

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

|  |  |
| --- | --- |
| **Институт**  **информационных систем**  **и технологий** | **Кафедра**  **информационных систем** |

**Майорко Денис Сергеевич**

**Тема**: **«Разработка автоматизированной системы выявления вредоносного программного обеспечения на основе механизма динамического анализа»**

**Выпускная квалификационная работа на присвоение квалификации «бакалавр» по направлению 09.03.02 «Информационные системы и технологии»**

|  |  |  |
| --- | --- | --- |
| Заведующий кафедрой  д.т.н., проф. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Позднеев Б.М.**  подпись |  |
| Научный руководитель  к.т.н., доц. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Тясто С.А.**  подпись |  |
| Студент  группы ИДБ-15-13 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Майорко Д.С.**  подпись |  |
|  |  |  |
|  |  |  |

Регистрационный № \_\_\_\_\_\_\_\_

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc6836081)

[ГЛАВА 1. ИССЛЕДОВАНИЕ СОВРЕМЕННЫХ СРЕДСТВ ДИНАМИЧЕСКОГО АНАЛИЗА. 7](#_Toc6836082)

[1.1. СРАВНЕНИЕ СУЩЕСТВУЮЩИХ РЕШЕНИЙ 7](#_Toc6836083)

[1.2. МАСШТАБИРУЕМОСТЬ 8](#_Toc6836084)

[1.3. СТЕПЕНЬ ИНТЕГРИРОВАННОСТИ В ИНФРАСТРУКТУРУ КОМПАНИИ 11](#_Toc6836085)

[1.4. ОГРАНИЧЕНИЯ ПО ТЕХНИЧЕСКИМ ПАРАМЕТРАМ 11](#_Toc6836086)

[1.5. ОСОБЕННОСТИ ИНТЕРФЕЙСА 12](#_Toc6836087)

[1.6. ИСПОЛЬЗУЕМЫЕ МЕТОДИКИ АНАЛИЗА СОБЫТИЙ 13](#_Toc6836088)

[1.7. ВЫБОР ПЛАТФОРМЫ РЕАЛИЗАЦИИ И ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ 14](#_Toc6836089)

[1.8. ОСНОВНЫЕ НОРМАТИВНЫЕ ДОКУМЕНТЫ, ПРИМЕНЯЕМЫЕ ПРИ РАЗРАБОТКЕ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ 18](#_Toc6836090)

[ГЛАВА 2. ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ ПРОЦЕССА ИССЛЕДОВАНИЯ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ МЕХАНИЗМА ДИНАМИЧЕСКОГО АНАЛИЗА 20](#_Toc6836091)

[2.1. ОПРЕДЕЛЕНИЕ ПОДХОДОВ К РЕШЕНИЮ ПОСТАВЛЕННОЙ ЗАДАЧИ 20](#_Toc6836092)

[2.2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ МОДЕЛЕЙ МОДУЛЯ ИССЛЕДОВАНИЯ ВРЕДОНОСНОГО ПО НА ОСНОВЕ МЕХАНИЗМА ДИНАМИЧЕСКОГО АНАЛИЗА 22](#_Toc6836093)

[ГЛАВА 3. РАЗРАБОТКА МОДУЛЯ ВЫЯВЛЕНИЯ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ МЕХАНИЗМА ДИНАМИЧЕСКОГО АНАЛИЗА 34](#_Toc6836094)

[3.1. ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ РЕАЛИЗАЦИИ. 34](#_Toc6836095)

[3.2. РЕАЛИЗАЦИЯ МОДУЛЯ ВЫЯВЛЕНИЯ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ МЕХАНИЗМА ДИНАМИЧЕСКОГО АНАЛИЗА 35](#_Toc6836096)

[ЗАКЛЮЧЕНИЕ 45](#_Toc6836097)

[СПИСОК ЛИТЕРАТУРЫ 46](#_Toc6836098)

[ПРИЛОЖЕНИЕ 48](#_Toc6836099)

