МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт Факультет информационных технологий и электроники	Кафедра «Информационная безопасность систем и технологий»
Специальность 10.05.03 –	Информационная безопасность автоматизированных систем
Специализация -	Защищенные автоматизированные системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

БАЗОВАЯ СИСТЕМА АВТОМАТИЗАЦИИ ПОИСКА ПОТЕНЦИАЛЬНЫХ УЯЗВИМОСТЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИ ПРОВЕДЕНИИ СЕРТИФИКАЦИОННЫХ ИСПЫТАНИЙ СРЕДСТВ ЗАЩИТЫ ИНФОРМАЦИИ В СИСТЕМЕ СЕРТИФИКАЦИИ ФСТЭК РОССИИ

Студент _____ Ворошилло Дмитрий Борисович

Руководитель _____ Маркин В.В.

Нормоконтролёр _____ Фатеев А.Г.

Рецензент ведущий инженер ПФ АО «НТЦ «Атлас» _____ Безяев А.В.

ОТДЕЛ КАДРОВ

Работа допущена к защите (протокол заседания кафедры от 25.06.21 № 16)

Заведующий кафедрой _____ С.Л. Зефиров

Работа защищена с отметкой отлично (протокол заседания ГЭК от 28.06.21 № 1)

Секретарь ГЭК _____ А.П. Иванов

Пенза 2021

Оглавление

Введение 4
1 Анализ общедоступных баз данных уязвимостей 5
1.1 Анализ нормативно-методической базы ФСТЭК России в части сертификационных испытаний СЗИ 5
1.2 Изучение формата и практики использования общедоступных баз данных уязвимостей при проведении сертификационных испытаний СЗИ ... 8
1.3 Результаты анализа нормативно-методической базы ФСТЭК России .. 9
1.4 Анализ БДУ ФСТЭК России 11
1.5 Анализ БДУ CVE 13
1.6 Анализ БДУ NVD 17
1.7 Общие сведения о базе «Security Focus» 18
1.8 Выявленные проблемы общедоступных БДУ 19
2 Разработка архитектуры ПК САПУ 20
2.1 Общие сведения 20
2.2 Состав системы 21
2.3 Выбранные технологии 21
3 Разработка серверной части ПК САПУ 25
3.1 Получение данных из NVD 25
3.2 Получение данных из CNVD 27
3.3 Получение данных из ФСТЭК 28
3.4 Проектирование API 29
3.5 Модуль работы с авторизацией 31
3.6 Модуль работы с уязвимостями 35
4 Проектирование пользовательского интерфейса ПК САПУ 44
5 Разработка клиентской части ПК САПУ 47
6 Вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие 55
Заключение 60
Список используемых источников 61
Приложение А Код алгоритма обработки БДУ NVD 62
Приложение Б Код алгоритма обработки БДУ CNVD 69

Приложение В Код алгоритма обработки БДУ ФСТЭК 72

Приложение Г Контроллер для автоматизированного обновления БДУ CNVD и ФСТЭК 76

Приложение Д Протокол проверки на оригинальность в системе «Антиплагиат.ВУЗ» 78

Введение

В настоящее время при проведении сертификационных испытаний и тематических исследований автоматизированных систем в защищенном исполнении в системах сертификации ФСТЭК, ФСБ и МО России со стороны экспертных организаций (далее – ЭО) выдвигается требование анализа на защищенность от потенциальных уязвимостей программного обеспечения (далее – ПО). Данное требование следует как непосредственно из руководящих документов, так и из замечаний ЭО к разрабатываемым программам, методикам исследований и отчетным материалам.

Проведение анализа уязвимостей ПО в настоящее время является обязательной активностью экспертов испытательных лабораторий в ходе сертификационных испытаний, выполняемых по линии системы сертификации ФСТЭК России. Данный вид работ выполняется как при сертификации на соответствие требованиям утвержденных ФСТЭК России профилей защиты (ПЗ), так и при испытаниях на соответствие требованиям Технических условий.

Задачи дипломной работы:

– провести анализ нормативно-методической базы ФСТЭК России в части сертификационных испытаний СЗИ;

– изучить формат и практики использования общедоступных баз данных уязвимостей при проведении сертификационных испытаний СЗИ;

– выявить основные проблемы общедоступных баз данных уязвимостей;

– разработать серверную часть программного комплекса системы автоматизации поиска потенциальных уязвимостей (далее - ПК САПУ);

– спроектировать пользовательский интерфейс ПК САПУ;

– разработать клиентскую часть ПК САПУ;

– рассмотреть вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации и привести методы и средства физической культуры, снижающие их воздействие.

1 Анализ общедоступных баз данных уязвимостей

1.1 Анализ нормативно-методической базы ФСТЭК России в части сертификационных испытаний СЗИ

Ключевыми нормативными документами для проведения испытаний по линии ФСТЭК России являются:

– «Положение о системе сертификации средств защиты информации» (утверждено приказом ФСТЭК России от 3.04.18 г. №55)[1];

– Профили защиты для различных типов СЗИ;

– «Требования по безопасности информации, устанавливающим уровни доверия к средствам технической защиты информации и средствам обеспечения безопасности информационных технологий» (утверждены приказом ФСТЭК России от 02.06.2020 г. № 76) [2];

– «Методика выявления уязвимостей и не декларированных возможностей в программном обеспечении» (далее – Методика НДВ) (утверждена ФСТЭК России 11 февраля 2019 г.) [3].

Перечень работ, проводимых в части выявления уязвимостей и НДВ для 4 уровня доверия СЗИ включает:

– анализ документации (ПОД.1);

– подготовка исследовательского стенда (ПОД.2);

– анализ архитектуры (КАО.1);

– идентификация потенциальных уязвимостей (КАО.2);

– тестирование на проникновение (КАО.3);

– синтаксический анализ исходного кода (САО.1);

– фаззинг-тестирование (ДАО.1);

– ручной анализ участков исходного/восстановленного кода (РАО.1).

Согласно Методике НДВ [3] должен быть проведен анализ архитектуры СЗИ с целью выявления в нем потенциально опасных функциональных возможностей и архитектурных уязвимостей.

В ходе анализа архитектуры СЗИ должны быть определены состав и назначение всех программ (файлов) и программных модулей, в том числе сторонних разработчиков (заимствованное ПО), а также интерфейсов их взаимодействия между собой и с другими программами.

В результате анализа состава и назначения программ (файлов) и программных модулей, в том числе сторонних разработчиков, а также интерфейсов их взаимодействия между собой и с другими программами выявляются архитектурные уязвимости и потенциально опасные функциональные возможности.

Идентификация архитектурных уязвимостей осуществляется на основе сведений о программных ошибках архитектуры программного обеспечения, содержащихся в соответствующих общедоступных базах данных уязвимостей (БДУ). В отношении каждой выявленной потенциально опасной функциональной возможности СЗИ должен быть проведен анализ возможности возникновения угроз безопасности.

Согласно Методике НДВ должна быть проведена идентификация уязвимостей СЗИ, предусматривающая выявление не устраненных известных уязвимостей в СЗИ и средах его функционирования, а также потенциальных уязвимостей СЗИ.

К сведениям, подлежащим анализу, относятся публикации разработчика об СЗИ (включая официальный сайт), базы данных уязвимостей, ошибок, шаблонов или векторов компьютерных атак. Для проведения анализа могут использоваться следующие источники:

– банк данных угроз безопасности информации (bdu.fstec.ru);

– зарубежные базы данных уязвимостей и недостатков (ошибок) (Common Vulnerabilities and Exposures, Common Weakness Enumeration и иные);

– базы данных шаблонов компьютерных атак (Common Attack Pattern Enumeration and Classification и иные);

– специальные публикации и форумы, содержащие отзывы пользователей;

– результаты исследований, полученные сторонними исследователями.

Поиск сведений об СЗИ и его уязвимостях необходимо проводить в тех источниках, которые имеют или могут иметь отношение к СЗИ. При определении требуемой широты поиска необходимо исходить из результатов анализа представления об СЗИ. По результатам поиска должен быть определен перечень источников, в которых содержатся сведения об уязвимостях программного обеспечения, соответствующего исследуемому СЗИ. В ходе поиска также должны быть выявлены источники, содержащие сведения об известных уязвимостях заимствованных программ, программных модулей, программного кода, интегрированного в СЗИ.

Анализ источников, содержащих сведения об уязвимостях СЗИ, должен предусматривать идентификацию известных уязвимостей, не устраненных в СЗИ и в заимствованных программах, программных модулях, программном коде на момент проведения исследований, или подтверждение того, что проанализированные источники не содержат таких сведений.

Наряду с выявлением известных уязвимостей должна проводиться идентификация потенциальных уязвимостей, предусматривающая выявление и анализ всех возможных недостатков (слабостей), использование которых может привести к нарушению целостности СЗИ или нарушению конфиденциальности, целостности, доступности обрабатываемых (защищаемых) данных. Идентификация потенциальных уязвимостей осуществляется на основе анализа информации о типовых уязвимостях, ошибках, недостатках, характерных для программного обеспечения, аналогичного по архитектуре исследуемому СЗИ.

Для каждой идентифицированной потенциальной уязвимости должны быть определены:

– источник сведений об уязвимости;

– ошибка (недостаток), которая может привести к потенциальной уязвимости;

– функции безопасности, которые могут быть нарушены в результате использования потенциальной уязвимости;

– возможность использования потенциальной уязвимости в конкретной среде или средах функционирования;

– шаблон или вектор атаки (нескольких атак), которые могут быть реализованы в отношении объекта оценки с использованием потенциальной уязвимости;

– количество времени, уровень компетенции, уровень знания объекта оценки, уровень доступа к СЗИ, оборудование, необходимое для использования идентифицированных уязвимостей.

Идентифицированные потенциальные уязвимости подлежат исследованию в ходе тестирования на проникновения (КАО.3), поэтому максимальное количество найденных уязвимостей должно быть отсеяно на этапе экспертной оценки с учетом модели нарушителя, архитектурных особенностей СЗИ и условий его эксплуатации.

1.2 Изучение формата и практики использования общедоступных баз данных уязвимостей при проведении сертификационных испытаний СЗИ

Методика НДВ требует проведения большого объема работ с использованием метода экспертной оценки, плохо поддающегося автоматизации. Однако, очевидно, что некоторые выполняемые технологические операции могут быть автоматизированы, что приведет к снижению трудоемкости и сокращению времени проведения сертификационных испытаний. Автоматизация становится особенно актуальной при использовании в ПО СЗИ больших объемов заимствованного

ПО, которое в современных СЗИ может доходить от 80% до 90% от общего объема кода.

1.3 Результаты анализа нормативно-методической базы ФСТЭК России

По результатам анализа нормативно-методической базы ФСТЭК России в части выявления уязвимостей и НДВ были определены следующие возможные направления автоматизации:

– автоматизация анализа архитектурных уязвимостей СЗИ. Анализ архитектурных уязвимостей проводится на этапе КАО.1 и может быть частично автоматизирован в части получения сведений об уязвимостях и методах проведения атак с использованием каталогов CWE и CAPEC. Для нейтрализации опубликованных и потенциальных уязвимостей могут быть также предложены целевые меры защиты. Источником целевых мер защиты могут являться сведения каталогов CWE и CAPEC о методах обнаружения отдельных типов уязвимостей и защиты от атак с их использованием;

– частичная автоматизация идентификации уязвимостей СЗИ (КАО.2) с использованием общедоступных БДУ. Автоматизация подразумевает получение интегральной информации об уязвимостях заимствованного ПО СЗИ с использованием различных источников. БДУ могут отличаться как по перечню уязвимостей, так и по формату хранимых данных. Основной задачей автоматизации является представление результатов разных БДУ в едином формате, а также формирование и экспорт результатов для последующей экспертной оценки;

– автоматизация формирования перечня общих мер защиты СЗИ и его использование при частичной автоматизации экспертной оценки найденных опубликованных уязвимостей с точки зрения применимости к конкретному СЗИ.

В качестве основных источников информации об уязвимостях и шаблонах атак могут использоваться:

– каталог потенциальных уязвимостей (слабостей) CWE (Common Weakness Enumeration). Многие БДУ для классификации ошибок ПО используют каталог CWE (Common Weakness Enumeration). Этот каталог представляет собой формируемый международным сообществом официальный справочник ошибок программных продуктов и служит стандартным измерительным средством эффективности механизмов защиты;

– CAPEC (Common Attack Pattern Enumeration) (https://capec.mitre.org). Каталог CAPEC описывает возможный способ атаки с использованием различных типов уязвимостей;

– банк данных угроз безопасности информации ФСТЭК России (https://bdu.fstec.ru) (далее – БДУ ФСТЭК). БДУ ФСТЭК это общедоступная база данных угроз и уязвимостей безопасности, которая формируется Государственным научно-исследовательским институтом проблем технической защиты информации (ГНИИ ПТЗИ) ФСТЭК России. Доступ к базе открыт в марте 2015 года. Целью ведения Банка данных угроз безопасности информации является повышение информированности заинтересованных лиц о существующих угрозах безопасности информации в информационных (автоматизированных) системах. БДУ ФСТЭК России содержит сведения об основных угрозах безопасности информации и уязвимостях, в первую очередь, характерных для государственных информационных систем и автоматизированных систем управления производственными и технологическими процессами критически важных объектов. Сведения об угрозах безопасности информации и уязвимостях программного обеспечения, содержащиеся в Банке данных угроз безопасности информации, не являются исчерпывающими и могут быть дополнены по результатам анализа угроз безопасности информации и уязвимостей в конкретной информационной (автоматизированной) системе с учетом особенностей ее эксплуатации;

– база данных уязвимостей ПО «Common Vulnerabilities and Exposures» (https://cve.mitre.org) (далее – CVE). БДУ CVE представляет собой

международный проект, разработанный американской некоммерческой исследовательской корпорацией MITRE Corporation и является основным стандартом в области унифицированного именования и регистрации обнаруженных уязвимостей ПО;

– база данных уязвимостей ПО «National Vulnerability Database» (https://nvd.nist.gov) (далее – NVD). БДУ NVD (National Vulnerability Database) национальная база данных уязвимостей США разработана отделением компьютерной безопасности (Computer Security Division) национального института стандартов и технологий (National Institute of Standards and Technology) США. Данная база представляет собой хранилище данных уязвимостей, основанное на стандартах протокола автоматизации содержимого безопасности SCAP. База NVD объединила в себе описание уязвимостей, названия ПО с этими уязвимостями и оценки опасности уязвимостей;

– база данных уязвимостей ПО China National Vulnerability Database (CNVD) (https://www.cnvd.org.cn). БДУ CNVD является аналогом БДУ NVD в Китае, однако содержит большее количество записей и существенно быстрее обновляется;

– база данных уязвимостей ПО сообщества Security Focus (https://www.securityfocus.com). База Security Focus разработана в 1990 г. и принадлежит компании Symantec. В данной БДУ при добавлении уязвимости присваивается уникальный идентификатор BID и определяется ее класс. BID используется многими программными продуктами для ссылок на уязвимости или атак.

1.4 Анализ БДУ ФСТЭК России

Данная общедоступная база данных угроз и уязвимостей формируется Государственным научно-исследовательским институтом проблем технической защиты информации (ГНИИ ПТЗИ) ФСТЭК России. Доступ к базе открыт в марте 2015 года.

Последнее обновление базы данных было выполнено 01.07.2019 г. Целью ведения Банка данных угроз безопасности информации является повышение информированности заинтересованных лиц о существующих угрозах безопасности информации в информационных (автоматизированных) системах. БДУ ФСТЭК России содержит сведения об основных угрозах безопасности информации и уязвимостях, в первую очередь, характерных для государственных информационных систем и автоматизированных систем управления производственными и технологическими процессами критически важных объектов. Сведения об угрозах безопасности информации и уязвимостях программного обеспечения, содержащиеся в Банке данных угроз безопасности информации, не являются исчерпывающими и могут быть дополнены по результатам анализа угроз безопасности информации и уязвимостей в конкретной информационной (автоматизированной) системе с учетом особенностей ее эксплуатации.

В БДУ ФСТЭК России включается информация об уязвимостях и угрозах безопасности информации, полученная по результатам выявления и анализа:

– сведений, опубликованных в общедоступных источниках, в том числе в информационно-телекоммуникационной сети Интернет;

– сведений, полученных по результатам работ, проводимых федеральными органами исполнительной власти в рамках своих полномочий (компетенций);

– сведений, поступивших из органов государственной власти, организаций и иных лиц, выполняющих работы по защите информации.

