авторизации. Это первый шаг в процессе авторизации. Пользователь предоставляет свои учетные данные, которые включают в себя логин и пароль,

2) проверка на наличие зарегистрированной УЗ с таким логином: Система проверяет, существует ли учетная запись пользователя с предоставленным логином. Если такой учетной записи нет, процесс авторизации завершается, и система возвращает сообщение об ошибке,

3) если нет, верните ошибку отсутствия такой УЗ в БД. Если учетная запись пользователя не найдена, система возвращает сообщение об ошибке, информируя пользователя о том, что такой учетной записи не существует,

4) если да, выполнение процесса авторизации. Если учетная запись пользователя найдена, система переходит к следующему шагу - проверке пароля,

5) если успешно, формируем токен авторизации и возвращаем его в ответе пользователю. Если пароль совпадает с тем, который хранится в системе, система генерирует токен авторизации и отправляет его пользователю. Токен авторизации — это строка символов, которая представляет собой уникальный идентификатор, подтверждающий, что пользователь был успешно авторизован,

6) если нет, верните ошибку авторизации. Если пароль не совпадает с тем, который хранится в системе, система возвращает сообщение об ошибке, информируя пользователя о том, что авторизация не удалась.

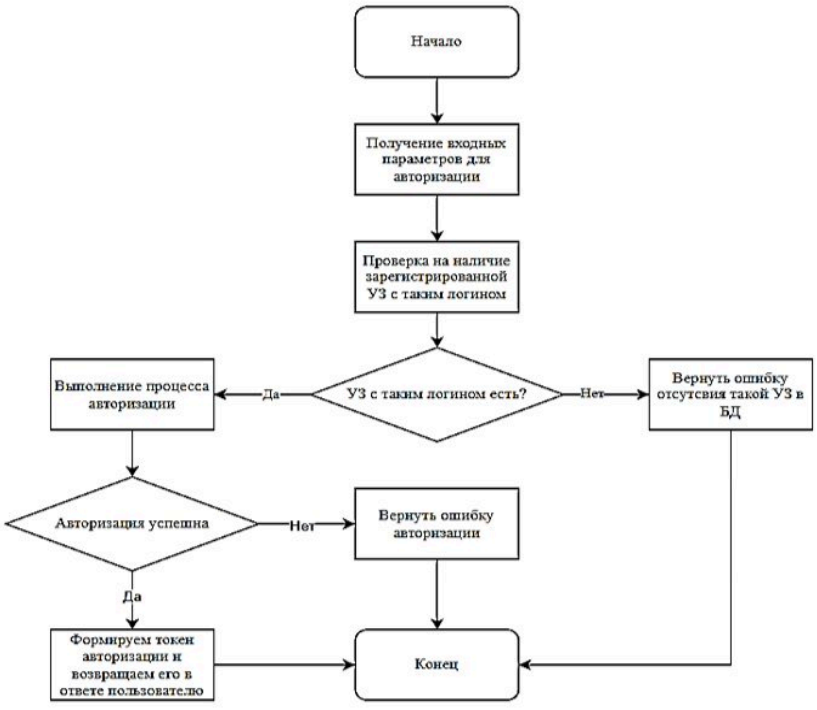

Рис. 13. Алгоритм авторизация пользователя в системе

2 Алгоритмизация наполнения базы данных

Наполнения БД из данных, полученных из XML, включает в себя пять ключевых этапов (рис. 14):

1) чтение XML-файла. Сначала необходимо прочитать XML-файл, из которого будут извлекаться данные. Многие языки программирования поддерживают работу с XML напрямую, что упрощает этот этап,

2) парсинг XML. Затем XML-файл парсится, чтобы извлечь из него данные. Парсинг обычно включает в себя обход XML-структуры и извлечение информации из каждого элемента,

3) преобразование данных. Извлеченные данные затем преобразуются в формат, который можно сохранить в базе данных. Это может включать в себя преобразование типов данных, нормализацию данных и т.д.,

4) запись данных в базу данных. Наконец, преобразованные данные записываются в базу данных. Это может быть выполнено с помощью SQL-запросов или с использованием API, предоставляемого системой управления базами данных,

5) проверка и валидация данных. После записи данных в базу данных, важно провести проверку и валидацию данных, чтобы убедиться, что они были корректно сохранены и что нет ошибок.