[ЛИСТНИНГ КОДА 48](#_Toc6836100)

# **ВВЕДЕНИЕ**

Сложность и количество современных киберугроз неизменно растет. Одной из самых опасных современных форм атак являются таргетированные атаки. Они, как правило, начинаются с запуска загруженных файлов и почтовых электронных вложений. Для осуществления таргетированной атаки могут использовать как известные вирусы, информация по которым присутствует в базах данных антивирусных решений, так и новые вирусы, ранее неизвестные, что затрудняет их обнаружение традиционными решениями. Также, с развитием вирусов, появились новые бесфайловые вредоносные программы. [7]

Для обеспечения надежной защиты современная антивирусная система должна:

* Обнаруживать вредоносные программы разного рода;
* Обнаруживать новые модификации известных вредоносных программ;
* Обнаруживать вредоносное программное обеспечение, упакованное в архив;
* Обнаруживать подозрительное функционирование в выполняемой среде.

Существует множество решений для проверки программного обеспечения на вредоносные действия, но большинство из них пропускает много новых вирусов, имеет большой процент ложных срабатываний, не носит комплексный характер, имеет сложности в установке и поддержке. В современных решениях можно выделить следующие наиболее востребованные методы анализа вредоносного программного обеспечения:

* сканирование;
* обнаружение изменений;
* эвристический анализ.

Программа, выполняющая сканирование просматривает содержимое файлов, расположенных на дисках компьютера, а также содержимое оперативной памяти компьютера с целью поиска вирусов. При этом, классическое сканирование предполагает поиск вредоносных программ по их сигнатурам – по последовательностям байтов данных, характерных для данных вирусов. [5]

Метод сканирования позволяет обнаружить такие вредоносные программы, которые не используют для противодействия антивирусным программам шифрование своего программного кода, а также полиморфизм. [6]

Посредством данного метода можно обнаружить как вирус, так и сетевого червя, но приложение, использующее данный способ требует постоянной поддержки, а именно, обновления базы данных вирусов, поэтому данный способ не имеет смысла без постоянного сопровождения.

При поиске вредоносного программного обеспечения методом обнаружения изменений периодически сканируется содержимое дисков компьютера и записываются контрольные суммы файлов и критически важных внутренних областей файловых систем. При анализе новые значения контрольных сумм сравниваются со старыми значениями. Основным недостатком данного подхода является то, что вредоносную программу можно обнаружить только после своего выполнения. Также необходимо учитывать, что изначально проверяемые приложения устанавливаются из доверенного источника.

При использовании эвристического анализа контролируются все действия, которые может выполнить проверяемая программа. При этом отслеживаются потенциально опасные действия, характерные для вредоносного программного обеспечения, однако подход также имеет ряд недостатков. Одними из важнейших, заключается в возможности появления «ложной» тревоги. Кроме того, данный вид анализа занимает большой отрезок процессорного времени, поэтому не всегда применим.

Учитывая многие факторы, эвристический (динамический) анализ, может применяться с большей результативностью, однако реализация требует особого подхода для повышения эффективности. Данный вид анализа набирает все большую популярность, особенно, среди крупных компаний. Причиной такой тенденции является эффективность подхода, препятствием же является стоимость и сложность систем.

В сравнении данных методов преимущества эвристического анализа очевидны. Повышение надежности защиты является главным показателем в борьбе с вредоносным программным обеспечением с учетом возможных многомиллионных потерь крупных компаний.

При условии постоянного совершенствования вредоносного программного обеспечения, даже самые современные методики выявления необходимо совершенствовать.

С учетом данного факта была поставлена цель исследования –повысить качество вынесения вердиктов и уменьшить количество затрачиваемых на анализ ресурсов с помощью разработки автоматизированной системы выявления вредоносного по на основе механизма динамического анализа поведения программного обеспечения.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать современные методы анализа BigData, существующие средства выявления вредоносного программного обеспечения на основе механизма динамического анализа;

2. Разработать функциональную модель процесса выявления вредоносного программного обеспечения на основе механизма динамического анализа;

3. Разработать эффективную методику выявления вредоносного программного обеспечения на основе механизма динамического анализа;

4. Разработать автоматизированную систему выявления вредоносного программного обеспечения на основе механизма динамического анализа.