Все хранящиеся в БДУ ФСТЭК России записи имеют текстовое описание уязвимости, дату обнаружения уязвимости, названия, версии и производителей уязвимого ПО, информацию о типе ошибки, классе уязвимости и текущем ее статусе (потенциально возможная либо подтвержденная производителями ПО или независимыми исследователями уязвимость, устранена ли уязвимость в новых версиях ПО). Также записи содержат оценку критичности уязвимости и сопутствующий вектор CVSS, пометку о наличии известных готовых сценариев

эксплуатации уязвимости и возможного результата эксплуатации уязвимости, указание уязвимых аппаратных платформ или ОС, список возможных методов противодействия уязвимости и ссылки на источники дополнительной информации по уязвимости.

БД угроз ФСТЭК ведется с 2014 г. и на 12 мая 2018г. содержит 208 записей. Записи в БДУ ФСТЭК заносятся на русском языке.

БД угроз ФСТЭК доступна в формате XLSX. Информация, содержащаяся в БД угроз доступна по адресу http://bdu.fstec.ru/threat.

Каждой угрозе в БД присваивается идентификатор вида УБИ-XXX, где XXX — уникальный идентификационный номер.

Каждая запись об угрозе содержит следующие поля:

– идентификатор;
– наименование;
– описание;
– источник угрозы (характеристика и потенциал нарушителя);
– объект воздействия;
– нарушаемые характеристики безопасности информации (конфиденциальность, целостность, доступность).

1.5 Анализ БДУ CVE

База данных уязвимостей CVE представляет собой международный проект, разработанный американской некоммерческой исследовательской корпорацией MITRE Corporation и является основным стандартом в области унифицированного именования и регистрации обнаруженных уязвимостей ПО. В рамках проекта MITRE обеспечивает выполнение следующих функции в CVE List:

– поиск и сбор информации об уязвимостях ПО;
– классификация найденных уязвимостей;
– резервирование CVE идентификаторов для найденных уязвимостей;

– актуализация соответствующей информации в двух официальных каталогах: реестре уязвимостей MITRE CVE и базе данных уязвимостей NVD, поддерживаемой NIST.

Последнее обновление базы данных было выполнено 01.07.2019 г. Базы данных CVE и NVD различаются на уровне функциональных возможностей, предоставляемой пользователю, но сами списки записей об уязвимостях идентичный друг другу. CVE выступает первоначальным источником для NVD.

CVE представляет список стандартных идентификаторов (IDs) для общеизвестных уязвимостей информационной безопасности. Идентификаторы CVE (CVE Ids, CVE entries, CVE numbers, CVEs) служат единым языком для эффективного обмена данными между отдельными БД и инструментами сетевой безопасности. Каждый ID CVE включает в себя:

– идентификационный номер;
– краткое описание уязвимости;
– соответствующие ссылки.

Идентификационный номер имеет следующий синтаксис (формат) CVE-YYYY-NNNN и содержит в себе следующие атрибуты:

– CVE (префикс);
– YYYY (год обнаружения уязвимости);
– NNNN (порядковый номер, уникален в рамках года обнаружения).

Для каждой из обнаруженных уязвимостей запись в базе CVE List содержит краткое описание типа и причин уязвимости, уязвимые версии ПО, оценку критичности уязвимости в соответствии со стандартом CVSS и ссылки на внешние источники с информацией об уязвимости – информационные бюллетени на сайтах производителей ПО или исследовательских организаций. Процесс добавления уязвимости в базу CVE List содержит три этапа:

– обработку - анализ, исследование и процесс приведения уязвимости к формату CVE;

– присвоение - назначение конкретной записи уязвимости идентификатора CVE;

– публикацию - добавление новой записи и публикацию ее на интернет ресурсе CVE, как только он будет официально присвоен.

Веб-интерфейс CVE List предоставляет ограниченные возможности поиска по идентификационному номеру CVE ID, используя ключевые слова (теги). Расширенный поиск уязвимостей в CVE List доступен, через NVD. База данных CVE представляет собой полный список уязвимостей и имеет большое количество ссылок на базы уязвимостей и производителей программных и аппаратных средств, но в ней отсутствуют механизмы фильтрации и описания принадлежности уязвимостей к конкретным продуктам.

БДУ CVE является наиболее полной и систематизированной, поэтому ее используют как основу для указания соответствия записей уязвимостей в других БДУ.

База данных уязвимостей CVE ведется с 1999 г. и на 15 марта 2018г включает 97 888 записей.

Общий вид записи CVE выглядит следующим образом:

– CVE ID, где записывается с указанием года и порядкового номера, например, "CVE-1999-03";

– Description содержит описание уязвимости;

– Reference записываются различного рода ссылки на патчи, рекомендательного рода документы или комментарии разработчика.

Также запись дополнительно может содержать такие поля как:

– статус (Status) — в этом поле может содержаться либо значение Entry (проверенная запись), либо значение Candidate (еще не проверенная уязвимость);

– фаза (Phase) — в этом поле содержится указание стадии развития уязвимости, а также дата наступления этапа. Уязвимость может находится на одной из следующих стадий:

– assignment phase (начальная стадия записи об уязвимости);

– proposal phase (уязвимость предложена на рассмотрение экспертам);

– voting phase (стадия голосования экспертами по данной уязвимости);

– modification phase (запись об уязвимости корректируется по замечаниям экспертов;

– interim decision (рассмотрение уязвимости экспертами завершено, и необходимо принять окончательное решение по данной уязвимости: каждый эксперт должен выбрать одно из предлагаемых решений: "принять" факт наличия уязвимости или "отклонить");

– final decision (фаза окончательного принятия решения по уязвимости, после неё уязвимость считается "принятой" или "отклонённой" с указанием причины);

– голоса (Votes) — поле содержит имена участников голосования, принявших решение о занесении уязвимости в базу;

– комментарии (Voter Comments) — в поле содержится текстовое содержание комментария и имя его автора.

Полностью БД CVE доступна на официальном сайте (cve.mitre.org/data/downloads) в следующих форматах:

– xml;

– html;

– txt;

– csv;

– cvrf (Common Vulnerability Reporting Framework).

Анализ базы данных CVE показал, что она обладает высокой степенью связности с другими источниками описания уязвимостей. Однако недостатком данной базы является отсутствие в описании уязвимостей спецификации программно-аппаратного обеспечения, для которого характерна эта уязвимость.

1.6 Анализ БДУ NVD

National Vulnerability Database национальная база данных уязвимостей США. Разработана отделением компьютерной безопасности (Computer Security Division) национального института стандартов и технологий (National Institute of Standards and Technology) США. Данная база представляет собой хранилище данных уязвимостей, основанное на стандартах протокола автоматизации содержимого безопасности SCAP. База NVD объединила в себе описание уязвимостей, названия ПО с этими уязвимостями и оценки опасности уязвимостей.

Структура записи уязвимости в базе NVD является расширенной формой представления записи в базе CVE, за счет наличия следующих полей:

– конфигурации уязвимых продуктов с учетом зависимостей;

– списка уязвимых продуктов с учетом зависимостей;

– показателей, характеризующих уязвимость в формате CVSS версии 2.0 и 3.0;

– типа доступа для реализации уязвимости;

– информацией о программно-аппаратных продуктах, подверженных уязвимости, в формате CPE;

– информацией по устранению уязвимости (в виде ссылок на соответствующие сайты).

При наличии, для уязвимости также указывается идентификатор в репозитории OVAL. База NVD является общедоступной. Веб-интерфейс NVD предоставляет возможность расширенного поиска в БД: по уникальному идентификатору уязвимости (CVE-ID), по названию ОС, по имени производителя, по имени продукта, по номеру версии ПО, по типу уязвимости и др. Официальный сайт NVD – nvd.nist.gov, веб-интерфейс поиска в БД – web.nvd.nist.gov/view/vuln/search-advanced. Также возможно скачивание базы

NVD (в формате XML) на локальный носитель, адрес страницы загрузки: nvd.nist.gov/download.cfm.

1.7 Общие сведения о базе «Security Focus»

База SecurityFocus разработана в 1990 г. и принадлежит компании Symantec. В данной БД при добавлении уязвимости присваивается уникальный идентификатор BID и определяется ее класс. BID используется многими программными продуктами для ссылок на уязвимости или атак.

Каждая уязвимость, размещенная в сообществе SecurityFocus, имеет следующие характеристики:

- идентификатор CVE;
- дату опубликования и обновления;
- уязвимых продуктах.

Последнее обновление базы данных было выполнено 01.07.2019 г.

Сведения об уязвимостях, размещенных в сообществе SecurityFocus, представлена в виде следующих вкладок:

- «общие сведения» (info);
- «обсуждение» (discussion);
- «информация об эксплойте» (exploit);
- «решение по устранению уязвимости» (solution);
- «ссылки для получения дополнительной информации» (references).

Веб-интерфейс SecurityFocus предоставляет следующие возможности поиска:

- по имени производителя (Vendor);
- по имени продукта (Title);
- по номеру версии ПО (Version);
- по идентификационному номеру CVE (Search by CVE).

1.8 Выявленные проблемы общедоступных БДУ

Для получения полных и достоверных результатов проверка СЗИ должна проводится с использованием нескольких БДУ с последующим объединением результатов. При выполнении автоматизированного поиска уязвимостей необходимо учитывать некоторые недостатки существующих БДУ:

– наличие ошибок и неточностей при описании уязвимостей;

– различные наименования программных продуктов (пакетов) в разных БДУ и в рамках одной БДУ;

– некорректное определение уязвимого ПО (например, уязвимо ядро ОС Linux версии 3.16, а в БДУ содержится запись об уязвимости ОС CentOS);

– не формализованная информация о версиях продуктов;

– поисковый движок некоторых БДУ может давать некорректный результат и множество ложных срабатываний;

– для некоторых БДУ доступно Web-API, однако скорость его работы может оказаться недостаточной для выполнения большого числа запросов;

– результаты, возвращаемые БДУ через Web-API, поисковый движок сайта и с использованием файла-данных БДУ могут значительно отличаться;

– описание уязвимостей в иностранных БДУ сформировано на английском (китайском для CNVD) языке. Для отчетных материалов сертификационных испытаний желательно иметь русскоязычное описание;

– для уязвимостей нет формализованных способов идентификации эксплоитов и патчей.

2 Разработка архитектуры ПК САПУ

2.1 Общие сведения

По результатам анализа нормативно-методической базы сертификационных испытаний и нескольких БДУ были сформированы технические требования для программного средства автоматизации, в соответствии с которыми был разработано веб-приложение, которое реализовано в виде клиент-серверной архитектуры. Схематически клиент-серверная архитектура представлена на рисунке 1.

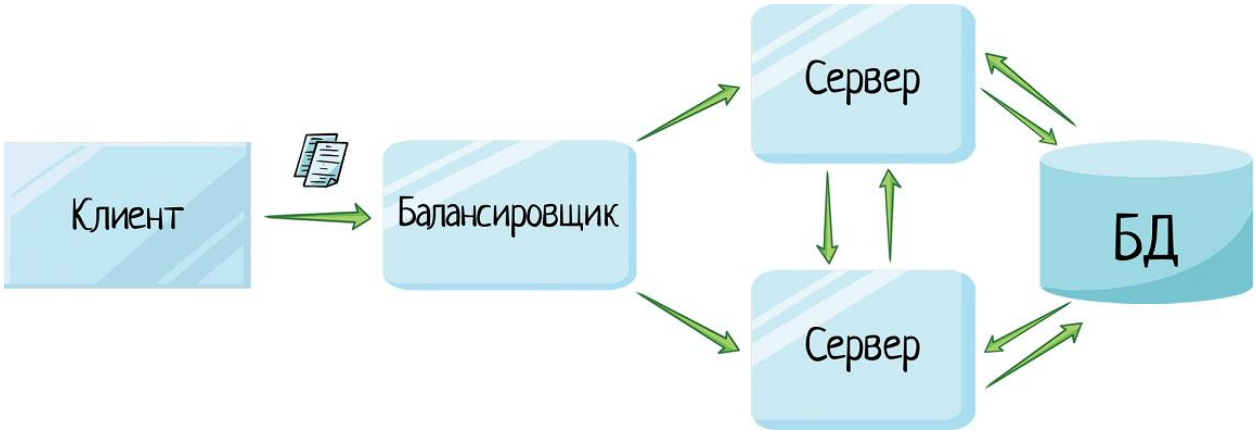


Рисунок 1 – Клиент-серверная архитектура

Для реализации серверной части используется архитектурный стиль REST, который имеет следующие принципы:

- единый интерфейс;
- разграничение клиента и сервера;
- нет сохранения состояния;
- кэширование всегда разрешено;
- многоуровневая система;
- код предоставляется по запросу.

2.2 Состав системы

В состав системы входит Web-сервер на базе свободно распространяемого дистрибутива линукс Debian.

В систему должны быть установлены следующие пакеты:

– Python, версии 3.7 (или выше);

– Nginx, версии 1.18.0;

– NodeJS, версии 12.14.0;

– PostgreSQL, версии 9.3;

– Redis, версии 6.0.0 (или выше).

2.3 Выбранные технологии

Был описан выбор технологий, используемых для реализации веб-приложения.

В качестве языка программирования для реализации серверной части был выбран язык Python3, так как БДУ содержат в себе большие объемы данных, которые приложение должно обработать. Python3 отлично подходит для работы с большими данными, так как он имеет большое количество библиотек для работы с ними (pandas и тд.).

Как фреймворк для реализации серверной части был выбран falcon. Он имеет следующие преимущества:

– это невероятно быстрая и минималистичная платформа веб-API Python для создания надежных серверных приложений и микросервисов.

– Falcon фокусируется на стабильности, корректности и производительности в масштабе;

– поддерживает архитектурный стиль REST. Классы ресурсов реализуют обработчики методов HTTP, которые выполняют запросы и переходы между состояниями.

Nginx - мощный инструмент для развертывания веб-сервера. В отличие от обычного веб-сервера, Nginx не создаёт один поток под каждый запрос, а разделяет его на меньшие однотипные структуры, называемые рабочими соединениями. Каждое такое соединение обрабатывается отдельным рабочим процессом, а после выполнения они сливаются в единый блок, возвращающий результат в основной процесс обработки данных. Одно рабочее соединение может обрабатывать до 1024 запросов одного вида одновременно. Конфигурация NGINX представлена ниже.

```
map $http_upgrade $connection_upgrade {
     default upgrade;
     '' close;
}
upstream frontend {
  server 127.0.0.1:8080;
}
upstream backend {
  server 127.0.0.1:8000;
}
server {
  listen 80;
  server_name bdu.local;
  client_max_body_size 100M;

  location /api/ {
    proxy_pass http://backend/;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_set_header X-Real-IP $remote_addr;
  }

  location /storage/ {
    alias /Users/dmitry/Projects/defense-analityc/storage/;
  }
```

```
  location / {
    proxy_pass http://frontend/;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Host 127.0.0.1;
  }
}
```

PostgreSQL – это продвинутая объектно-реляционная система с открытым исходным кодом, которая применяет язык SQL. PostgreqSQL имеет следующие достоинства:

– надежность - нет абсолютно никаких проблем с безотказной работой или целостностью данных;

– гибкость - широкий диапазон поддерживаемых типов данных дает разработчику огромную гибкость с точки зрения того, какие данные он может использовать для хранения;

– внутренние функции – дают возможность разработчику использовать меньше внешних зависимостей для выполнения общих бизнес-задач.

Redis - это размещаемое в памяти хранилище ключ-значение, в ПК САПУ оно используется для хранения сессии пользователя.

Redis Queue – это являющаяся частью Redis технология, которая отвечает за выполнение задач в очереди, чтобы пользователю не ожидать долгого ответа с сервера.

VueJS – это javascript фреймворк для реализации клиентской части. Он был выбран, так как имеет низкий порог вхождения, а также очень имеет расширяемую структуру и большие количество готовых модулей.

Vuetify – это CSS фреймворк, работающий на основе VueJS, который быстро и легко позволяет создать отзывчивые интерфейсы и имеет большое количество готовых UI блоков, которые дают пользователю лучший опыт.

Была спроектирована архитектура приложения, представленная на рисунке 2.