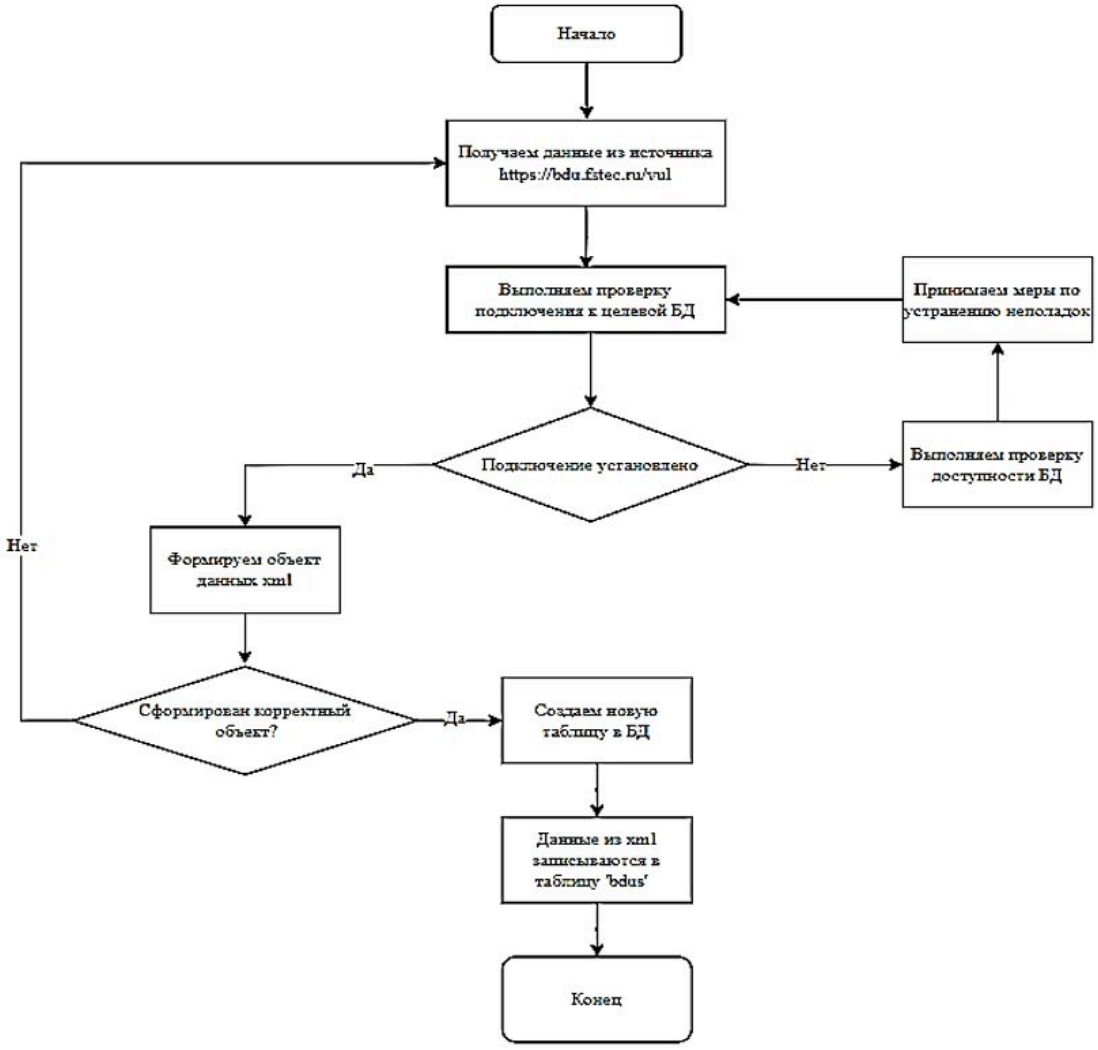

Рис. 14. Алгоритм получения наполнения базы данных из источника BDU ФСТЭК

Алгоритм для получения данных из источника ФСТЭК и записи их в БД выглядит следующим образом (рис. 14):

1) получение данных из источника. Необходимо выполнить HTTP-запрос к URL https://bdu.fstec.ru/vul, чтобы получить данные. Это может быть выполнено с помощью различных библиотек и функций в зависимости от используемого языка программирования,

2) проверка подключения к целевой БД. После получения данных, программный модуль проверяет, может ли он подключиться к целевой базе данных. Это может быть выполнено с помощью функции подключения к базе данных, которая возвращает ошибку, если подключение не удалось,