# **ИССЛЕДОВАНИЕ СОВРЕМЕННЫХ СРЕДСТВ ДИНАМИЧЕСКОГО АНАЛИЗА.**

## **СРАВНЕНИЕ СУЩЕСТВУЮЩИХ РЕШЕНИЙ**

На данный момент рынок антивирусных систем на основе динамического анализа представлен большим числом продуктов в этой сфере. Среди зарубежных конкурирующих компаний и их разработок можно выделить:

* SandBlast (от компании Check Point);
* FortiSandbox (от компании Fortinet);
* Joe Sandbox (от компании Joe Security);
* AMP Threat Grid (от компании Cisco);
* WildFire (от компании PaloAlto Networks);
* Threat Intelligence (от компании FireEye);
* Lastline Sandbox (от компании Lastline);
* VxStream Sandbox (от компании Payload-Security).

Также существуют аналоги, созданные российскими разработчиками:

* Kaspersky Anti Targeted Attack (KATA) Platform (от компании Лаборатория Касперского);
* TDS Polygon (от компании Group-IB);
* Athena (от компании АВ-Софт).

В рамках данной работы будут рассматриваться последние, так как, разрабатываясь отечественным производителем, они наиболее точно отвечают потребностям российского рынка. Кроме того, в связи с существующей ситуацией в Российской Федерации, многочисленными указами президента и распоряжениями и постановления правительства, существует потребность в импортозамещении программного обеспечения [9].

Выбранные разработки одинаково направлены на анализ и выявление вредоносного программного обеспечения на основе метода динамического анализа, однако существуют значимые различия по многим параметрам. Среди наиболее значимых и различимых оценок можно выделить:

* масштабируемость;
* степень интегрированности в инфраструктуру компании;
* ограничения по техническим параметрам;
* особенности интерфейса;
* эффективность выявления вредоносного программного обеспечения.

## **МАСШТАБИРУЕМОСТЬ**

Во всех системах наблюдается большая схожесть архитектур. В каждой из систем присутствует:

* легковесный клиент, следящий за активностью на устройстве;
* полигон динамического анализа;
* центр анализа информации;
* сервер хранения информации;
* центр анализа трафика;
* интерфейс пользователя.

С помощью данной архитектуры реализуются следующие задачи:

* все файлы, в том числе те, что появились до активации системы на устройстве пользователя, находится под контролем антивируса;
* все новые файлы, вошедшие в зону мониторинга предварительно анализируются и тестируются;
* весь входящий и исходящий трафик системы проверяется на несанкционированную утечку данных.

Однако одним из главных различий в программных продуктах являются особенности реализаций данной архитектуры.

Так, в разработке TDS-Polygon для установки системы предусмотрены только собственные устройства:

* Polygon-2500S;
* Polygon-5000S;
* Polygon-10000S.

Для полного функционирования системы необходимы следующие подсистемы (рис. 1.1):

* TDS Polygon (песочница);
* TDS Sensor (узел анализа трафика);
* Подключение к SOC Group-IB (центр обработки данных).