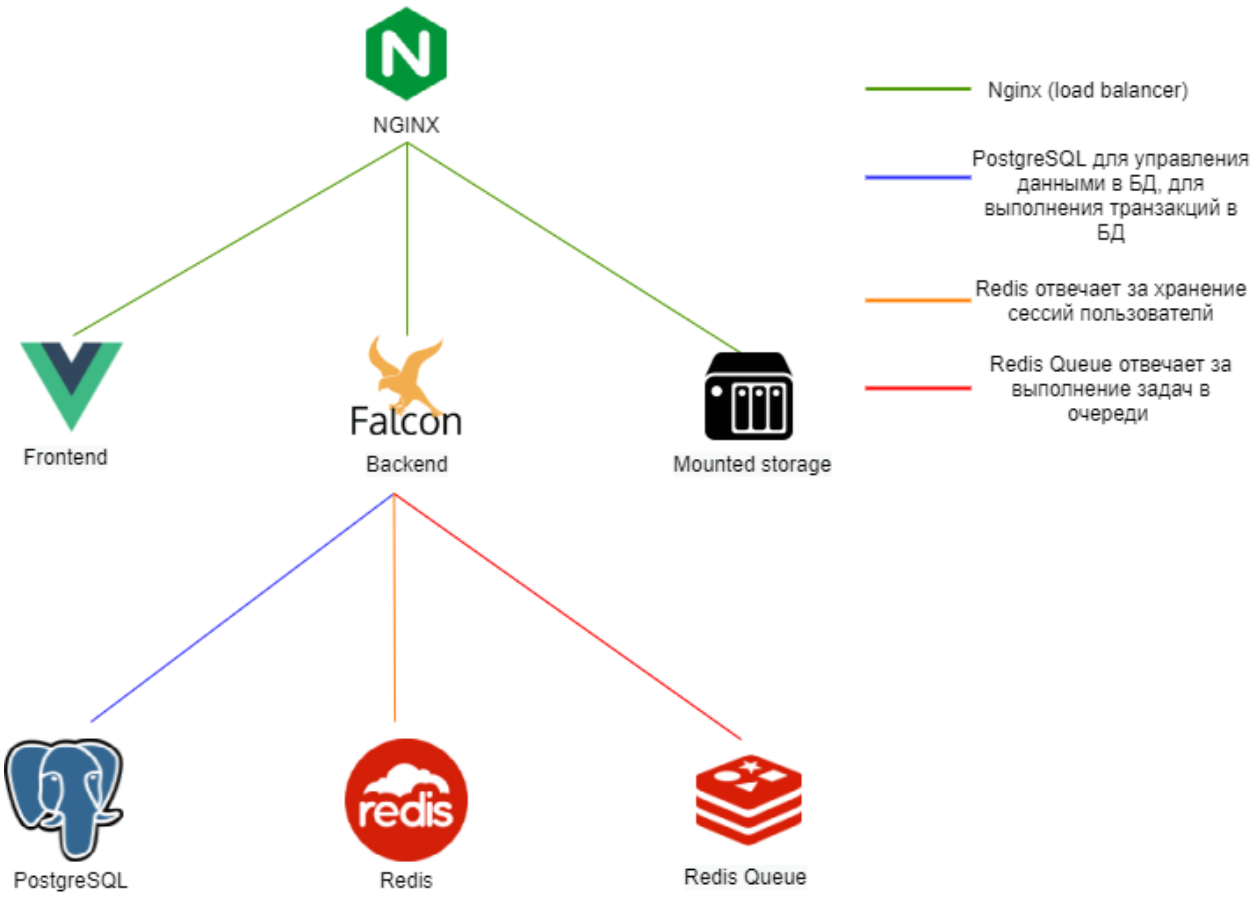
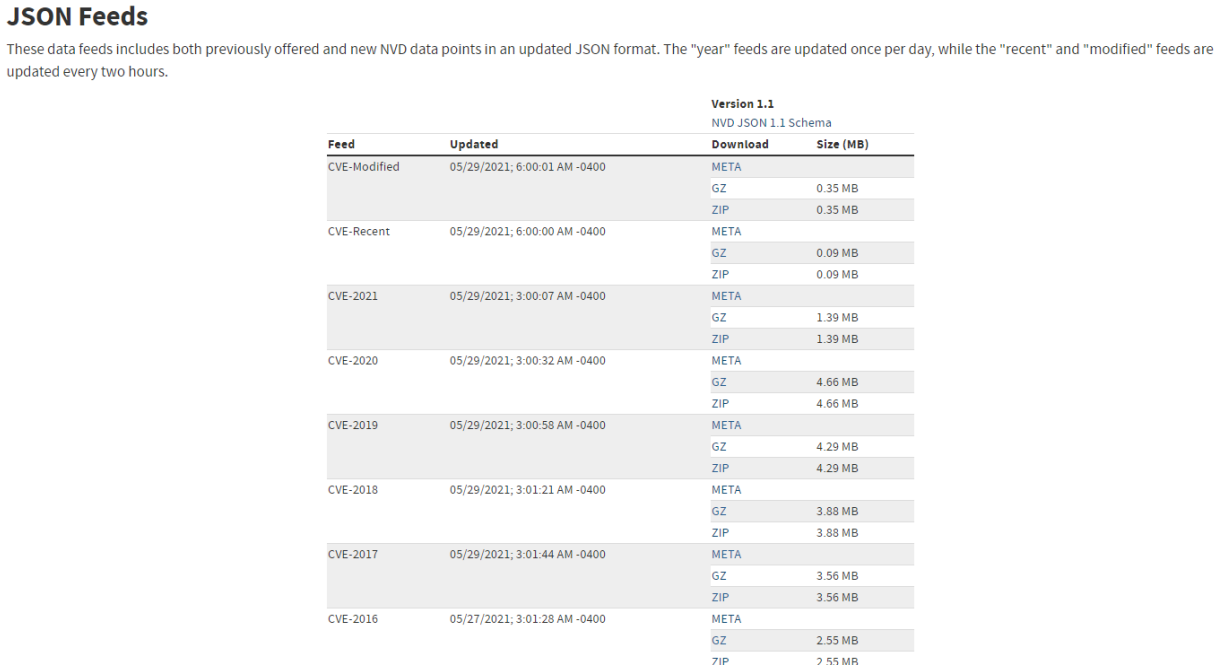


Рисунок 2 – Архитектура приложения

3 Разработка серверной части ПК САПУ

3.1 Получение данных из NVD

На сайте БДУ NVD был найден раздел в котором, хранятся записи БДУ в формате JSON. Записи сгруппированы по годам, начиная с 2002 года по 2021 год. Раздел сайта БДУ NVD, представлен на рисунке 3.



JSON Feeds

These data feeds includes both previously offered and new NVD data points in an updated JSON format. The "year" feeds are updated once per day, while the "recent" and "modified" feeds are updated every two hours.

		Version 1.1 NVD JSON 1.1 Schema	
Feed	**Updated**	**Download**	**Size (MB)**
CVE-Modified	05/29/2021; 6:00:01 AM -0400	META	
		GZ	0.35 MB
		ZIP	0.35 MB
CVE-Recent	05/29/2021; 6:00:00 AM -0400	META	
		GZ	0.09 MB
		ZIP	0.09 MB
CVE-2021	05/29/2021; 3:00:07 AM -0400	META	
		GZ	1.39 MB
		ZIP	1.39 MB
CVE-2020	05/29/2021; 3:00:32 AM -0400	META	
		GZ	4.66 MB
		ZIP	4.66 MB
CVE-2019	05/29/2021; 3:00:58 AM -0400	META	
		GZ	4.29 MB
		ZIP	4.29 MB
CVE-2018	05/29/2021; 3:01:21 AM -0400	META	
		GZ	3.88 MB
		ZIP	3.88 MB
CVE-2017	05/29/2021; 3:01:44 AM -0400	META	
		GZ	3.56 MB
		ZIP	3.56 MB
CVE-2016	05/27/2021; 3:01:28 AM -0400	META	
		GZ	2.55 MB
		ZIP	2.55 MB

Рисунок 3 – Раздел сайта БДУ NVD с уязвимостями

Затем с помощью инструментов разработчика была найдена ссылка по которой можно скачать json файл с уязвимостями. Результат представлен на рисунке 4.

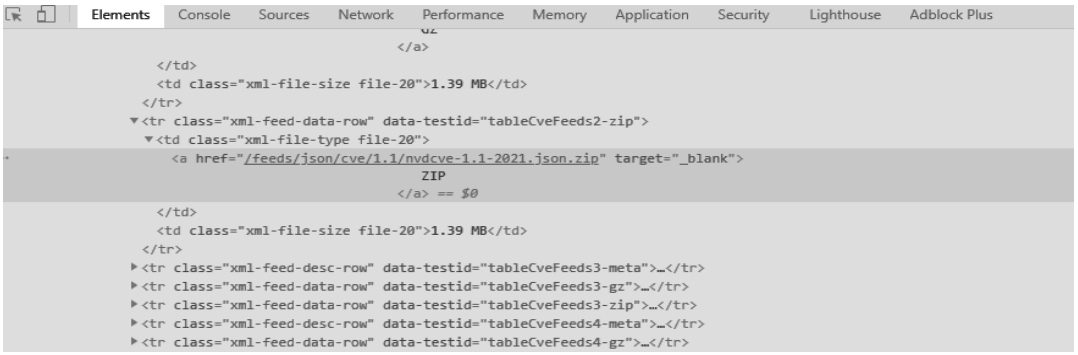


Рисунок 4 – Ссылка на скачивание json файла с уязвимостями

Был проведен анализ полученных json файлов, в результате чего, было выявлено, что каждая уязвимость в БДУ NVD будет иметь следующие необходимые поля:

- название;
- описание;
- дата публикации;
- конфигурации;
- CVSSv2;
- CVSSv3 (если есть).

Каждая из записей в NVD имеет список конфигураций, при которых реализуется уязвимость, что позволяет выполнять более точный поиск уязвимостей под конкретную конфигурацию.

Далее был реализован алгоритм скачивания и обновления БДУ NVD.

Алгоритм скачивает архивы с 2002 по 2021 года по найденной ссылке, затем сохраняет их, распаковывает и удаляет архивы. Фрагмент алгоритма, который отвечает за скачивание архивов, сохранение, распаковывание и удаление, представлен рисунке 5.

```
for year in range(START_YEAR, dt.now().year + 1):
    archive_path = f'{NVD_STORAGE}{year}.zip'
    urlretrieve(f'{NVD_URL}{year}.json.zip', archive_path)
    json_path = extract_archive(archive_path, NVD_STORAGE)
    filestorage.cleaner(archive_path)
```

Рисунок 5 – Фрагмент алгоритма, который отвечает за скачивание архивов, сохранение, распаковывание и удаление

Была написана функция, которая разархивирует архив в переданную в параметры директорию.

Код функции представлен на рисунке 6.

```
def extract_archive(archive_path, direction):
    with ZipFile(archive_path) as zfile:
        list_of_files = zfile.namelist()
        for filename in list_of_files:
            if filename.endswith('.json'):
                zfile.extract(filename, direction)
                return f'{direction}{filename}'
```

Рисунок 6 – Код функции

Полученный в итоге список json файлов обрабатывается и сохраняется в заданном формате в базу данных. Тем самым сохраняя все записи об уязвимостях на сервере, что позволяет быстро забирать и управлять этими записями.

Код алгоритма представлен в приложении А.

3.2 Получение данных из CNVD

На сайте БДУ CNVD был найден раздел в котором, хранятся записи БДУ в формате xml. Записи сгруппированы по неделям, начиная с 2014 года по 2021 год. Раздел сайта БДУ CNVD, представлен на рисунке 8.

Рисунок 8 – Раздел сайта БДУ CNVD с уязвимостями

БДУ CNVD не предоставляет возможности скачивания файлов с уязвимостями через ссылку, поэтому скачивание было выполнено в ручном режиме.

Был проведен анализ полученных xml файлов, в результате чего, было выявлено, что каждая уязвимость в БДУ CNVD будет иметь следующие необходимые поля:

- название;
- описание;
- дата публикации;
- конфигурации;

Каждая из записей в CNVD имеет список конфигураций, при которых реализуется уязвимость, что позволяет выполнять более точный поиск уязвимостей под конкретную конфигурацию.

Полученный в итоге список xml файлов обрабатывается и сохраняется в заданном формате в базу данных. Тем самым сохраняя все записи об уязвимостях на сервере, что позволяет быстро забирать и управлять этими записями.

Код алгоритма представлен в приложении Б.

3.3 Получение данных из ФСТЭК

На сайте БДУ ФСТЭК был найден раздел в котором, хранятся записи БДУ в формате xml. Файл был скачан по найденной ссылке.

Был проведен анализ полученного xml файла, в результате чего, было выявлено, что каждая уязвимость в БДУ ФСТЭК будет иметь следующие необходимые поля:

- название;
- описание;
- дата публикации;

– конфигурации;

– CVSSv2.

Каждая из записей в ФСТЭК имеет список конфигураций, при которых реализуется уязвимость, что позволяет выполнять более точный поиск уязвимостей под конкретную конфигурацию.

Полученный в итоге список xml файлов обрабатывается и сохраняется в заданном формате в базу данных. Тем самым сохраняя все записи об уязвимостях на сервере, что позволяет быстро забирать и управлять этими записями.

Код алгоритма представлен в приложении В.

3.4 Проектирование API

API приложения было спроектировано в соответствии с шаблоном Model-View-Controller.

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

– модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние;

– представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели;

– контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Было принято решение об использовании в проекте ORM (Object-Relational Mapping) – Pewee. Для того чтобы иметь богатую объектно-ориентированную бизнес-модель и при этом иметь возможность хранить ее и быстро писать эффективные запросы к реляционной базе данных.

Был написан модуль, который отвечает за подключение к базе данных, используя ORM. Результат представлен на рисунке 9.

```
import peewee
import peewee_async

from config import config


db = peewee_async.PostgresqlDatabase(
    config['db'].pop('db'),
    **config['db'],
    autorollback=True,
    autoconnect=True
)


class BaseModel(peewee.Model):
    class Meta:
        database = db


db.Model = BaseModel
```

Рисунок 9 - Модуль, который отвечает за подключение к базе данных, используя ORM

Был создан конфигурационный файл, в котором хранятся данные о проекте, базе данных, соль для пароля и сессии пользователя, данные для подключения к Redis, а также мод в котором в данным момент запущено приложение (в режиме разработки или в режиме эксплуатации).

Все значения хранятся в переменных окружения и задаются в файле с расширением .env. Переменные окружения более безопасны, чем файлы с открытым текстом, потому что они ограничивают доступ к значениям этих переменных, то есть они доступны только на том, сервере, где развернуто ПК САПУ.

Конфигурационный файл, представлен на рисунке 10.

```
import os
from environs import Env


env = Env()
env.read_env()


config = {
    'project': {
        'project': env("PROJECT_NAME"),
        'version': env("PROJECT_VERSION"),
        'mode': env("APP_MODE")
    },
    'db': {
        'host': env("DB_HOST"),
        'db': env("DB_NAME"),
        'user': env("DB_USER"),
        'password': env("DB_PASSWORD"),
        'port': env("DB_PORT")
    },
    'secure': {
        'salt_password': env("SALT_PASSWORD"),
        'salt_session': env("SALT_SESSION")
    }
}

REDIS_HOST = env("REDIS_HOST")
REDIS_PORT = env("REDIS_PORT")

NVD_URL = env("NVD_URL")


app_mode = env('APP_MODE')
is_production = app_mode == 'production'
```

Рисунок 10 - Конфигурационный файл

3.5 Модуль работы с авторизацией

Была написана модель для работы с авторизацией, которая нужна для аутентификации и предоставления прав пользователю на пользование функциями ПК САПУ.

Была написана модель пользователя. Результат представлен на рисунке 11.

```
class User(db.Model):

    login = CharField(unique=True)
    email = CharField(null=True, unique=True)
    password = CharField()
    created_at = DateTimeField(default=dt.now)
    is_active = BooleanField(default=True)
```

Рисунок 11 – Модель пользователя

Также в модели пользователя были дописаны две функции:

– первая функция выставляет пароль пользователю, используя алгоритм хэширования sha256;

– вторая проверяет пароль на правильность, который приходит с клиента. Результат представлен на рисунке 12.

```
def set_password(self, password):
    self.password = sha256((password+config['secure']['salt_password']).encode()).hexdigest()

def check_password(self, value):
    return (
        self.password == sha256((value+config['secure']['salt_password']).encode()).hexdigest()
    )
```

Рисунок 12 – Функции для работы с паролем пользователя

Была написана миграция, которая создает таблицу в базе данных в соответствии с моделью. Результат представлен на рисунке 13.

```
from datetime import datetime as dt

from peewee import Model, BooleanField, CharField, DateTimeField


def migrate(migrator, database, fake=False, **kwargs):

    @migrator.create_model
    class User(Model):

        login = CharField(unique=True)
        email = CharField(null=True, unique=True)
        password = CharField()
        created_at = DateTimeField(default=dt.now)
        is_active = BooleanField(default=True)


def rollback(migrator, database, fake=False, **kwargs):
    migrator.remove_model('user')
```

Рисунок 13 – Миграция для модели пользователя

Были написаны схемы для пользователя, двух типов Public и Manage. Public схема – это схема, которая отражает поля, которые получит клиент. Manage схема – это схема, по которой проверяется тело запроса, которое пришло с клиента. Результат представлен на рисунке 14.

```
from marshmallow import Schema, fields


class UserManageSchema(Schema):

    login = fields.String(required=True)
    password = fields.String(required=True)


class UserPublicSchema(Schema):

    id = fields.Integer()
    login = fields.String()
    email = fields.String()
    created_at = fields.String()
    is_active = fields.Boolean()
```

Рисунок 14 – Схемы для пользователя

Были написан котроллер для аутентификации пользователя, который получает пользователя по пришедшему с клиента логину.