3) проверка доступности БД и устранение неполадок. Если подключение к базе данных не удалось, программный модуль проверяет, доступна ли база данных. Если база данных недоступна, код принимает меры по устранению неполадок. Это может включать в себя повторную попытку подключения, проверку сетевого подключения и т.д.,

4) формирование объекта данных XML. Если подключение к базе данных было успешным, программный модуль формирует объект данных XML из полученных данных. Это может быть выполнено с помощью функций парсинга XML, которые преобразуют полученные данные в структурированный формат,

5) создание новой таблицы в БД и запись данных из XML. Если объект данных XML был успешно сформирован, программный модуль создает новую таблицу в базе данных и записывает данные из XML в эту таблицу. Это может быть выполнено с помощью SQL-запросов или функций API базы данных,

6) возвращение к этапу получения XML. Если формирование объекта данных XML не было успешным, программный модуль возвращается к этапу получения XML и повторяет процесс.

Этот алгоритм обеспечивает надежное и эффективное получение данных из источника и запись их в базу данных, даже в случае возникновения проблем с подключением или формированием данных.

В свою очередь, алгоритм для получения данных из источника MITRE и записи их в БД, выглядит следующим образом (рис. 15):

1) получение данных из источника: Программный модуль делает HTTP-запрос к URL https://capec.mitre.org/, чтобы получить данные. Это может быть выполнено с помощью различных библиотек и функций в зависимости от используемого языка программирования,

2) проверка подключения к целевой БД. После получения данных, программный модуль проверяет, может ли он подключиться к целевой базе данных. Это может быть выполнено с помощью функции подключения к базе данных, которая возвращает ошибку, если подключение не удалось,

3) проверка доступности БД и устранение неполадок. Если подключение к базе данных не удалось, программный модуль проверяет, доступна ли база данных. Если база данных недоступна, код принимает меры по устранению неполадок. Это может включать в себя повторную попытку подключения, проверку сетевого подключения и т.д.,

4) формирование объекта данных в JSON формате. Если подключение к базе данных было успешным, программный модуль формирует объект данных в формате JSON из полученных данных. Это может быть выполнено с помощью функций парсинга JSON, которые преобразуют полученные данные в структурированный формат,

5) запись данных из JSON в таблицу 'capec'. Если объект данных JSON был успешно сформирован, программный модуль записывает эти данные в таблицы 'capec' в базе данных. Это может быть выполнено с помощью SQL-запросов или функций API базы данных,

6) возвращение к этапу получения XML. Если формирование объекта данных JSON не было успешным, программный модуль возвращается к этапу получения данных из источника повторяет процесс.

Этот алгоритм обеспечивает надежное и эффективное получение данных из источника и запись их в базу данных, даже в случае возникновения проблем с подключением или формированием данных.