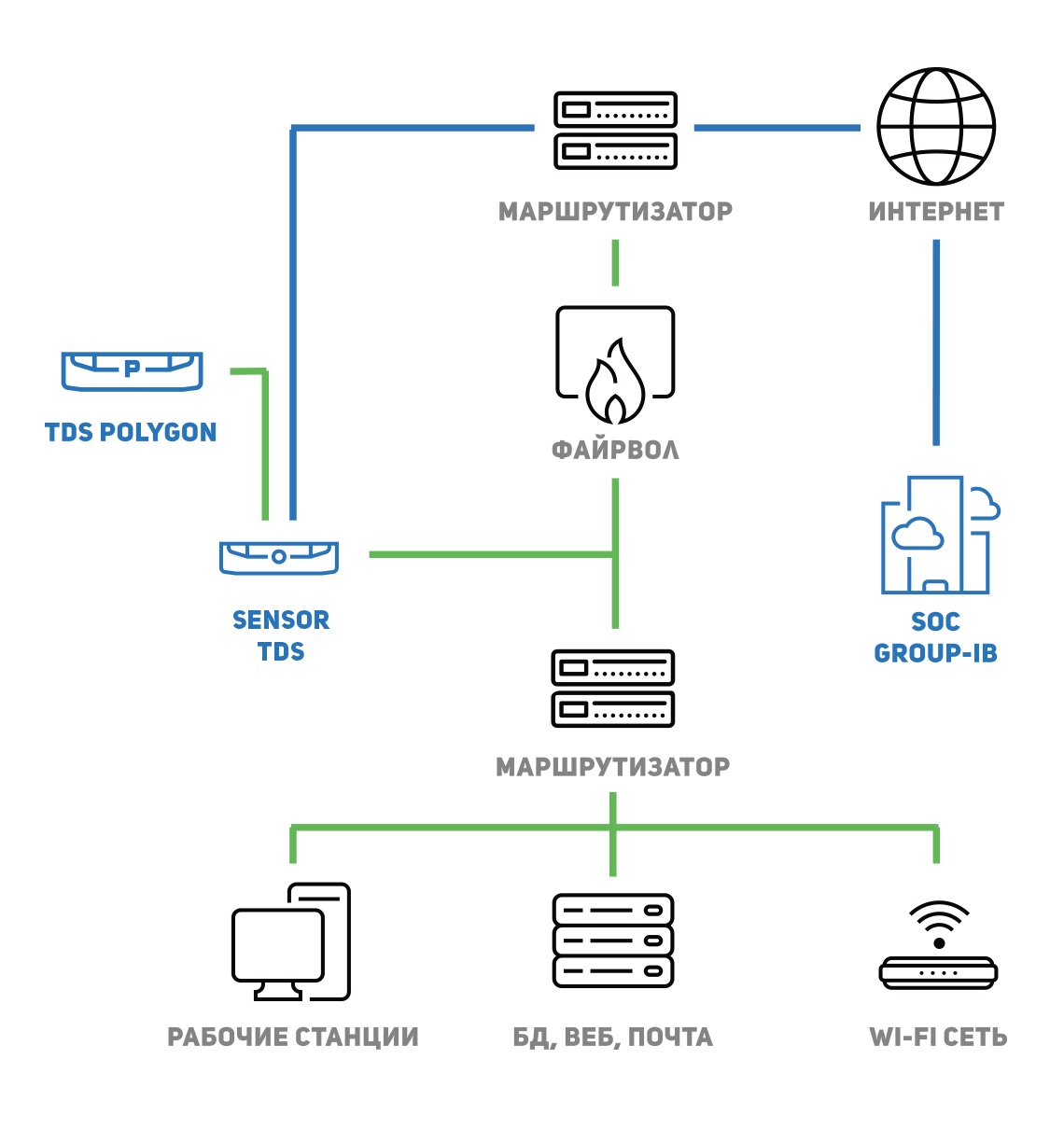


Рис. 1.1. Архитектура TDS-Polygon с подключением SOC

Данные особенности ведут к закрытости экосистемы, а значит любые неисправности могут быть решены только с участием компании – поставщика.

Кроме того, для полного анализа необходимо будет высылать данные использования программ за пределы инфраструктуры компании. Среди посылаемых данных может быть любая информация, что кратно повышает риски разглашения конфиденциальной информации.

Однако стоит заметить, что при такой архитектуре повышается уровень надежности системы благодаря точно подобранным сочетаниям машина-программа. Другим же плюсом будет удобство поддержки пользователей, благодаря шаблонным реализациям.