При выполнении данного запроса, сервер может вернуть следующие ошибки:

– пользователь не найден, если пользователь с пришедшим с запроса логином не найден;

– отказ в доступе, если пользователь с пришедшим с запроса логином заблокирован;

– неверный пароль, если пароль, пришедший в запросе, не совпадает с паролем, который хранится в базе данных.

Алгоритм работы контроллера представлен на рисунке 15.

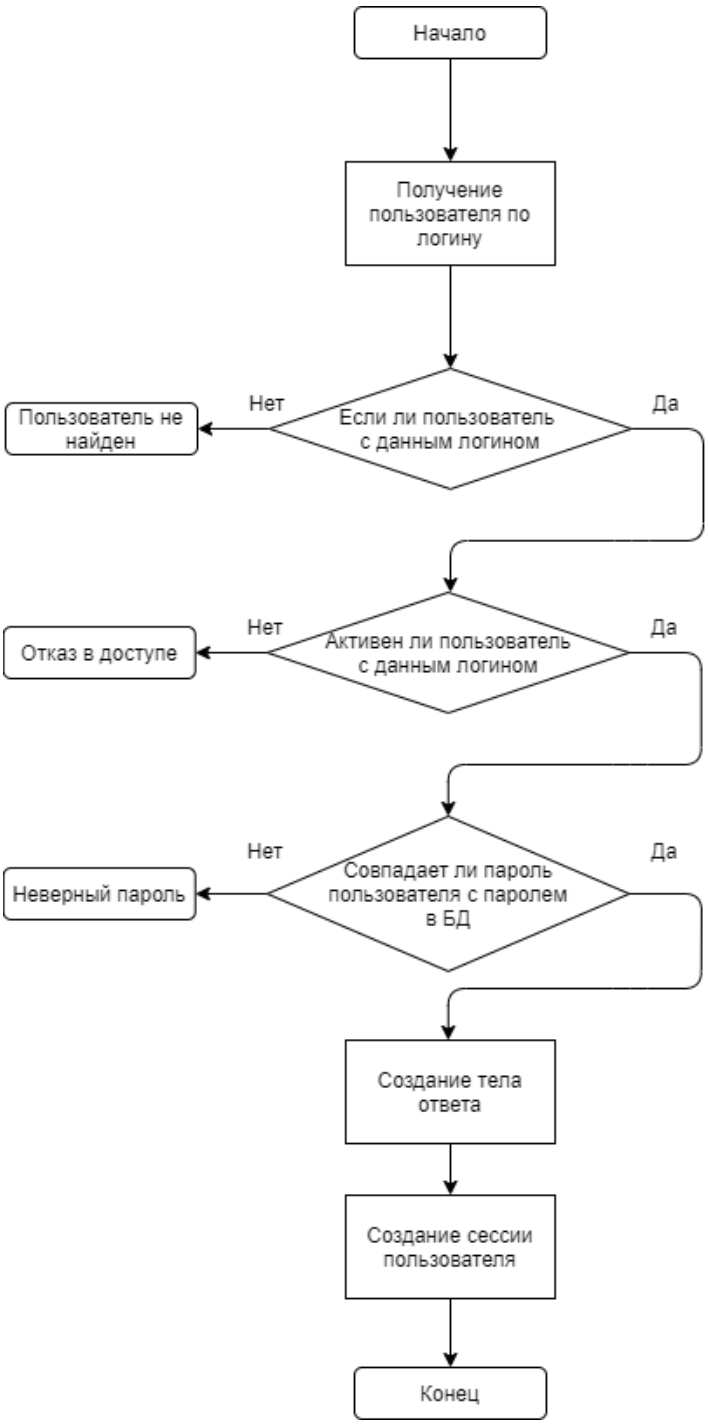


Рисунок 15 – Алгоритм работы контроллера

Был написан контроллер для отдачи клиенту, пользователя, который в данный момент авторизован.

Был написан контроллер для выхода пользователя из приложения, который удаляет сессию пользователя, удаляет файлы cookie и отдает ответ о успешном выходе клиенту.

3.6 Модуль работы с уязвимостями

Так как ПК САПУ сводит несколько БДУ в одно унифицированную структуру. Для реализации данной структуры была спроектирована логическая модель уязвимости. Она представлена рисунке 16.

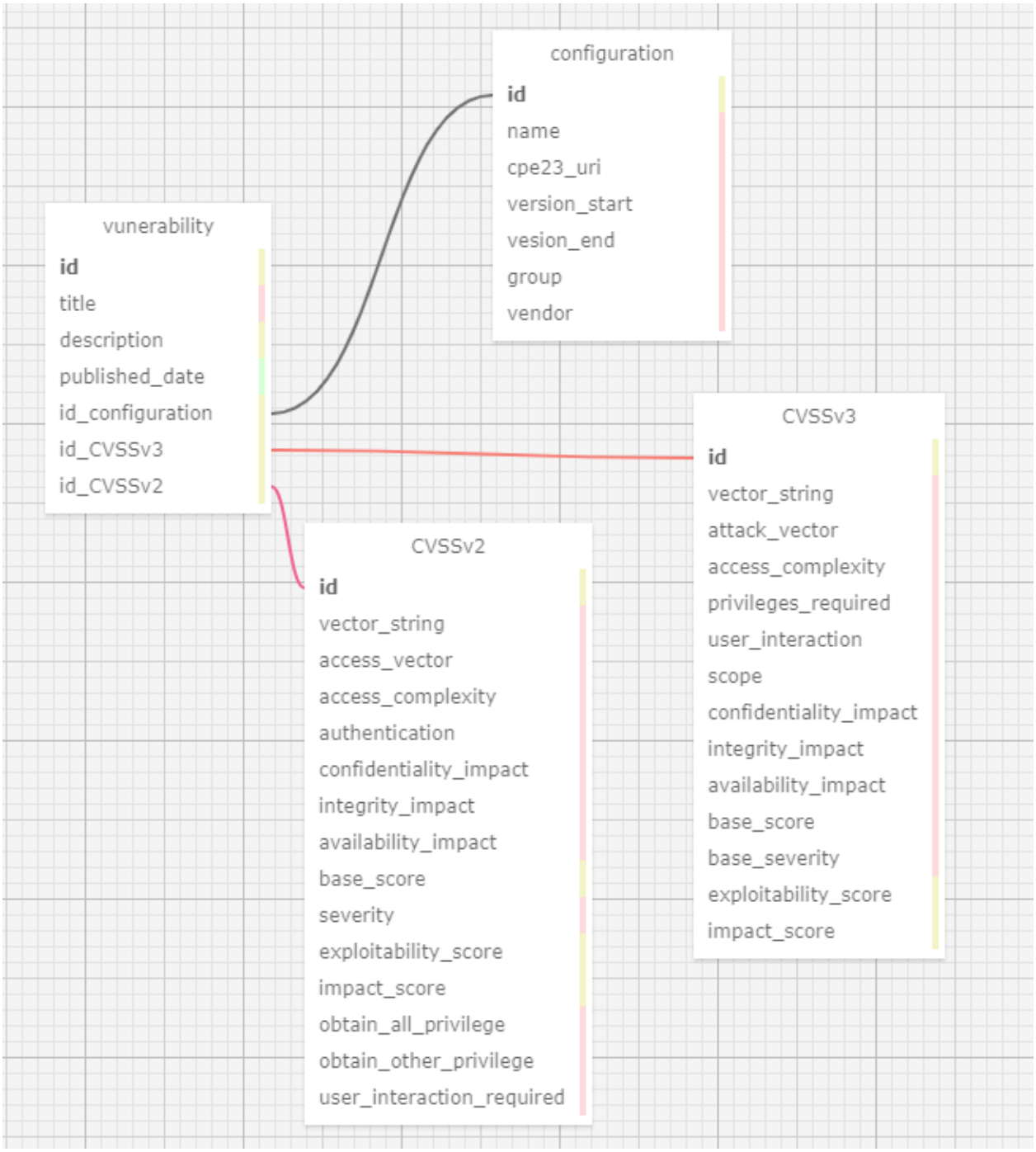


Рисунок 16 - Логическая модель уязвимости

Была написана модель уязвимости. Результат представлен на рисунке 17.

```
class Vulnerability(db.Model):

    title = CharField()
    description = TextField(null=True)
    published_date = DateTimeField(null=True)

    class Meta:
        table_name = 'vulnerability'
```

Рисунок 17 – Модель уязвимости

Была написана модель конфигурации. Результат представлен на рисунке 18.

```
class Configuration(db.Model):

    vulnerability = ForeignKeyField(Vulnerability, backref='configurations')
    name = CharField(null=True)
    cpe23_uri = CharField()
    version_start = CharField(null=True)
    version_end = CharField(null=True)
    group = CharField(null=True)
    vendor = CharField(null=True)

    class Meta:
        table_name = 'cofiguration'
```

Рисунок 18 – Модель конфигурации

Были написаны модели CVSSv2 и CVSSv3. Результаты представлены на рисунках 19 и 20.

```
class CVSSv2(db.Model):

    vulnerability = ForeignKeyField(Vulnerability, backref='cvss_v2')
    vector_string = CharField()
    access_vector = CharField(null=True)
    access_complexity = CharField(null=True)
    authentication = CharField(null=True)
    confidentiality_impact = CharField(null=True)
    integrity_impact = CharField(null=True)
    availability_impact = CharField(null=True)
    base_score = FloatField(null=True)
    severity = CharField(null=True)
    exploitability_score = FloatField(null=True)
    impact_score = FloatField(null=True)
    obtain_all_privilege = BooleanField(default=False)
    obtain_other_privilege = BooleanField(default=False)
    user_interaction_required = BooleanField(default=False)
```

Рисунок 19 – Модель CVSSv2

```
class CVSSv3(db.Model):

    vulnerability = ForeignKeyField(Vulnerability, backref='cvss_v3')
    vector_string = CharField()
    attack_vector = CharField(null=True)
    access_complexity = CharField(null=True)
    privileges_required = CharField(null=True)
    user_interaction = CharField(null=True)
    scope = CharField(null=True)
    confidentiality_impact = CharField(null=True)
    integrity_impact = CharField(null=True)
    availability_impact = CharField(null=True)
    base_score = FloatField(null=True)
    base_severity = CharField(null=True)
    exploitability_score = FloatField(null=True)
    impact_score = FloatField(null=True)
```

Рисунок 20 – Модель CVSSv3

Были написаны миграции для моделей уязвимости, конфигурации, CVSSv2 и CVSSv3 к базе данных, которые создают таблицы в базе данных в соответствии с моделью. Пример миграции представлен на рисунке 21.

```
from peewee import Model, BooleanField, CharField, ForeignKeyField


def migrate(migrator, database, fake=False, **kwargs):

    class Vulnerability(Model):
        class Meta:
            table_name = 'vulnerability'
        pass

    @migrator.create_model
    class Configuration(Model):

        vulnerability = ForeignKeyField(
            Vulnerability, backref="configurations")
        cpe23_uri = CharField()
        version_start = CharField(null=True)
        version_end = CharField(null=True)

        class Meta:
            table_name = 'cofiguration'


def rollback(migrator, database, fake=False, **kwargs):
    migrator.remove_model('cofiguration')
```

Рисунок 21 – Пример миграции модели конфигурации

Были написаны схемы для тех же моделей, которые отвечают за то, какие поля из модели получит клиент в ответе. Пример представлен на рисунке 22.

```
class VulnerabilityPublicSchema(Schema):

    id = fields.Integer()
    title = fields.String()
    description = fields.String()
    configurations = fields.Nested(ConfigurationNestedSchema, many=True)
    cvssv2 = fields.Nested(CVSSv2NestedSchema)
    cvssv3 = fields.Nested(CVSSv3NestedSchema)
    published_date = fields.DateTime()


class VulnerabilitiesPublicSchema(Schema):

    vulnerabilities = fields.Nested(VulnerabilityPublicSchema, many=True)
    amount = fields.Integer()
```

Рисунок 22 – Пример схемы

Был написан контроллер для поиска уязвимостей в БД, который принимает на вход два обязательных параметра:

– название ПО;

– версия.

А также два опциональных поля:

– вендор;

– способ эксплуатации (локально или удаленно).

Была написана функция для получения уязвимостей по запрашиваемым параметрам для каждой БДУ, которая уже сохранена в нашей базе данных. Функция получения уязвимостей из БДУ NVD, представлена на рисунке 23. Функция получения уязвимостей из БДУ CNVD, представлена на рисунке 24. Функция получения уязвимостей из БДУ ФСТЭК, представлена на рисунке 25.

```
async def get_nvd_vulnerabilities(self, name, start, vendor=None, attack_vector=None):

    nvd_name = ('' + '_'.join(name.split(' '))).lower()

    configurations = (
        NVDConfiguration
        .select()
        .join(NVDVulnerability, on=(NVDVulnerability.id == NVDConfiguration.nvd))
        .where(
            (NVDConfiguration.name == nvd_name) &
            (
                (NVDConfiguration.version_start.contains(start)) |
                (NVDConfiguration.version_start == 'any') |
                (NVDConfiguration.version_start == '-')
            ) &
            (NVDVulnerability.source == 'nvd')
        )
    )

    if vendor:
        vendor = ('' + '_'.join(vendor.split(' '))).lower()

        configurations = configurations.where(NVDConfiguration.vendor == vendor)

    configurations_ids = [conf.id for conf in configurations]

    nvd_vulnerabilities = (
        NVDVulnerability
        .select()
        .join(NVDConfiguration, on=(NVDConfiguration.nvd == NVDVulnerability.id))
        .where(
            (NVDConfiguration.id << configurations_ids) &
            (NVDVulnerability.source == 'nvd')
        )
        .group_by(NVDVulnerability.id)
        .order_by(NVDVulnerability.title.asc())
    )

    if attack_vector:

        if attack_vector not in ['local', 'network']:
            raise falcon.HTTPInvalidParam(msg='Invalid attack vector', param_name='attack_vector')

        nvd_vulnerabilities_v2 = (
            nvd_vulnerabilities
            .select()
            .join(CVSSv2, on=(CVSSv2.nvd == NVDVulnerability.id))
            .where(
                (CVSSv2.access_vector == attack_vector)
            )
        )

        nvd_vulnerabilities_v3 = (
            nvd_vulnerabilities
            .select()
            .join(CVSSv3, on=(CVSSv3.nvd == NVDVulnerability.id))
            .where(
                (CVSSv3.attack_vector == attack_vector)
            )
        )

        nvd_vulnerabilities = nvd_vulnerabilities_v2.union(nvd_vulnerabilities_v3).group_by(NVDVulnerability.id)

    return nvd_vulnerabilities
```

Рисунок 23 – Функция получения уязвимостей из NVD

```
async def get_cnvd_vulnerabilities(self, name, start):

    cnvd_configurations = (
        NVDConfiguration
        .select()
        .join(NVDVulnerability, on=(NVDVulnerability.id == NVDConfiguration.nvd))
        .where(
            (
                (NVDConfiguration.cpe23_uri.contains(name)) |
                (NVDConfiguration.cpe23_uri.contains(start))
            ) &
            (NVDVulnerability.source == 'cnvd')
        )
    )

    cnvd_vulnerabilities = (
        NVDVulnerability
        .select()
        .join(NVDConfiguration, on=(NVDConfiguration.nvd == NVDVulnerability.id))
        .where(
            (
                (NVDConfiguration.id << cnvd_configurations) &
                (NVDVulnerability.description.contains(name)) &
                (NVDVulnerability.source == 'cnvd')
            ) |
            (
                (NVDVulnerability.description.contains(name)) &
                (NVDVulnerability.description.contains(start)) &
                (NVDVulnerability.source == 'cnvd')
            )
        )
        .group_by(NVDVulnerability.id)
        .order_by(NVDVulnerability.title.asc())
    )

    return cnvd_vulnerabilities
```

Рисунок 24 – Функция получения уязвимостей из CNVD

```
async def get_bdu_vulnerabilities(self, name, start, vendor=None):

    bdu_configurations = (
        NVDConfiguration
        .select()
        .join(NVDVulnerability, on=(NVDVulnerability.id == NVDConfiguration.nvd))
        .where(
            (
                (NVDConfiguration.cpe23_uri.contains(name)) |
                (NVDConfiguration.cpe23_uri.contains(start))
            ) &
            (NVDVulnerability.source == 'bdu')
        )
    )

    if vendor:
        vendor = ('' + '_'.join(vendor.split(' '))).lower()

        bdu_configurations = bdu_configurations.where(NVDConfiguration.vendor == vendor)

    bdu_vulnerabilities = (
        NVDVulnerability
        .select()
        .join(NVDConfiguration, on=(NVDConfiguration.nvd == NVDVulnerability.id))
        .where(
            (
                (NVDConfiguration.id << bdu_configurations) &
                (NVDVulnerability.description.contains(name)) &
                (NVDVulnerability.source == 'bdu')
            ) |
            (
                (NVDVulnerability.description.contains(name)) &
                (NVDVulnerability.source == 'bdu')
            )
        )
        .group_by(NVDVulnerability.id)
        .order_by(NVDVulnerability.title.asc())
    )

    return bdu_vulnerabilities
```

Рисунок 25 – Функция получения уязвимостей из ФСТЭК

Полученные записи были сгруппированы в один массив, где хранятся все уязвимости по всем трем БДУ. Итоговый массив отдается на клиент, где этот массив будет отображаться пользователю в соответствии с введенными им параметрами.