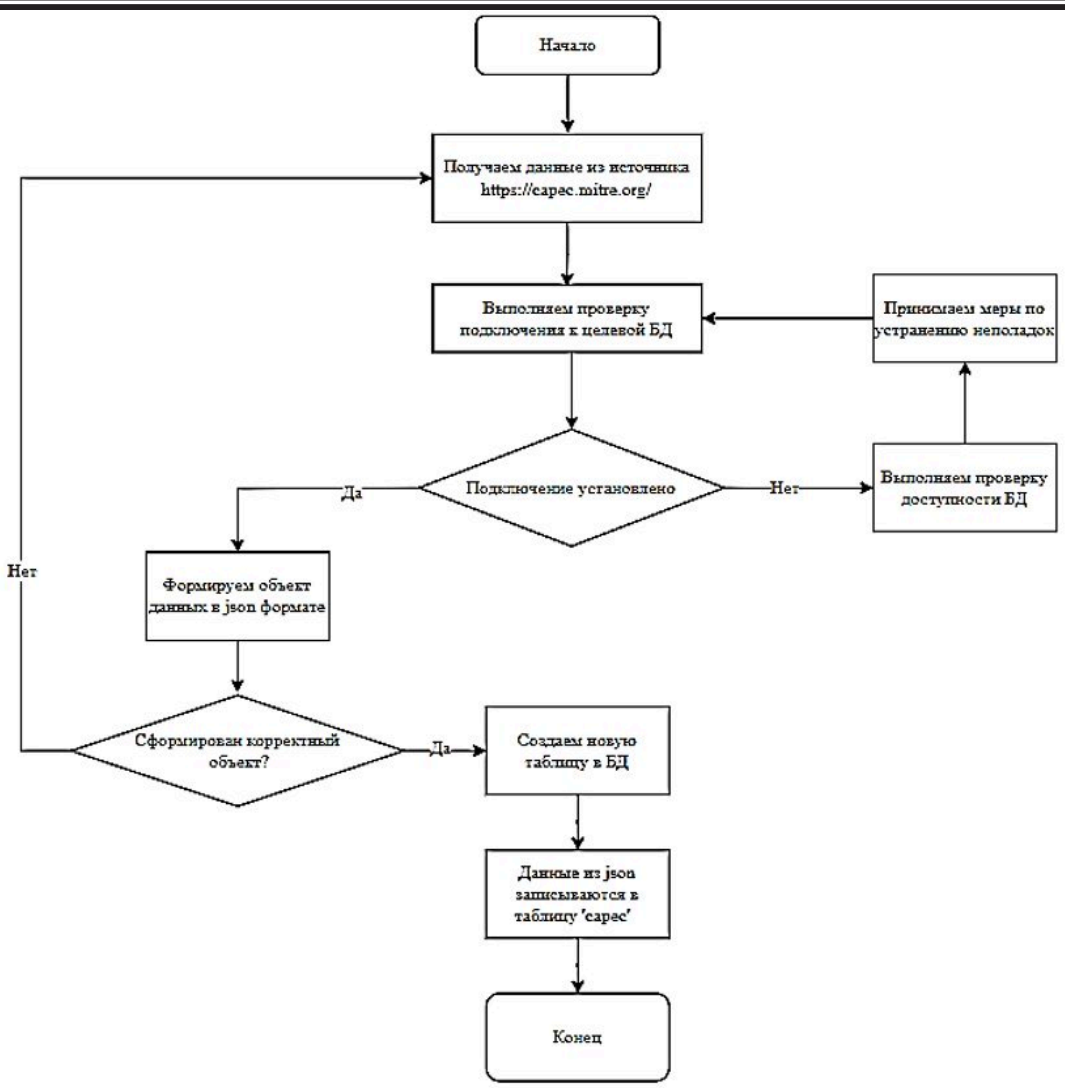

Рис. 15. Алгоритм формирования базы данных для Capec

Описание алгоритма для получения данных из API и записи их в БД (рис. 16) следующее:

1) установка URL API и заголовков для последующего запроса. Программный модуль устанавливает URL API и необходимые заголовки для последующего запроса. Это может быть выполнено с помощью переменных или конфигурационных файлов,

2) выполнение GET запроса к API. Программный модуль выполняет GET запрос к API, используя указанный URL и заголовки. Это может быть выполнено с помощью функций HTTP-запросов, таких как fetch() или axios(),

3) повтор выполнения запроса при ошибке. Если API возвращает ошибку, программный модуль повторяет запрос. Это может быть выполнено с помощью цикла или рекурсии,

4) получение ответа от API и преобразование его в формат JSON. Если запрос был успешным, программный модуль получает ответ от API и преобразует его в формат JSON. Это может быть выполнено с помощью функций парсинга JSON,

5) установление соединения с базой данных: Программный модуль устанавливает соединение с базой данных, используя указанные параметры подключения. Это может быть выполнено с помощью функций подключения к базе данных,

6) проверка доступности БД и устранение неполадок. Если подключение к базе данных не установлено, программный модуль проверяет, доступна ли база данных. Если база данных недоступна, код принимает меры по устранению неполадок,

7) формирование объекта данных в JSON формате. Если подключение к базе данных было успешным, программный

модуль формирует объект данных в формате JSON из полученных данных,

8) возвращение к запросу получения данных при ошибке. Если формирование объекта данных JSON не было успешным, программный модуль возвращается к запросу получения данных и повторяет процесс,

9) создание новой таблицы в БД. Если формирование объекта данных JSON было успешным, программный модуль создает новую таблицу в базе данных. Это может быть выполнено с помощью SQL-запросов,

10) запись данных из JSON в таблицу 'cves'. Наконец, программный модуль записывает данные из объекта JSON в таблицу 'cves' в базе данных. Это может быть выполнено с помощью SQL-запросов или функций API базы данных.