Существует также решение, при котором анализ событий отдается компании-потребителю системы, однако при этом отсутствует сам инструмент анализа, что без дополнительных затрат приводит только к скоплению информации о событиях (рис. 1.2).

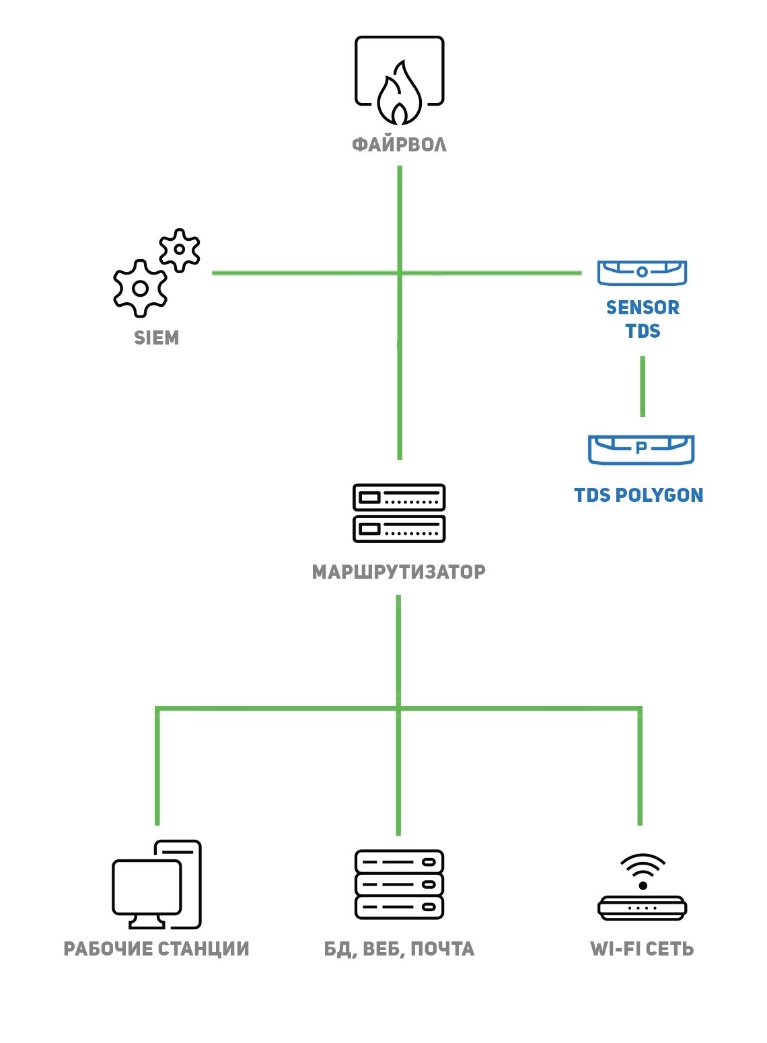


Рис. 1.2. Архитектура TDS-Polygon без подключения к SOC

В разработках KATA и Athena системы могут быть развернуты на устройствах, собранных из компонентов разных производителей. В данном случае необходимо отметить, что для KATA существует список жестких системных требований к серверам развертывания, что резко повышает стоимость всей системы. Athena же может быть развернута практически на любом устройстве. Системные требования к ним будут зависеть от предполагаемых нагрузок, а узлы тестирования могут масштабироваться в зависимости от потребностей.

## **СТЕПЕНЬ ИНТЕГРИРОВАННОСТИ В ИНФРАСТРУКТУРУ КОМПАНИИ**

Из-за схожего принципа функционирования систем, различия в данной сфере минимальны. Главной особенностью в данном аспекте будет поддержка различных сред функционирования программ. Для KATA и TDS-Polygon основными средами анализа будут:

* веб трафик;
* почтовый трафик;
* файлы персональных компьютеров.