Был написана функция, которая формирует файл формата xls, в котором представлены все полученные записи в соответствии с параметрами, полученными от клиента. Код функции представлен на рисунке 26.

В файл включены следующие поля:

– ID;
– название;
– описание;
– дата создания;
– источник;
– CVSSv2.

```
def make_file_vulnerabilities(direction, vulnerabilities):

    columns = ['ID', 'Название', 'Описание', 'Дата создания', 'Источник', 'CVSSv2']

    rows = [
        [
            f'DAV_{cve.id}',
            cve.title,
            cve.description,
            str(cve.published_date),
            cve.source,
            cve.cvss_v2[0].vector_string if cve.cvss_v2.exists() else 'None'
        ]
        for cve in vulnerabilities
    ]

    if rows.__len__() == 0:
        return None
    else:
        vul_df = pd.DataFrame(rows, columns=columns)

    file_path = direction + '/result_' + \
        dt.now().strftime('%d.%m.%Y_%H:%M:%S') + '.xls'

    if not os.path.exists(direction):
        os.makedirs(direction)

    with pd.ExcelWriter(file_path) as writer:
        vul_df.to_excel(writer, 'Уязвимости')

    return file_path
```

Рисунок 26 – Функция для формирования xls файла

Также был реализован функционал, позволяющий автоматизировать актуализацию данных об уязвимостях для всех трех БДУ.

Так как NVD предоставляет возможность скачивания архивов с уязвимостями по ссылке, для него был написан контроллер, который может принимать на вход опциональный параметр - год с которого нужно начать обновление записей из данной БДУ, либо обновить все записи, начиная с 2002 года.

Результат представлен на рисунке 27.

```
class UpdateNVDController(Controller):

    @falcon.before(auth_required)
    async def on_get(self, req, resp):

        year = req.params.get('year', 2002)

        if year < 2002:
            raise falcon.HTTPBadRequest(description='Incorrect year param')

        append_update_nvd_to_queue(year)

        resp.text = json.dumps({'success': True})
```

Рисунок 27 – Контроллер для автоматизированного обновления БДУ NVD

БДУ CNVD и ФСТЭК, не предоставляют возможность скачивания архивов с уязвимостями по ссылке, поэтом для этих БДУ был написан отдельный контроллер, который принимает на вход zip архив с xml файлом в случае ФСТЭК и несколькими xml файлами в случае CNVD.

Код контроллера представлен в приложении Г.

Обновление всех трех БДУ выполняется с использованием Redis Queue, которая позволяет ставить процессы в очередь, тем самым не заставляя клиента ожидать полного завершения процесса обновления одной или нескольких БДУ. Код по работе с Redis Queue, представлен на рисунке 28.

```
from rq import Queue

from libs.redis import UpdateQueue
from libs.logger import logger
from cron_nvd_update import main as nvd_update
from cron_fstek_update import main as fstek_update
from cron_cnvd_update import main as cnvd_update
from cron_cwe_update import main as cwe_update


def append_update_nvd_to_queue(year):
    queue = Queue(connection=UpdateQueue)
    queue.enqueue(nvd_update, year, job_timeout=30000)

    logger.info('Update job in queue')


def append_update_archieve_to_queue(direction, database):
    queue = Queue(connection=UpdateQueue)

    if database == 'fstek':
        queue.enqueue(fstek_update, direction, job_timeout=30000)
    elif database == 'cnvd':
        queue.enqueue(cnvd_update, direction, job_timeout=80000)

    logger.info('Update job in queue')
```

Рисунок 28 – Код по работе с Redis Queue

В результате выполнения данного пункта была реализована серверная часть ПК САПУ.

4 Проектирование пользовательского интерфейса ПК САПУ

Дизайн пользовательского интерфейса (UI) – это интерфейс, который фокусируется на предвидении того, что может потребоваться пользователям, и обеспечении того, чтобы интерфейс содержал элементы, к которым легко получить доступ, понять и использовать для облегчения этих действий. Пользовательский интерфейс объединяет концепции дизайна взаимодействия, визуального дизайна и информационной архитектуры.

Был спроектирован пользовательский интерфейс экрана авторизации в приложении.

Результат представлен на рисунке 29.

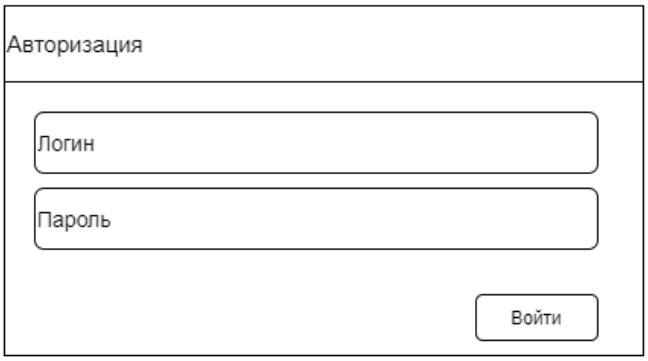


Рисунок 29 - Пользовательский интерфейс экрана авторизации

Был спроектирован пользовательский интерфейс экрана поиска уязвимостей в режиме, когда параметры поиска задаются внутри приложения. Результат представлен на рисунке 30.

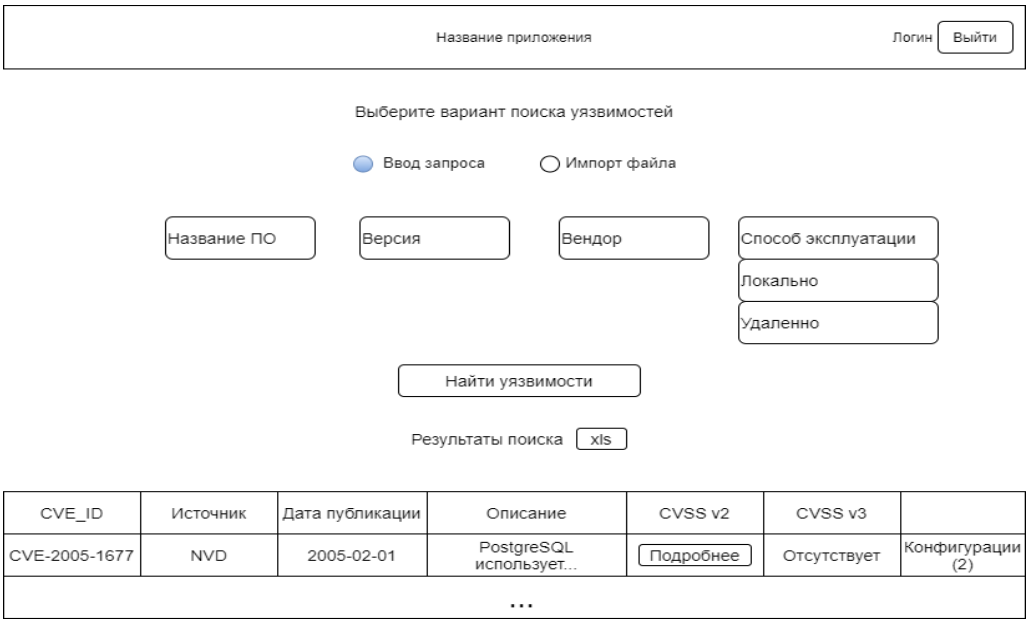


Рисунок 30 - Пользовательский интерфейс экрана поиска уязвимостей в режиме, когда параметры поиска задаются внутри приложения

Был спроектирован пользовательский интерфейс экрана поиска уязвимостей в режиме, когда параметры поиска задаются через файл. Результат представлен на рисунке 31.

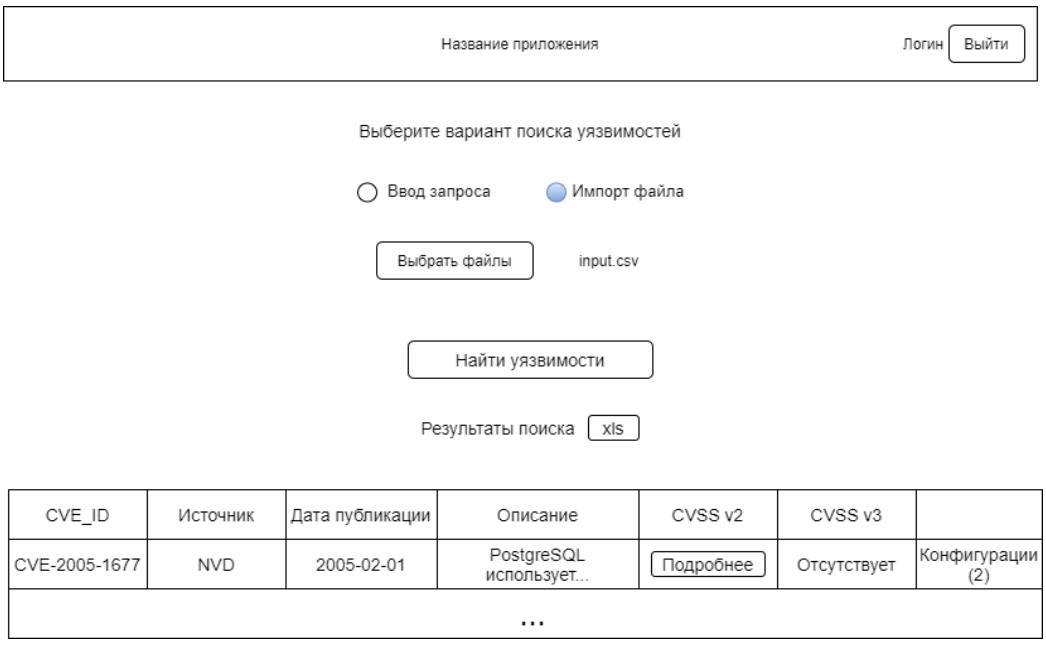


Рисунок 31 - Пользовательский интерфейс экрана поиска уязвимостей в режиме, когда параметры поиска задаются через файл

Был спроектирован пользовательский интерфейс экрана профиля пользователя. Результат представлен на рисунке 32.

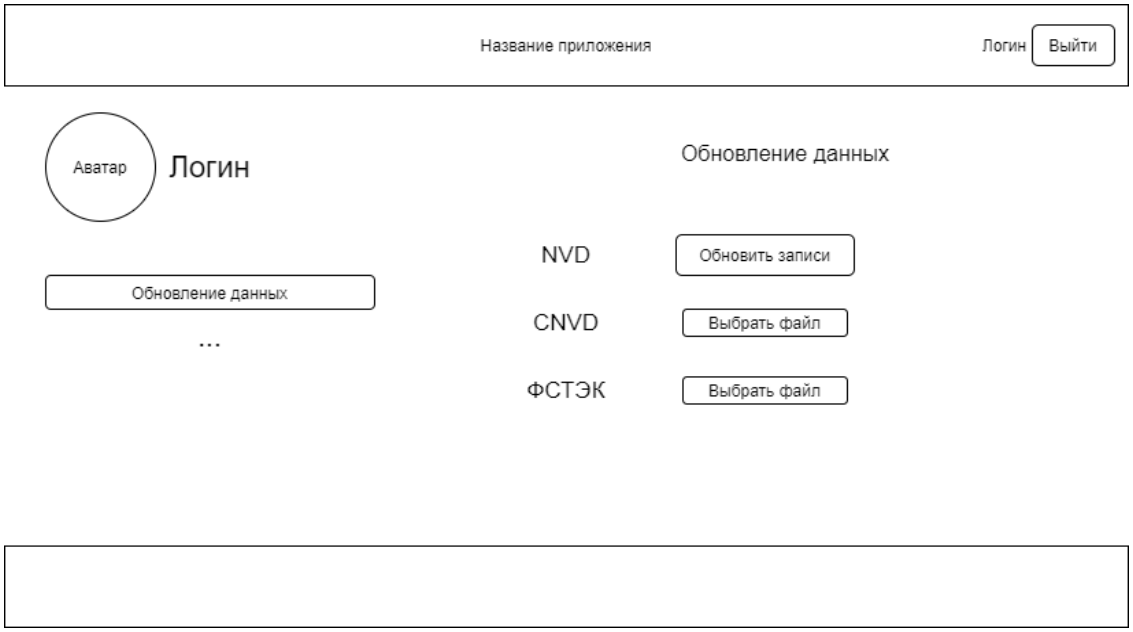


Рисунок 32 - Пользовательский интерфейс экрана профиля

Была сверстана страница авторизации. Результат представлен на рисунке 33.

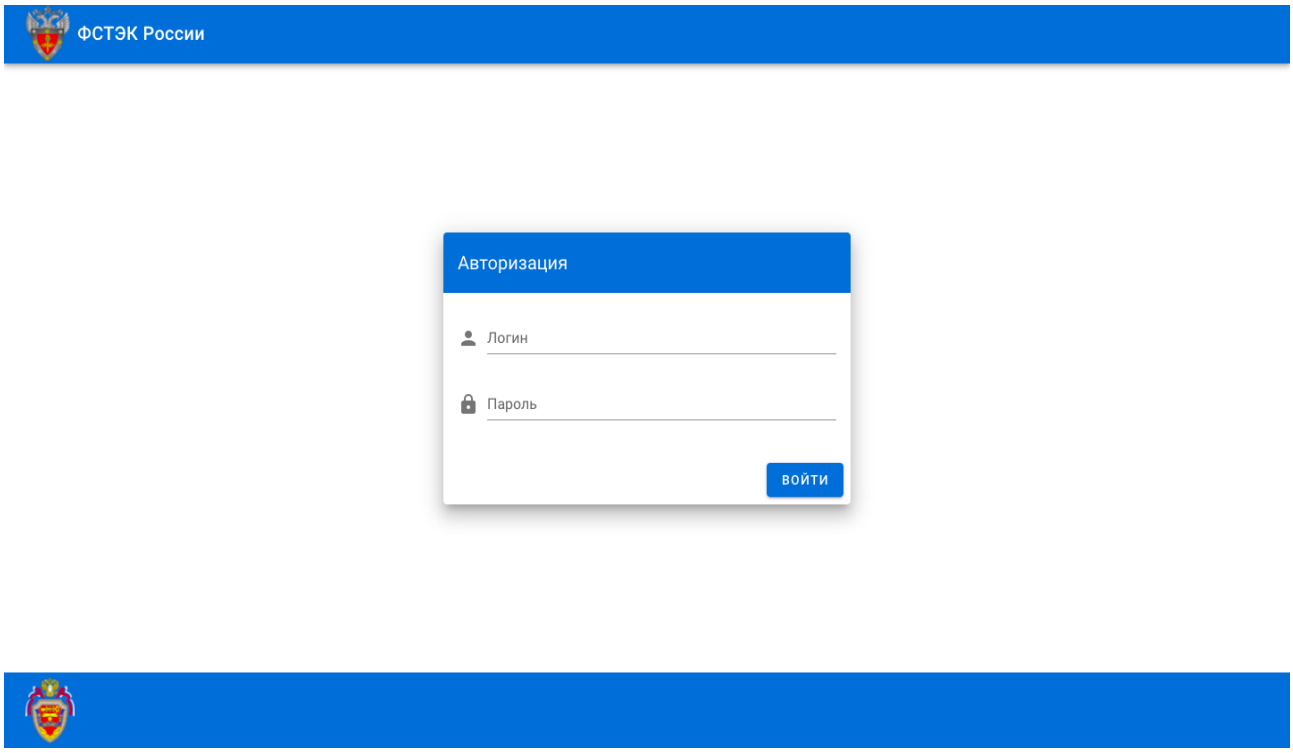


Рисунок 33 – Экран авторизации

Была реализована логика авторизации.

Клиент отправляет запрос на адрес `/login`, в теле запроса отправляются логин и пароль.

Затем сервер получает эти данные, если пользователь есть в базе данных, и он не заблокирован, то аутентифицирует его. После этого сразу отправляется запрос на адрес `/users/current`, чтобы получить данные о пользователе, который в данный момент авторизован.

Также был написан запрос на адрес `/logout`, который удаляет сессию пользователя.