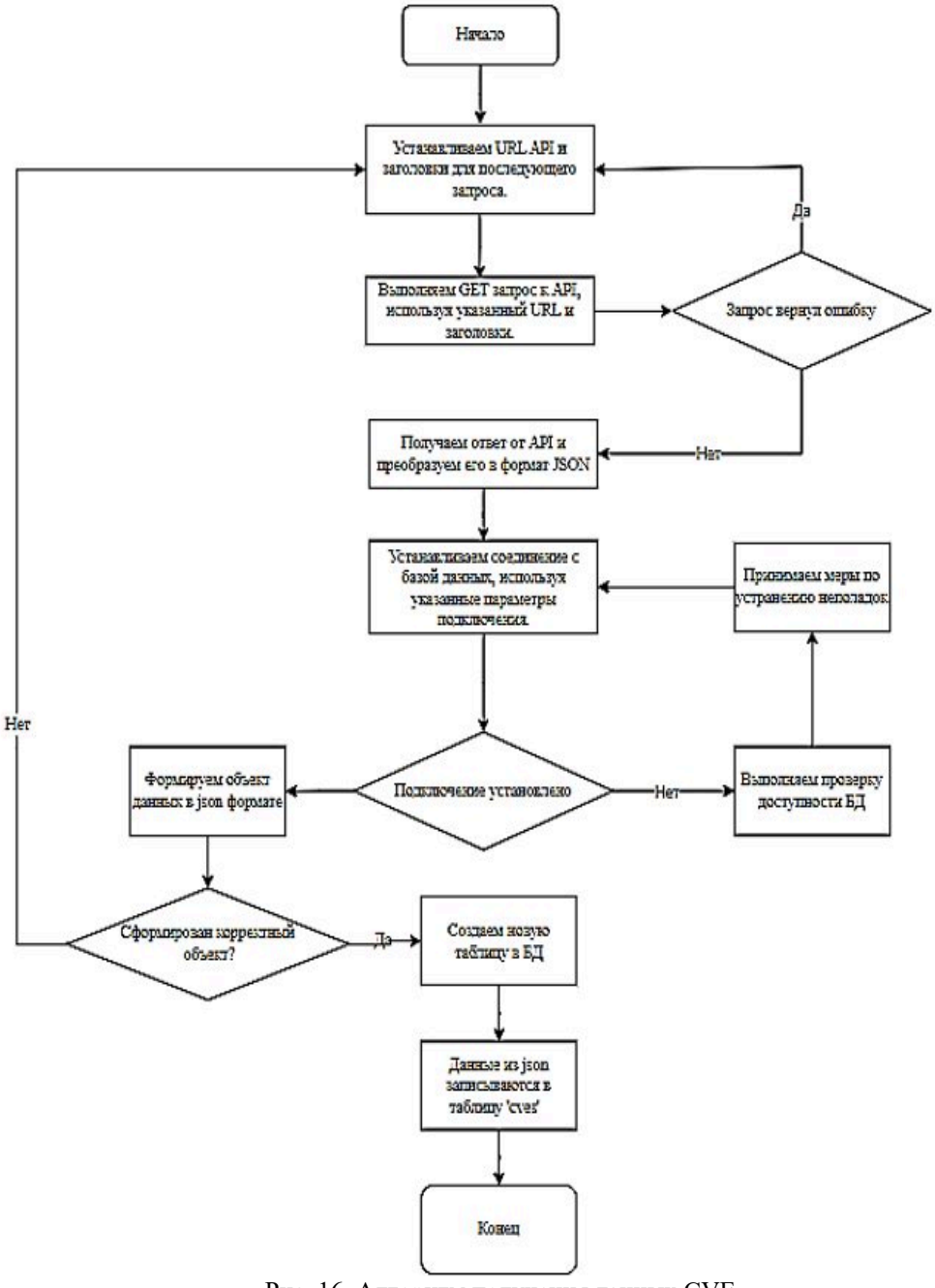

Рис. 16. Алгоритм получения данных CVE

3 Алгоритмизация потока регулярного обновления базы данных уязвимостей

В контексте безопасности информационных систем, уязвимости CISA KEV (Common Vulnerabilities and Exposures) являются важным источником информации. Они регулярно обновляются, чтобы отражать текущее состояние угроз безопасности. Однако, это также создает необходимость в регулярном обновлении собственных БД или систем, которые используют эти данные.