В данном случае явной отличительной чертой обладает система Athena, в которой также существует анализ мобильных устройств как на iOS, так и на Android. Данный факт сильно увеличивает возможности взаимодействия с системой, так как с ростом числа мобильных устройств среди пользователей различных систем повышаются риски атак на данные среды.

Дополнительным плюсом является доступность API системы и возможность использовать её, как платформу для модулей собственной разработки. Как один из примеров реализации, вместе с системой может поставляться Telegram-бот, с помощью которого можно проводить анализ файлов на вредоносность.

Кроме того, в системе предоставлен функционал эмулирования пользователя, а также анализа на основе статического исследования структуры файла. На основании данных, полученных на этапе статического исследования системой может приниматься самостоятельное решение по выбору среды и особенностей проведения исследования, а также вынести предварительный вердикт с помощью нейронной сети.

## **ОГРАНИЧЕНИЯ ПО ТЕХНИЧЕСКИМ ПАРАМЕТРАМ**

В открытом доступе открыта лишь информация ограничений по двум системам: Athena(рис. 1.3) и KATA(рис. 1.4).

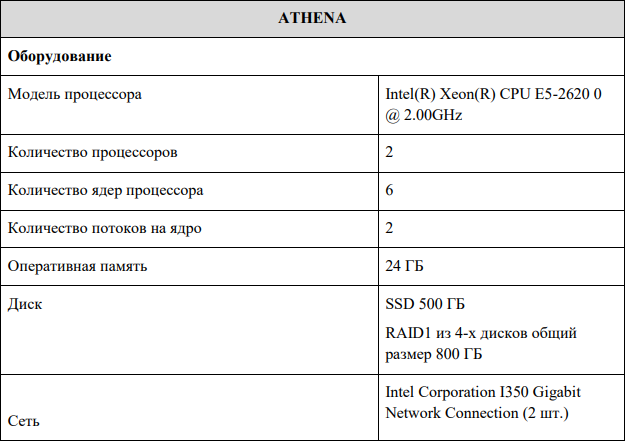


Рис. 1.3. Требования к оборудованию Athena

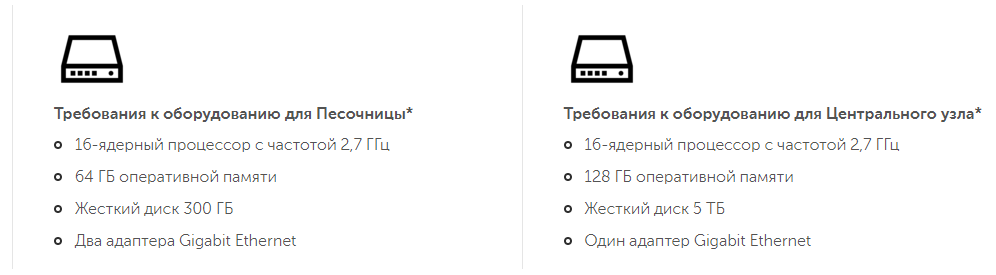


Рис. 1.4. Требования к оборудованию KATA

Можно заметить, что требования по техническим требованиям к процессору и оперативной памяти у системы Athena существенно ниже. Кроме того, в отличии от TDS в данных продуктах существует возможность более гибкого выбора комплектующих под запросы компаний.

## **ОСОБЕННОСТИ ИНТЕРФЕЙСА**

В сфере особенностей интерфейса можно выделить только доступность системы из разных сред взаимодействия.

Доступность системы можно оценить по показателю числа сред откуда можно без затруднений пользоваться системой. Во всех трех случаях, основной интерфейс взаимодействия с системой является веб-интерфейс. Из этого следует, что доступ к системам возможен из любых сред с поддержкой браузера. В данном случае можно выделить только адаптивность интерфейса системы Athena и мультиязычность. В продукте KATA отсутствует поддержка русского языка, а TDS имеет лишь мобильное приложение, требующее обязательную установку. Данный факт резко уменьшает удобство пользования системой. Также в системе Athena существует режим цветовой слепоты.