Запросы к API представлены на рисунке 34.

```
import axios from './index'

export default {
  async getCurrentUser() {
    const resp = await axios.get('/users/current', {})
    return resp
  },

  async login(payload) {
    const resp = await axios.post('/login', { ...payload })
    return resp
  },

  async logout() {
    const resp = await axios.get('/logout', {})
    return resp
  }
}
```

Рисунок 34 – Запросы к API

Была сверстана страница поиска уязвимостей. Результат представлен на рисунке 35.

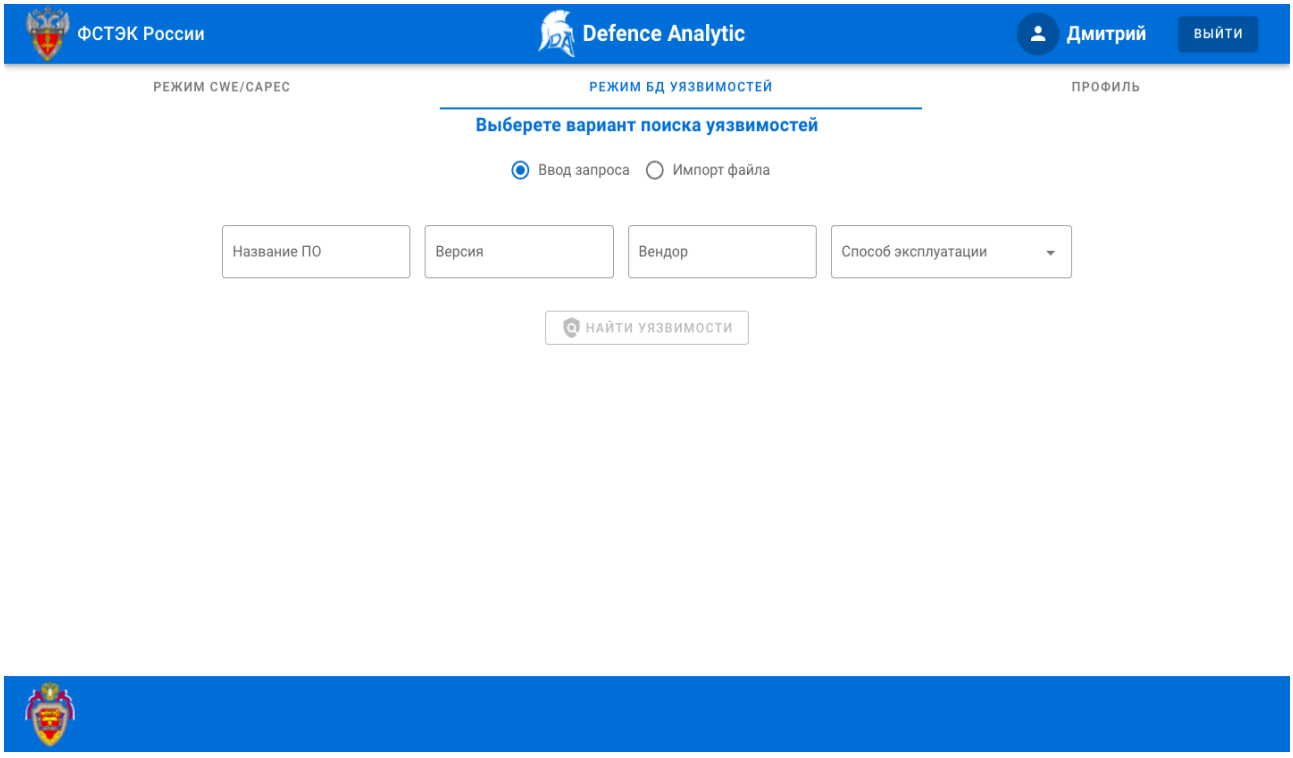


Рисунок 35 – Страница поиска уязвимостей

На данной странице пользователь может заполнить данные, в соответствии с которыми ему будет выдан список уязвимостей.

Была реализована логика получения уязвимостей.

Клиент отправляет запрос на адрес `/search_vulnerabilities`. В теле запроса отправляется два обязательных параметра:

– название ПО;

– версия;

А также возможна отправка двух опциональных полей:

– вендор;

– способ эксплуатации (локально или удаленно).

Запрос к API представлен на рисунке 36.

```
import axios from './index'

export default {
  async getVunerabilitiesFromQuery(payload) {
    const resp = await axios.get('/search_vulnerabilities', {
      params: { ...payload }
    })
    return resp
  }
}
```

Рисунок 36 – Запрос к API

Была сверстана таблица, в которой будут выводиться результаты поиска уязвимостей. Результат представлен на рисунке 37.

Она имеет следующие поля:

– локальный идентификатор (ID);

– источник (БДУ в который была найдена уязвимость);

– дата публикации;

– описание;

– CVSSv2 (если есть);

– CVSSv3 (если есть);

– конфигурации (если есть).

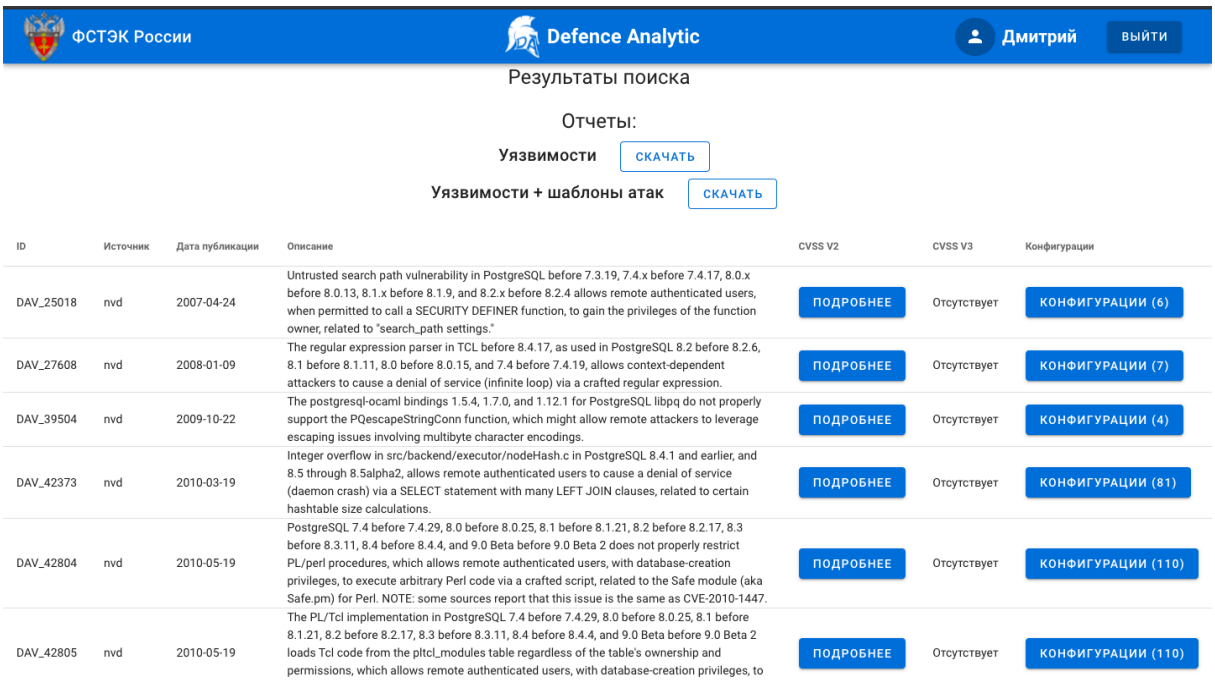


Рисунок 37 – Таблица с результатами поиска

Также данная страница предоставляет возможность пользователю скачать результаты поиска в виде файла формата xls. Результат представлен на рисунке 38.

	A	B	C	D	E	F	G
1		ID	Название	Описание	Дата создания	Источник	CVSSv2
2	0	DAV_25018	CVE-2007-2138	Untrusted search path vulnerability in PostgreSQL before 7.3.19, 7.4.x before 7.4.17, 8.0.x before 8.0.13, 8.1.x before 8.1.9, and 8.2.x before 8.2.4 allows remote authenticated users, when permitted to call a SECURITY DEFINER function, to gain the privileges of the function owner, related to "search_path settings."	2007-04-24 20:19:00	nvd	av:n/ac:m/au:s/c:p/i:p/a:p
3	1	DAV_27608	CVE-2007-4772	The regular expression parser in TCL before 8.4.17, as used in PostgreSQL 8.2 before 8.2.6, 8.1 before 8.1.11, 8.0 before 8.0.15, and 7.4 before 7.4.19, allows context-dependent attackers to cause a denial of service (infinite loop) via a crafted regular expression.	2008-01-09 21:46:00	nvd	av:n/ac:l/au:s/c:n/i:n/a:p
4	2	DAV_39504	CVE-2009-2943	The postgresql-ocaml bindings 1.5.4, 1.7.0, and 1.12.1 for PostgreSQL libpq do not properly support the PQescapeStringConn function, which might allow remote attackers to leverage escaping issues involving multibyte character encodings.	2009-10-22 16:30:00	nvd	av:n/ac:l/au:n/c:p/i:p/a:p
5	3	DAV_42373	CVE-2010-0733	Integer overflow in src/backend/executor/nodeHash.c in PostgreSQL 8.4.1 and earlier, and 8.5 through 8.5alpha2, allows remote authenticated users to cause a denial of service (daemon crash) via a SELECT statement with many LEFT JOIN clauses, related to certain hashtable size calculations.	2010-03-19 19:30:00	nvd	av:n/ac:m/au:s/c:n/i:n/a:p
				PostgreSQL 7.4 before 7.4.29, 8.0 before 8.0.25, 8.1 before 8.1.21, 8.2 before 8.2.17, 8.3			

Уязвимости

Рисунок 38 – Результаты поиска в виде файла формата xls

Была сверстано модальное окно, в котором выводится CVSSv2. Также была написана функция, которая получает на вход вектор CVSSv2 и расшифровывает каждое его значение и приводит в понятный для пользователя вид. Результат представлен на рисунке 39.

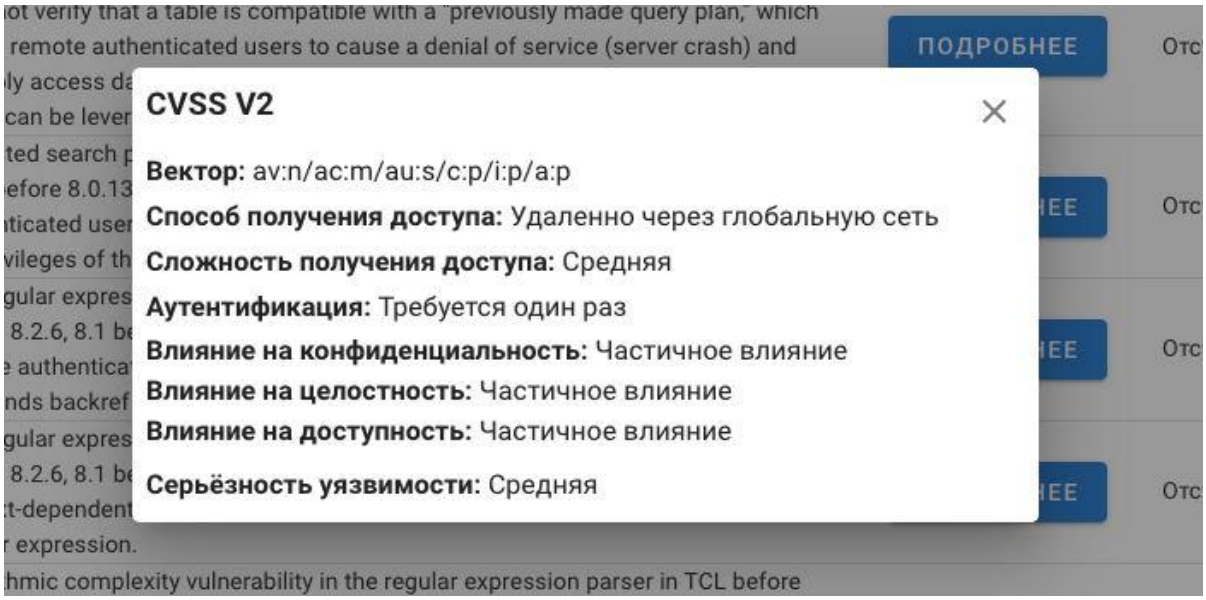


Рисунок 39 – Модальное окно CVSSv2

Была сверстано модальное окно в котором выводится CVSSv3. Также была написана функция, которая получает на вход вектор CVSSv3 и расшифровывает каждое его значение и приводит в понятный для пользователя вид. Результат представлен на рисунке 40.

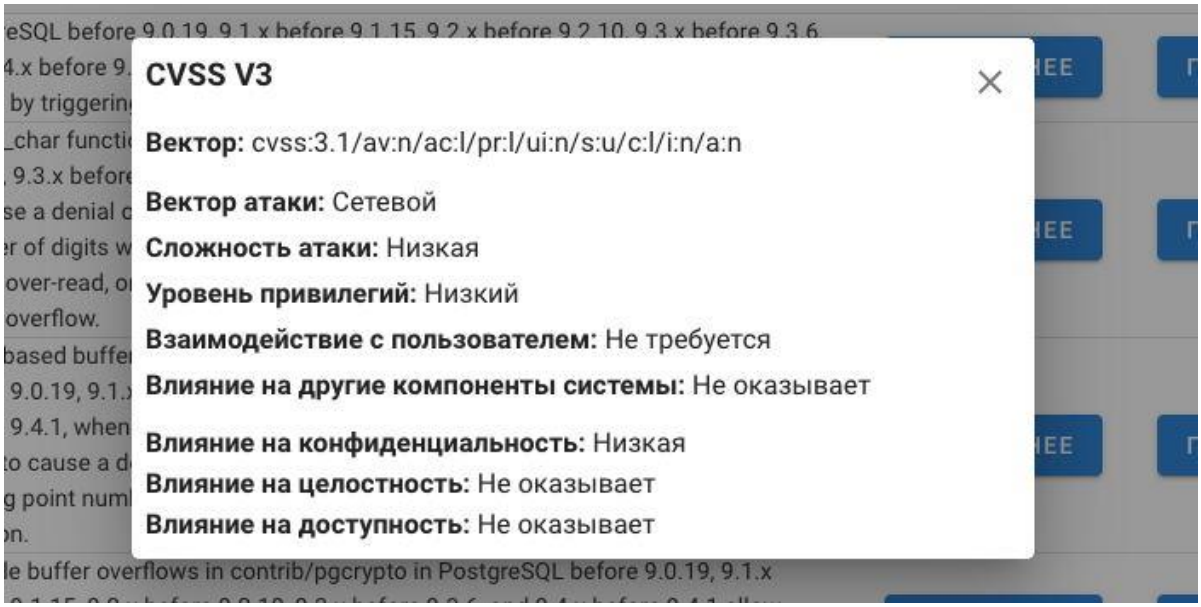


Рисунок 40 – Модальное окно CVSSv3

Была сверстано модальное окно в котором выводятся конфигурации. Результат представлен на рисунке 41.

Рисунок 41 – Модальное окно конфигураций

Была сверстана страница профиля пользователя. Результат представлен на рисунке 42.

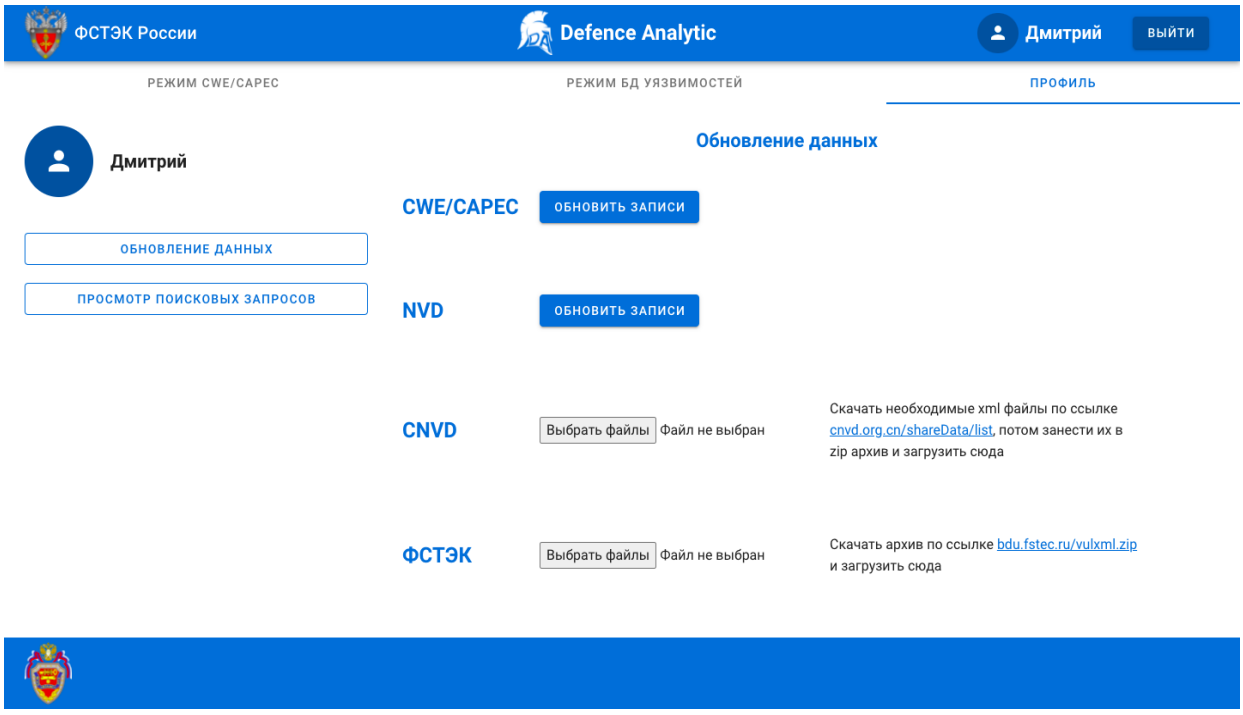


Рисунок 42 – Страница профиля пользователя

На данной странице пользователь может обновить записи во всех трех БДУ.