Чтобы автоматизировать этот процесс и обеспечить актуальность данных, был реализован алгоритм, который запускается по таймеру. Эта функция выполняет запрос к API CISA KEV, чтобы получить самые свежие данные об уязвимостях. Это обеспечивает, что наша система всегда имеет доступ к самым актуальным и точным данным.

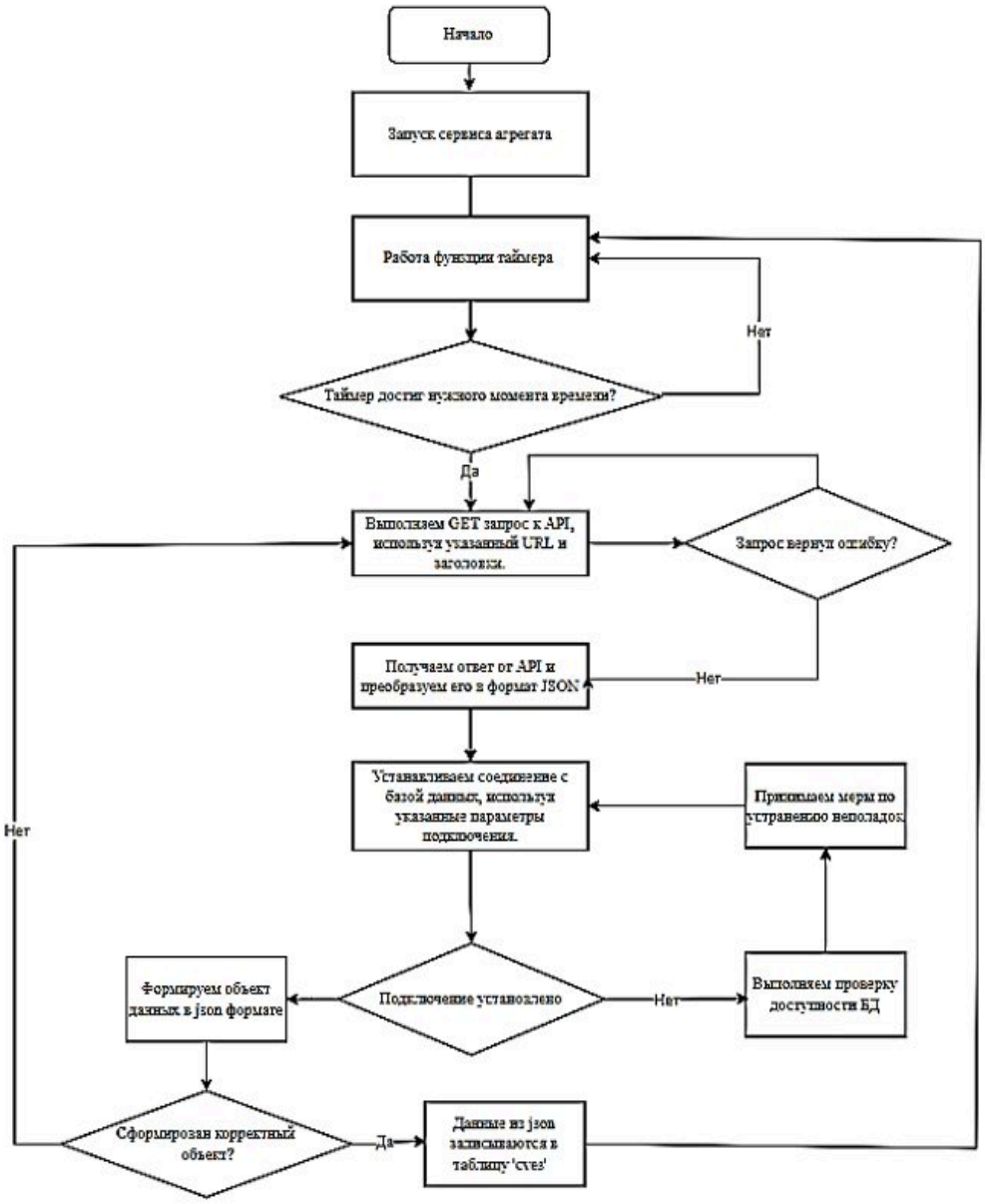

Рис. 17. Алгоритм функции таймера обновления

Функция обеспечивается следующим образом:

1) задается интервал времени, через который функция будет запускаться. Это может быть каждый день, каждую неделю или любой другой период, в зависимости от потребностей,

2) когда наступает время запуска функции, она выполняет HTTP GET запрос к API CISA KEV. Этот запрос обычно включает

в себя указание конкретного типа данных, которые нужны (например, все уязвимости за последний месяц),

3) API CISA KEV возвращает данные в формате JSON, который затем преобразуется и сохраняется в нашей системе,

4) после сохранения данных, функция завершает свою работу и ждет следующего запуска.