## **ИСПОЛЬЗУЕМЫЕ МЕТОДИКИ АНАЛИЗА СОБЫТИЙ**

Во всех представленных системах методики анализа достаточно приближены. В каждом из предложенных вариантов она определяется:

1. модулем сбора событий;
2. аналитиками (правилами, используемыми для классификации событий), задаваемыми экспертами;
3. правилами вынесения вердикта.

Модуль сбора событий собирает действия, выполняемые отслеживаемыми процессами на виртуальных машинах. Аналитики классифицируют собранную информацию. Правила вынесения вердикта определяют результат работы алгоритма на основании классифицированных событиях.

Основные различия в анализе были определены в собираемых данных и правилах анализа. Особенности реализации последнего определяют основные различия в результатах анализа.

Для реализации данного алгоритма могут применятся методы анализа Big Data, которые будут проставлять веса классам для определения вредоносности программного обеспечения.

Существуют следующие классы методы анализа Big Data [8]:

* методы класса Data Mining;
* краудсорсинг;
* смешение и интеграция данных;
* машинное обучение;
* искусственные нейронные сети, сетевой анализ, оптимизация, в том числе генетические алгоритмы;
* распознавание образов;
* прогнозная аналитика;
* имитационное моделирование;
* пространственный анализ (англ. Spatial analysis);
* статистический анализ, в качестве примеров методов приводятся A/B-тестирование и анализ временных рядов;
* визуализация аналитических данных.

Так как исследований и срабатываний аналитик достаточно мало относительно стандартных объемов анализа Big Data, для решения данной задачи в первую очередь могут быть полезны методы класса Data Mining из-за небольшой требовательности к объему данных и производительности системы.

Основным решением можно выбрать метод машинного обучения с учителем на основе линейной модели. Данный метод позволяет быстро и качественно обучить систему выставлять верные веса аналитикам и даст возможность редактировать их во время работы.

## **ВЫБОР ПЛАТФОРМЫ РЕАЛИЗАЦИИ И ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ**

В процессе анализа программных решений были выявлены следующие преимущества и закономерности позиционирования продуктов компаний на рынке.

TDS-Polygon описывается как комплексное решение, главным отличием которого является сервис, связанный с продуктом. В данном случае, владелец системы получает также все имеющиеся сведения защиты, которые поддерживаются в максимально актуальном состоянии.

Athena описывается как новая, абсолютно гибкая и во всех смыслах доступная система, способная к масштабированию и использованию в различных условиях. Также, благодаря модулю статического анализа, для работы с системой пользователю нет необходимости иметь большой опыт использования подобного решения. Кроме того, среда Athena полностью изолирована и не требует дополнительных подключений, либо других способов передачи данных во вне.

KATA – продукт в котором выделяется опыт работы компании «Лаборатория Касперского». Данная система завоевала многие призы, что определяет её уровень надежности и качества работы.

В предложенных условиях, как платформа реализации модуля динамического анализа была выбрана система Athena из-за найденных фундаментальных преимуществ.

Однако в данной системе модуль динамического анализа имеет ряд существенных недостатков, которые возможно устранить в рамках дипломной работы.

Главным из них является то, что данные исследований собираются в базе SQL и проверяются простой процедурой на T-SQL. Данные технологии показывают низкие показатели эффективности использования ресурсов для анализа Big Data что приводит к серьезным проблемам с производительностью. Для эффективной работы процедуры за приемлемое время необходимо индексировать поля анализа. Кроме увеличения объема физической памяти, требуемой для хранения, из-за этого также повышается объем оперативной памяти, задействованной в ходе процедуры.

Решение проблемы в существующих условиях невозможно, так как удаление индексов ведет к кратному увеличению нагрузки на процессор, а он в свою очередь становится узким горлышком системы.