В БДУ NVD обновление происходит нажатием на кнопку с текстом «Обновить записи». При успешном запуске обновления записей в БДУ NVD, пользователь получает уведомление о том, что обновление успешно запущено. Уведомление представлено на рисунке 43.

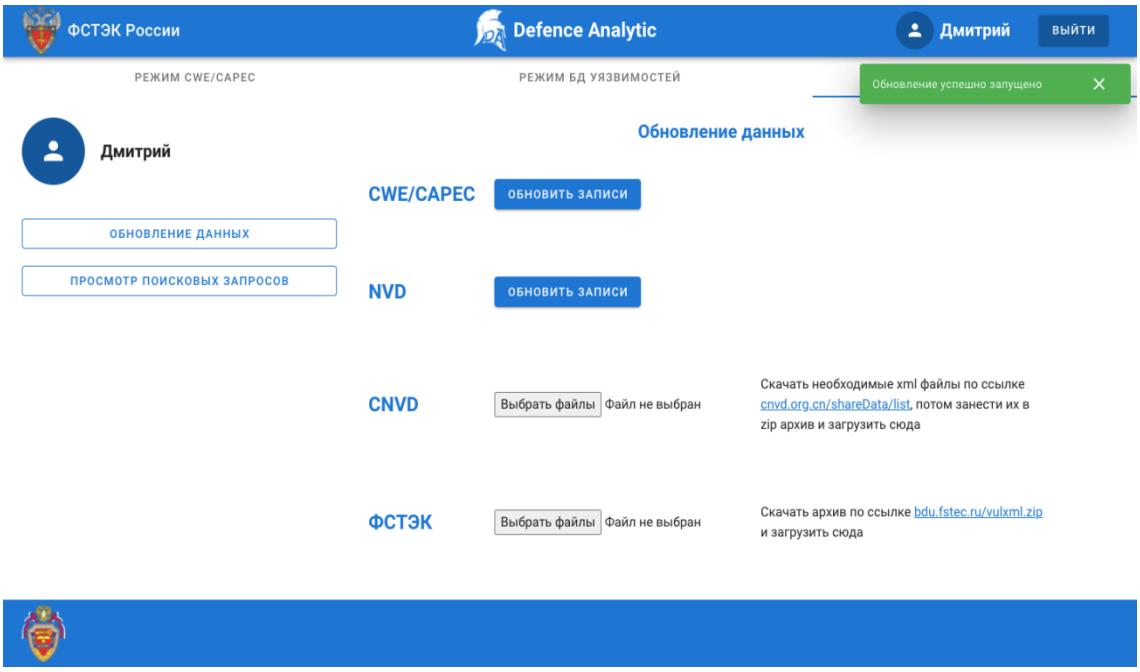


Рисунок 43 – Уведомление о запуске обновления записей в БДУ NVD

В БДУ CNVD и ФСТЭК обновление происходит с помощью загрузки zip архива. После добавления файла пользователю предоставляется возможность загрузить добавленный файл, либо же отменить данную процедуру. Результат, представлен на рисунке 44.

Рисунок 44 – Интерфейс конкретной БДУ после добавления файла

После нажатия на кнопку загрузки файла, запускается процесс загрузки файла на сервер, если формат файла не соответствует тому, который задает сервер, то пользователь получит уведомление с ошибкой. Результат представлен на рисунке 45. Если формат файл соответствует, то пользователь получит уведомление об успешном запуске процесса обновления. Результат представлен на рисунке 46.

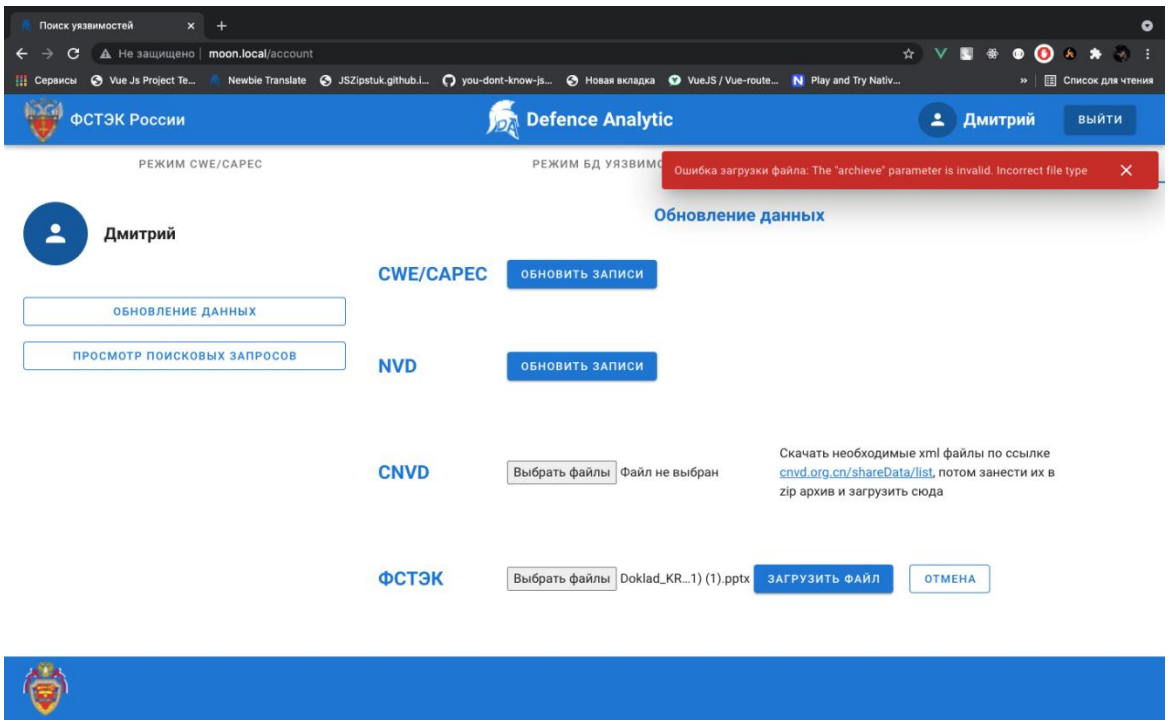


Рисунок 45 – Уведомление с ошибкой

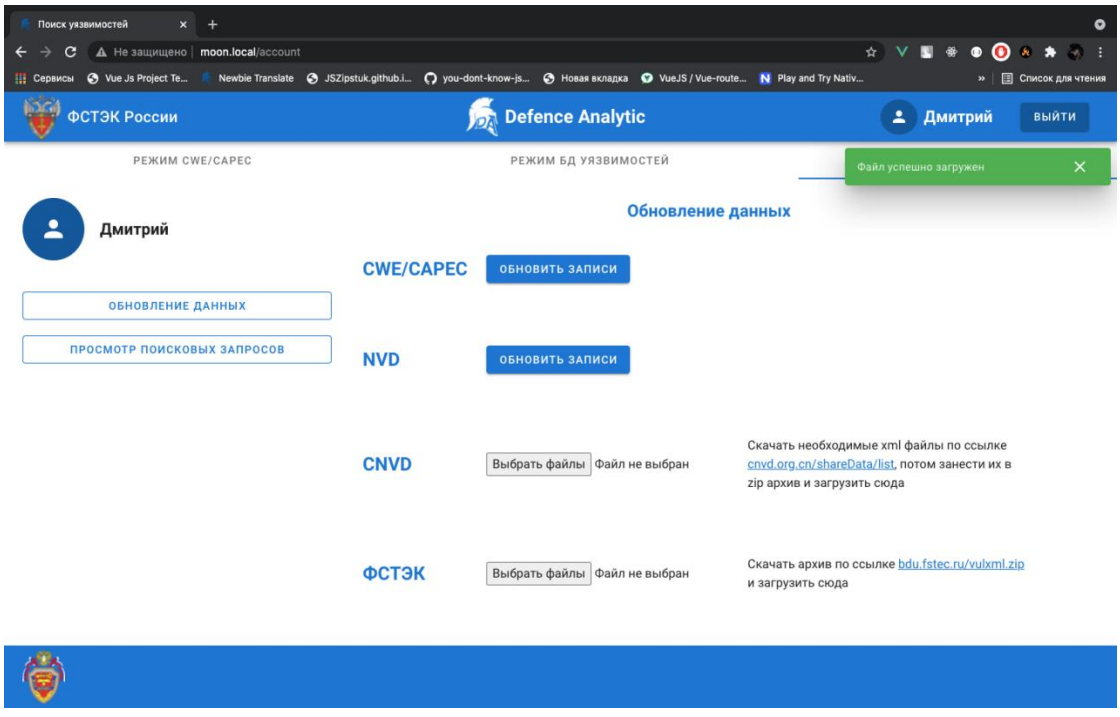


Рисунок 46 – Уведомление об успешном запуске процесса обновления БДУ

6 Вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

Специалисты по информационной безопасности при выполнении своих профессиональных задач часто подвергаются воздействию опасных и вредных факторов производства. Поэтому каждому сотруднику должен быть произведен инструктаж о возможности негативного влияния различных факторов производства на их организм.

Опасные и вредные производственные факторы делятся на физические, химические, биологические и психофизиологические. Последние в свою очередь подразделяются на физические и нервно-психические перегрузки.

К опасным психофизиологическим воздействиям относятся: чрезмерное умственное напряжение, чрезмерные физические нагрузки, однообразие работы.

В разработке новых технологий важнейшими задачами являются изучение и решение проблем, связанных с обеспечением здоровых и безопасных условий, в которых протекает труд человека. Эти условия помогают создавать выявление возможных причин производственных несчастных случаев, профессиональных заболеваний, аварий, взрывов, пожаров, и разработка мероприятий и требований, направленных на устранение этих причин.

Рабочая деятельность специалистов по информационной безопасности имеет некоторую специфику. Она состоит непосредственно из работы с большим объемом информации с использованием автоматизированных рабочих мест, СВТ, внедрения необходимых решений в короткие сроки, работы с людьми разного характера и др. Стоит также отметить, что работа специалистов связана с дополнительным вредным воздействием большой группы факторов, которые могут снижать производительность их труда. К таким факторам можно отнести:

– воздействие вредных излучений от монитора;
– неправильная освещенность;
– не нормированный уровень шума;
– нарушение микроклимата;
– наличие напряжения.

Постоянная нагрузка на нервную систему человека и его психику способствует снижению его функциональной активности, появляется предрасположенность к развитию заболеваний нервной системы, расстройствам, переутомлению, перегрузкам, стрессам. Стоит отметить, что длительная работа на автоматизированном рабочем месте (работа за монитором) организует чрезмерную нагрузку на все компоненты зрительного аппарата человека, а также способствует утомлению и перегрузке глазного анализатора. Работа, которая вызывает напряжение зрительного аппарата, может привести к следующим заболеваниям глаз:

– боль в глазах;
– покраснение глаз и век;
– жжение в глазах;
– появление глазной пелены;
– раздвоение предметов;
– неприятное мерцание;
– чрезмерная утомляемость при напряженной зрительной работе.

Все это коренным образом способствует нарушению работы органов зрения, что в итоге приводит к возникновению головной боли, появлению напряжения, а это, в свою очередь, обостряет нервно-психическое расстройство и снижает трудоспособность специалиста.

Можно сделать вывод, что к выбору методов и средств обеспечения безопасности следует относиться ответственно, учитывая факторы, присущие тому или иному производственному оборудованию или технологическому процессу.

Для нормальной работы зрительной системы необходимо соблюдать правила безопасности при работе с компьютером. У большого количества людей нет проблем со зрением, но в то же время их зрительные привычки не соответствуют правилам и физиологическим нормам. Таким образом, в конце концов, это приводит к нарушению зрительного аппарата. Чтобы избежать рисков и возможных проблем, необходимо постоянно проводить профилактику ухудшения зрения, которая включает в себя комплекс мер, направленных на сохранение здоровья глаз.

Перенапряжение глазодвигательных мышц является причиной большинства болезней органов зрения. Данное состояние приводит к ослаблению местного кровообращения и ухудшению трофики тканей (клеточного питания). Для защиты глаз от излишнего напряжения не рекомендуется длительное время фокусировать зрение на одном и том же предмете. В целях профилактики необходимо выработать полезную привычку: периодически отрывать глаза от работы и переводить взгляд на отдаленный предмет. Это простое действие способствует поддержанию упругости глазных мышц и получению постоянно четкого изображения предметов на сетчатке

Существует большое количество методов, позволяющих добиться расслабления глаз. Метод пальминга в последнее время пользуется популярностью. Суть упражнения заключается в том, что ладони обеих рук кладутся на закрытые глаза (без давления), в таком положении человек остается от 4 до 5 минут. Эта процедура направлена на естественное восстановление зрения, и ее рекомендуется выполнять ежедневно. Также не менее эффективной профилактикой для глаз является распределение по поверхности глазного яблока слезной жидкости, обладающей увлажняющими и дезинфицирующими свойствами, необходимо часто моргать. Это обеспечивает короткий отдых, и в результате укрепляет нервные волокна сетчатки, расслабляет глазные мышцы, улучшает центрирование и способность глаза фокусировать изображение.

При помощи солнечного света также можно укреплять сетчатку глаза, т.к. глаза, созданные для восприятия световых лучей, нуждаются в естественном

освещении не меньше, чем в периодическом отдыхе и расслаблении. Каждый день следует начинать с принятия световых ванн (соляризации). Закрытые глаза на несколько минут подставляют к солнцу или, при его отсутствии, другому источнику яркого света. В результате этой процедуры усиливается кровообращение, происходит нормализация обменных процессов в тканях глазного яблока и сохранение зрения.

Создание благоприятных условий труда и правильное эстетическое оформление рабочих мест на производстве имеет большое значение, как для облегчения труда, так и для повышения его привлекательности, положительно влияющей на производительность труда. Окраска помещений и мебели должна способствовать созданию благоприятных условий для зрительного восприятия, хорошего настроения. В служебных помещениях, в которых выполняется однообразная умственная работа, требующая значительного нервного напряжения и большого сосредоточения, окраска должна быть спокойных тонов – малонасыщенные оттенки холодного зеленого или голубого цветов.

Организация рабочего места и взаимное расположение всех его элементов должно соответствовать антропометрическим, физическим и психологическим требованиям. При правильной организации рабочего места производительность труда работника возрастает с 8 до 20 процентов.

Немаловажный фактор, повышающий остроту зрения и предотвращающий ухудшение зрения – это правильное освещение. Искусственный свет не должен резать глаза, однако при этом, он должен быть достаточно ярким.

Следует ограничивать прямую блесткость от источников освещения, при этом яркость светящихся поверхностей (окна, светильники и др.), находящихся в поле зрения, не должна быть более 200 кд/кв.м.

Следует ограничивать отраженную блесткость на рабочих поверхностях (экран, стол, клавиатура и др.) за счет правильного выбора типов светильников и расположения рабочих мест по отношению к источникам естественного и искусственного освещения.

Специалисты рекомендуют использовать настольные лампы мощностью 60 Вт с зонтиками для сохранения зрения при работе в офисе. Источник света должен располагаться (справа) на расстоянии 50 см от документов.

Также для обеспечения нормируемых значений освещенности в помещениях, где используют мониторы и ПЭВМ, следует проводить чистку стекол оконных рам и светильников не реже двух раз в год и проводить своевременную замену перегоревших ламп.