Этот процесс (рис. 17) повторяется каждый раз, когда функция запускается, обеспечивая доступ к самым свежим и актуальным данным об уязвимостях.

Заключение

В работе предложено описание требований для создаваемого программного инструментария и основных информационных потоков, включая агрегацию данных о многообразии векторов атак и уязвимостей, а также построение риск ландшафта для пар вектор атаки - уязвимость.

Построена архитектура создаваемого программного инструментария, включая:

- алгоритмизацию сервиса агрегата данных о многообразии векторов атак и уязвимостей;
- унифицированные алгоритмы CRUD операций сервиса, агрегирующего данные о многообразии векторов атак и уязвимостей;
- алгоритмы агрегации связанных данных о многообразии векторов атак и уязвимостей;
- алгоритм получения данных по ключевым словам;
- алгоритм получения потока связанных пар данных;
- алгоритмы авторизации и регистрации пользователей в системе;
- алгоритмизацию наполнения базы данных;
- алгоритмизацию потока регулярного обновления базы данных уязвимостей.

В качестве перспективы развития в настоящее время прорабатываются аспекты применения нейросетей [15, 16] в автоматизации противодействия кибератакам.

Список литературы

1. Организационно-правовая защита сетей / Г. А. Остапенко, Д. В. Щербакова, А. О. Калашников и др. Под ред. академика РАН Д. А. Новикова. М: Горячая линия - Телеком, 2023. 228с.:

2. The Common Attack Pattern Enumeration and Classification (CAPEC) URL: https://capec.mitre.org/ (дата обращения 25.01.23).

3. NIST Information Technology Laboratory National Vulnerability Database. URL: https://nvd.nist.gov/vuln (дата обращения 25.01.23).

4. MITRE ATT&CK. URL: https://attack.mitre.org/matrices/enterprise/ (дата обращения 25.01.23).

5. NIST Common Vulnerability Scoring System Calculator. URL: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator, (дата обращения 25.01.23).

6. База данных угроз безопасности информации. URL: https://bdu.fstec.ru/threat (дата обращения 25.01.23).

7. Каталог известных эксплуатируемых уязвимостей (CISA KEV). URL: https://www.cisa.gov/known-vulnerabilities-catalog (дата обращения 25.01.23)..

8. Остапенко Г.А. Совершенствование организационно-правового обеспечения информационной безопасности предприятия: формирование риск-ландшафта сетевых атак / Г.А. Остапенко, Д.В. Щербакова, Т.Ю. Мирошниченко, А.А. Остапенко, А.Ю. Пекло // Информация и безопасность. 2023. Т. 26. Вып. 2. С. 203-210.

9. С. Пекло А.Ю. Атаки типа «сетевая разведка»: риск-ландшафт и частная политика информационной безопасности предприятия / А.Ю. Пекло, Г.А. Остапенко, Д.В. леа, А.А. Осталенко // Информация и безопасность. 2023. Т. 26. Вып. 2. С. 235-246.

10. Хромых С.А. Сетевые атаки на уровне приложений: риск-ландшафт и частная политика информационной безопасности предприятия / С.А. Хромых, Г. А. Остапенко, Д. В. Щербакова, А. А. Остапенко // Информация и безопасность. 2023. Т. 26. Вып. 2. С. 261-276.

11. Остапенко Г.А. Организационно-правовая защита от сетевых атак: методики формирования частных политик, регламентов

и инструкций обеспечения безопасности организации (часть I) / Г.А. Остапенко, Д.В. Щербакова, Т.Ю. Мирошниченко, А.А. Остапенко, А.С. Кривошеин // Информация и безопасность. 2023. Т. 26. Вып. 3. С. 329-340.

12. Остапенко Г.А. Организационно-правовая защита от сетевых атак: методики формирования частных политик, регламентов и инструкций обеспечения безопасности организации (часть II) / Г.А. Остапенко, Д.В. Щербакова, Т.Ю. Мирошниченко, А.А. Остапенко, А.Г. Краснобородкин // Информация и безопасность. 2023. Т. 26. Вып. 3. С. 329-340.