Заключение

В результате выполнения дипломной работы был проведен анализ нормативно-методической базы ФСТЭК России в части сертификационных испытаний СЗИ; изучен формат и практики использования общедоступных баз данных уязвимостей при проведении сертификационных испытаний СЗИ; выявлены основные проблемы общедоступных баз данных уязвимостей; разработана серверная часть ПК САПУ на языке Python, используя фреймворк falcon; спроектирован пользовательский интерфейс ПК САПУ; разработана клиентская часть ПК САПУ на языке JavaScript, используя фреймворк VueJS; рассмотрены вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации и приведены методы и средства физической культуры, снижающие их воздействие.

Таким образом, задание на дипломную работу выполнено в полном объёме.

В процессе выполнения выпускной квалификационной работы была подготовлена и опубликована статья в сборнике статей VIII Всероссийской межвузовской научно-практической конференции «Информационные технологии в науке и образовании. Проблемы и перспективы 2021» на тему «Вопросы автоматизации идентификации уязвимостей средств защиты информации при проведении сертификационных испытаний».

Список используемых источников

1. «Положение о системе сертификации средств защиты информации» (утверждено приказом ФСТЭК России от 3.04.18 г. №55) [Текст]. — [Электронный ресурс] — URL: https://docs.cntd.ru/document/9047649.

2. «Требования по безопасности информации, устанавливающим уровни доверия к средствам технической защиты информации и средствам обеспечения безопасности информационных технологий» (утверждены приказом ФСТЭК России от 02.06.2020 г. № 76) [Текст]. — [Электронный ресурс] — URL: https://docs.cntd.ru/document/565278856.

3. «Методика выявления уязвимостей и недекларированных возможностей в программном обеспечении» (утвержден ФСТЭК России 11 февраля 2019 г.) [Текст]. — [Электронный ресурс] — URL: https://clck.ru/VV655.

4. The Falcon Web Framework – Documentation [Текст]. — [Электронный ресурс] — URL: https://falcon.readthedocs.io/en/stable/index.html

5. Vue.js – Documentation [Текст]. — [Электронный ресурс] — URL: https://ru.vuejs.org/.

6. Vuetify Material Design Framework – Documentation [Текст]. — [Электронный ресурс] — URL: https://vuetifyjs.com/en/.

7. Nginx – Documentation [Текст]. — [Электронный ресурс] — URL: https://nginx.org/ru/.

8. Redis – Documentation [Текст]. — [Электронный ресурс] — URL: https://redis.io/.

Приложение А

(обязательное)

Код алгоритма обработки БДУ NVD

```
import sys
import re
import requests
import json
from datetime import datetime as dt
from urllib.request import urlretrieve

from zipfile import ZipFile

from libs.filestorage import filestorage
from models.nvd_vulnerability import NVDVulnerability
from models.nvd_configuration import NVDConfiguration
from models.cvss import CVSSv2, CVSSv3

from config import NVD_URL

START_YEAR = 2002
NVD_STORAGE = 'storage/databases/nvd/'


def extract_archive(archive_path, direction):
    with ZipFile(archive_path) as zfile:
        list_of_files = zfile.namelist()
        for filename in list_of_files:
            if filename.endswith('.json'):
                zfile.extract(filename, direction)
                return f'{direction}{filename}'


def prepare_to_parse(cves):
```

```
    cve_configurations = []
    for conf in cves:
        for cve in conf:
            cve_configurations.append(cve)

    return cve_configurations


def convert_camel_case(phrase):
    return re.sub(r'(?<!^)(?=[A-Z])', '_', phrase).lower()


def parse_configuration(cve_configurations, nvd, need_check=True,
group=None):

    configurations = []

    for conf in cve_configurations:
        if need_check and not conf['vulnerable'] is True:
            continue

        conf_desc = [item for item in
conf['cpe23Uri'].split(':')[3:] if item != '*']

        if 'versionStartIncluding' in conf:
            version_start = conf['versionStartIncluding']
        else:
            version_start = None

        if 'versionEndExcluding' in conf:
            version_end = conf['versionEndExcluding']
        else:
            version_end = None

        if not version_start and not version_end and
```

```
len(conf_desc) >= 3:
            version_start = conf_desc[2]
            version_end = conf_desc[2]
        else:
            version_start = 'any'
            version_end = 'any'

        desc = '' + ','.join([item for item in
conf['cpe23Uri'].split(':') if item != '*'])

        check_configuration = NVDConfiguration.select().where(
            (NVDConfiguration.cpe23_uri == desc) &
            (NVDConfiguration.nvd == nvd.id)
        )

        if not check_configuration.exists():
            configuration = NVDConfiguration()
            configuration.nvd = nvd.id
            configuration.name = conf_desc[1]
            configuration.cpe23_uri = desc
            configuration.version_start = version_start
            configuration.version_end = version_end
        else:
            configuration = check_configuration.get()
            configuration.nvd = nvd.id
            configuration.name = conf_desc[1]
            configuration.cpe23_uri = desc
            configuration.version_start = version_start
            configuration.version_end = version_end

        configuration.vendor = conf_desc[0]

        if group:
            configuration.group = group
```

```
            configuration.save()
            configurations.append(configuration)

        return configurations


for year in range(START_YEAR, dt.now().year + 1):
    archive_path = f'{NVD_STORAGE}{year}.zip'
    urlretrieve(f'{NVD_URL}{year}.json.zip', archive_path)
    json_path = extract_archive(archive_path, NVD_STORAGE)
    filestorage.cleaner(archive_path)

    with open(json_path) as json_file:
        data = json.load(json_file)['CVE_Items']
        for item in data:
            cve_id = item['cve']['CVE_data_meta']['ID']
            cve_description = '' + \
                '\n'.join([desc['value'] for desc in item['cve']
                           ['description']['description_data']])
            cve_published_data = item['publishedDate']

            check_cve =
NVDVulnerability.select().where(NVDVulnerability.title == cve_id)

            if not check_cve.exists():
                nvd = NVDVulnerability()
                nvd.title = cve_id
                nvd.description = cve_description
                nvd.published_date = cve_published_data
                nvd.save()
            else:
                nvd = check_cve.get()
                nvd.description = cve_description
                nvd.published_date = cve_published_data
                nvd.save()
```

```
            if 'baseMetricV2' in item['impact']:

                check_cvss2 = CVSSv2.select().where(CVSSv2.nvd ==
nvd.id)

                if check_cvss2.exists():
                    cvss2 = check_cvss2.get()
                else:
                    cvss2 = CVSSv2()
                    cvss2.nvd = nvd.id

                cvssV2_desc =
item['impact']['baseMetricV2'].pop('cvssV2')

                for key, value in
item['impact']['baseMetricV2'].items():
                    cvssV2_desc[key] = value

                for key, value in cvssV2_desc.items():

                    if key == 'version':
                        continue

                    cvss2.__setattr__(
                        convert_camel_case(key),
                        value.lower() if type(value) == str else
value
                    )

                cvss2.save()

            if 'baseMetricV3' in item['impact']:

                check_cvss3 = CVSSv3.select().where(CVSSv3.nvd ==
```

```
nvd.id)

                if check_cvss3.exists():
                    cvss3 = check_cvss3.get()
                else:
                    cvss3 = CVSSv3()
                    cvss3.nvd = nvd.id

                cvssV3_desc =
item['impact']['baseMetricV3'].pop('cvssV3')

                for key, value in
item['impact']['baseMetricV3'].items():
                    cvssV3_desc[key] = value

                for key, value in cvssV3_desc.items():

                    if key == 'version':
                        continue

                    cvss3.__setattr__(
                        convert_camel_case(key),
                        value.lower() if type(value) == str else
value
                    )

                    cvss3.save()

            rel_cves = []
            cves = []

            for cve in item['configurations']['nodes']:
                if cve['operator'] == 'AND' and 'children' in cve:
                    rel_cves.append(cve['children'])
                else:
```

```
                cves.append(cve['cpe_match'])

        counter = 1

        for children in rel_cves:
            children_counter = 1

            for cpe in children:
                conf = parse_configuration(
                    cpe['cpe_match'],
                    nvd,
                    need_check=False,
                    group=f'{counter}_{children_counter}'
                )

                children_counter += 1

            counter += 1

        cve_configurations = []
        for conf in cves:
            for cve in conf:
                cve_configurations.append(cve)

        parse_configuration(cve_configurations, nvd)

filestorage.cleaner(json_path)
```

Приложение Б

(обязательное)

Код алгоритма обработки БДУ CNVD

```
import os
import collections
import requests
import urllib
import goslate
from urllib.request import urlretrieve
from libs.xml import xml_parser, check_type
from libs.logger import logger

from models.nvd_vulnerability import NVDVulnerability
from models.nvd_configuration import NVDConfiguration

from googletrans import Translator


g_translator = goslate.Goslate()


def translator(text):

    result = g_translator.translate(text, 'ru')
    return result

cnvd_storage = 'storage/databases/cnvd'
files_list = os.listdir(cnvd_storage)

for file in files_list:
    raw_vulnerabilities = xml_parser(f'{cnvd_storage}/{file}')
```

```
    if raw_vulnerabilities is None or
raw_vulnerabilities['vulnerabilitys'] is None:
        continue

    vulnerabilities =
raw_vulnerabilities['vulnerabilitys']['vulnerability']

    for raw_vulnerability in vulnerabilities:

        check_vulnerability = NVDVulnerability.select().where(
            NVDVulnerability.title == raw_vulnerability['number']
        )

        if check_vulnerability.exists():
            vulnerability = check_vulnerability.get()
        else:
            vulnerability = NVDVulnerability()
            vulnerability.title = raw_vulnerability['number']

        if 'description' in raw_vulnerability:
            vulnerability.description =
translator(raw_vulnerability['description'])

        if 'submitTime' in raw_vulnerability:
            vulnerability.published_date =
raw_vulnerability['submitTime']

        try:
            vulnerability.save()
        except Exception as e:
            logger.error(e)

        if 'products' in raw_vulnerability:
            products = raw_vulnerability['products']['product']
```

```
            for product in products:
                check_configuration                                    =
NVDConfiguration.select().where(
                    (NVDConfiguration.cpe23_uri == product) &
                    (NVDConfiguration.nvd == vulnerability.id)
                )

                if not check_configuration.exists():
                    configuration = NVDConfiguration()
                    configuration.nvd = vulnerability.id
                    configuration.cpe23_uri = product
                    configuration.save()

logger.info('All CNVD vulnerabilities already saved')
```

Приложение В

(обязательное)

Код алгоритма обработки БДУ ФСТЭК

```
import os
import collections
from urllib.request import urlretrieve

from libs.filestorage import filestorage
from libs.logger import logger
from libs.xml import xml_parser, extract_archive_xml, check_type

from models.nvd_vulnerability import NVDVulnerability
from models.nvd_configuration import NVDConfiguration
from models.cvss import CVSSv2

direction = 'storage/databases/export.xml'
parsed_xml = xml_parser(direction)


vulnerabilities = parsed_xml['vulnerabilities']['vul']

for raw_vulnerability in vulnerabilities:

    check_vulnerability = NVDVulnerability.select().where(
        NVDVulnerability.title == raw_vulnerability['identifier']
    )

    if check_vulnerability.exists():
        vulnerability = check_vulnerability.get()
    else:
        vulnerability = NVDVulnerability()
        vulnerability.title = raw_vulnerability['identifier']
```

```
    if 'description' in raw_vulnerability:
        vulnerability.description                                    =
raw_vulnerability['description']

    try:
        vulnerability.save()
    except Exception as e:
        logger.error(e)

    if 'cvss' in raw_vulnerability:
        check_cvss = CVSSv2.select().where(
            (CVSSv2.nvd == vulnerability.id)
        )

        if check_cvss.exists():
            cvss = check_cvss.get()
        else:
            cvss = CVSSv2()
            cvss.nvd = vulnerability.id

        cvss.vector_string                                           =
raw_vulnerability['cvss']['vector']['#text']
        cvss.save()

    if 'vulnerable_software' in raw_vulnerability:
        configurations                                               =
raw_vulnerability['vulnerable_software']['soft']

        try:
            NVDConfiguration.delete().where(NVDConfiguration.nvd
== vulnerability.id).execute()
        except Exception as e:
            logger.error(e)

        if type(configurations) == collections.OrderedDict:
```

```
            check_configuration                                    =
NVDConfiguration().select().where(
                (NVDConfiguration.nvd == vulnerability.id) &
                (NVDConfiguration.name  ==  configurations['name'])
&
                (NVDConfiguration.cpe23_uri                        ==
configurations['version'])
            )

            if 'vendor' in configurations:
                check_configuration = check_configuration.where(
                    NVDConfiguration.vendor                        ==
configurations['vendor']
                )

            if not check_configuration.exists():
                configuration = NVDConfiguration(
                    nvd=vulnerability.id,
                    name=configurations['name'],
                    cpe23_uri=configurations['version']
                )

                if 'vendor' in configurations:
                    configuration.vendor                           =
configurations['vendor']

                configuration.save()
        else:
            for conf in configurations:

                check_configuration                                =
NVDConfiguration().select().where(
                    (NVDConfiguration.nvd == vulnerability.id) &
                    (NVDConfiguration.name == conf['name']) &
```

```
                    (NVDConfiguration.cpe23_uri                    ==
conf['version'])
                )

                if 'vendor' in conf:
                    check_configuration                             =
check_configuration.where(
                        NVDConfiguration.vendor == conf['vendor']
                    )

                if not check_configuration.exists():
                    configuration = NVDConfiguration(
                        nvd=vulnerability.id,
                        name=conf['name'],
                        cpe23_uri=conf['version']
                    )

                    if 'vendor' in conf:
                        configuration.vendor = conf['vendor']

                    configuration.save()
```

Приложение Г

(обязательное)

Контроллер для автоматизированного обновления

БДУ CNVD и ФСТЭК

```
class UpdateByArchievementController(Controller):

    def cleaner(self, direction):

        if os.path.isfile(direction):
            os.remove(direction)

    @falcon.before(auth_required)
    async def on_post(self, req, resp):

        media = await req.get_media()

        async for part in media:
            if part.name.startswith('archieve'):

                archieve_ext = part.content_type.rsplit("/")[1]
                if archieve_ext != 'zip':
                    raise    falcon.HTTPInvalidParam(msg='Incorrect
file type', paramname='archieve')

                database = part.name.split('')[1]
                if database not in ['fstek', 'cnvd']:
                    raise falcon.HTTPInvalidParam(
                        msg='Incorrect database name',
                        param_name='database'
                    )

                archieve = await part.stream.read()
            else:
```

```
                raise  falcon.HTTPBadRequest(description='Incorrect
file')

        try:
            self.cleaner(f'storage/databases/{database}/vul.zip')
        except Exception as e:
            raise
falcon.HTTPInternalServerError(description=f'Somthing  went  wrong
{e}')

        with  open(f'storage/databases/{database}/vul.zip',  'wb')
as f:
            f.write(archieve)


append_update_archieve_to_queue(f'storage/databases/{database}/vul
.zip', database)

        resp.text = json.dumps({'success': True})
```

Приложение Д

(обязательное)

Протокол проверки на оригинальность в системе «Антиплагиат.ВУЗ»

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

СПРАВКА

Пензенский государственный университет

о результатах проверки текстового документа
на наличие заимствований

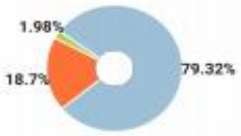
ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы: Ворошилло Дмитрий Борисович
Самоцитирование рассчитано для: Ворошилло Дмитрий Борисович
Название работы: Базовая система автоматизации поиска потенциальных уязвимостей программного обеспечения при проведении сертификационных испытаний средств защиты информации в системе сертификации ФСТЭК России
Тип работы: Дипломный проект
Подразделение: Кафедра ИБСТ

РЕЗУЛЬТАТЫ

ЗАИМСТВОВАНИЯ	18.7%
ОРИГИНАЛЬНОСТЬ	79.32%
ЦИТИРОВАНИЯ	1.98%
САМОЦИТИРОВАНИЯ	0%

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 20.06.2021



Модули поиска: ИПС Адилет; Библиография; Сводная коллекция ЭБС; Интернет Плюс; Сводная коллекция РГБ; Цитирование; Переводные заимствования (RuEn); Переводные заимствования по eLIBRARY.RU (EnRu); Переводные заимствования по Интернету (EnRu); Переводные заимствования издательства Wiley (RuEn); eLIBRARY.RU; СПС ГАРАНТ; Медицина; Диссертации НББ; Перефразирования по eLIBRARY.RU; Перефразирования по Интернету; Патенты СССР, РФ, СНГ; Модуль поиска "ПГУ"; СМИ России и СНГ; Шаблонные фразы; Кольцо вузов; Издательство Wiley; Переводные заимствования

Работу проверил: Иванов Алексей Петрович

ФИО проверяющего

Дата подписи:

Подпись проверяющего

Чтобы убедиться
в подлинности справки, используйте QR-код,
который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.