13. Остапенко Г.А. Организационно-правовая защита от сетевых атак: методики формирования частных политик, регламентов и инструкций обеспечения безопасности организации (часть III) / Г.А. Остапенко, Д.В. Щербакова, Т.Ю. Мирошниченко, А.А. Остапенко, А.Г. Краснобородкин // Информация и безопасность. 2023. Т. 26. Вып. 3. С. 341-358.

14. Остапенко А.Г. Проектная деятельность: научно-методическое развитие в направлении внедрения средств искусственного интеллекта для обеспечения организационно-правовой защиты корпоративных сетей / А.Г. Остапенко, Д.В. Щербакова, А.А. Остапенко, Д.А. Нархов // Информация и безопасность. 2023. Т. 26. Вып. 3. С. 447-454.

15. Остапенко Г.А. Нейросетевые задачи и компетенции проектной деятельности по созданию защищённых автоматизированных информационных систем / Г.А. Остапенко, А.П. Васильченко // Информация и безопасность. 2023. Т. 26. Вып. 4. С. 579-586.

16. Остапенко Г.А. Методики регламентации обеспечения информационной безопасности атакуемых автоматизированных систем / Г.А. Остапенко, А.П. Васильченко // Информация и безопасность. 2023. Т. 26. Вып.4. С. 597-602.

Финансовый университет при Правительстве Российской Федерации
Financial University under the Government of the Russian Federation

Воронежский государственный технический университет
Voronezh State Technical University

Поступила в редакцию 07.01.2024

Информация об авторах

Остапенко Григорий Александрович – д-р техн. наук, проректор, Финансовый университет при Правительстве Российской Федерации, e-mail: ostg@mail.ru

Васильченко Алексей Павлович – аспирант, Финансовый университет при Правительстве Российской Федерации, e-mail: zainichek@uandex.ru

Остапенко Александр Алексеевич – аспирант, Воронежский государственный технический университет, e-mail: alexostap123@gmail.com

Нестеров Дмитрий Сергеевич – студент, Воронежский государственный технический университет, e-mail: NesterovWork@yandex.ru

Дубов Андрей Сергеевич – студент, Воронежский государственный технический университет, e-mail: whodatandrey@gmail.com

Старцев Валерий Александрович – студент, Воронежский государственный технический университет, e-mail: startsev@keeneye.pro

AUTOMATED KNOWLEDGE BANK AND CYBER ATTACK RISK AND VULNERABILITY CALCULATOR (PART II)

G.A. Ostapenko, A.P. Vasilchenko, A.A. Ostapenko, D.S. Nesterov, A.S. Dubov, V.A. Startsev

In the development of the instrumentation of risk-analysis, the work proposes original risk assessment algorithms that take into account the violation of the qualities of information and the performance of secure systems and networks under the influence of a variety of attack vectors and vulnerabilities used by them. Algorithmization of the developed techniques in the form of risk calculators of partial or complete loss of value, confidentiality and accessibility of information, taking into account the specifics of the protected object and many vulnerabilities used. A series of algorithms for the formation of the Bank of Kiberataka and Vulnerability Bank in the form of aggregated regulations of various stages of opposition to invasions, including current response and elimination of consequences in relation to registered incidents, is proposed. Algorithmization allows the user in the dialog box to get practical recommendations from the bank to combat the variety of attack scenarios and gaps used by attackers.

Keywords: system, security, risk, damage, probability, regulations, knowledge and data base, algorithm.

Submitted 07.01.2024

Information about the authors

Grigory A. Ostapenko – Dr. Sc. (Technical), Vice-Rector, Financial University under the Government of the Russian Federation, e-mail: ostg@mail.ru

Alexey P. Vasilchenko – graduate student, Financial University under the Government of the Russian Federation, e-mail: zainichek@uandex.ru

Alexander A. Ostapenko, graduate student, Voronezh State Technical University, e-mail: alexostap123@gmail.com

Dmitry S. Nesterov – student, Voronezh State Technical University, e-mail: NesterovWork@yandex.ru

Andrey S. Dubov – student, Voronezh State Technical University, e-mail: whodatandrey@gmail.com

Valery A. Startsev – student, Voronezh State Technical University, e-mail: startsev@keeneye.pro