Таким образом, можно отметить следующие недостатки в системе:

1. отсутствует возможность горизонтального масштабирования;
2. отсутствует анализ нагрузки процессов на систему;
3. упрощена процедура вынесения вердикта;
4. высокая сложность функционального расширения;
5. огромный объем данных хранимых пустых значений и индексов;
6. большой объем ложных срабатываний и меньший объем истинных (из-за совокупности проблем).

Как показатели, необходимые к улучшению, можно выделить:

1. количество истинных срабатываний;
2. количество ложных срабатываний;
3. потребление ресурсов системы;
4. производительность системы.

При минимальных требованиях в сутки в среднем проверяется 1000 файлов. Таким образом, основой данных для проверки данных показателей будет:

1. количество истинных срабатываний на 1000 файлов;
2. количество ложных срабатываний на 1000 файлов;

3. количество потребляемых ресурсов на анализ одного файла (улучшение среднего показателя нагрузки на ЦП, диск и оперативную память);

4. скорость обработки одного файла.

Для реализации были предложены следующие технологии:

* База данных MongoDB для хранения событий;
* База данных MSSQL Server для хранения общих системных данных исследования;
* Веб-сервис на .net core;
* Протокол связи HTTP/HTTPS и формат данных JSON;
* Отображение результата работы на HTML генерируемый сервером на .net core;
* Шифрование данных на основе протоколов TLS и SSL;
* Сервер размещения – IIS/автономный режим на основе Kestrel;
* Интерфейсные компоненты DevExtreme;
* Фреймворк анализа данных Accord.NET.

База MongoDB зарекомендовала себя как мощный инструмент для анализа Big Data. Являясь NoSql базой, основанной на документах, существуют большие преимущества в объеме хранимой разнородной информации и скорости вставки, благодаря отсутствию валидации данных.

В существующей системе основная информация исследования хранится в MSSQL, но при этом её объемы в рамках одной процедуры минимальны, а внесения изменений не приведут к ощутимым изменениям. Таким образом, было решено частично оставить представление в этой базе.

Фреймворк .net core позволит получить все преимущества языка c#, платформы .net, а кроме того он полностью совместим с большинством новейших технологий. Его реализация в виде веб-сервиса позволит внедрить его в существующую архитектуру с возможностью гибкой настройки и масштабируемости.

Протокол связи HTTP имеет очевидные преимущества для связи различных веб модулей в общую систему из-за стандартности данного подхода.

Протокол связи JSON рекомендован к применению в RFC8259 [3] для обмена структурированными данными. Появившись в 1999 году на основе подмножества языка стандарта ECMA-262, все же считается независимым от него и может использоваться в том числе в c#. Плюсами данного формата является:

1. легковесность;
2. основанность на тексте;
3. независимость от языка.

HTML является стандартизованным языком разметки документов. Определяется как приложение стандартного обобщенного языка по стандарту ISO 8879. Обязательно используется как инструмент формирования страниц для вывода в сети интернет.

Интерфейсные компоненты DevExtreme позволят ускорить процесс разработки. Данные компоненты реализованы в различных версиях и дают широкие возможности по настройки своих компонентов. Используя код на c# в разметке Razor, связку html+jQuery или различные фреймворки (AngularJS, Angular, Vue, React) можно добиться быстрых результатов при затрате минимума времени. В данном наборе присутствует 14 основных компонентов с возможностью широкой кастомизации, что покрывает большую часть запросов на индивидуализацию интерфейса.

Accord.Net является самым популярным фреймворком анализа данных для c#. Данный фреймворк легок в использовании и гибок. В нем присутствуют все необходимые инструменты для текущей разработки.

Использование данных технологий поможет преодолеть некоторые недостатки системы в производительности, обеспечить расширяемую и масштабируемую архитектуру модуля, а также создать удобный и настраиваемый интерфейс.

## **ОСНОВНЫЕ НОРМАТИВНЫЕ ДОКУМЕНТЫ, ПРИМЕНЯЕМЫЕ ПРИ РАЗРАБОТКЕ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ**

**ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств.**

В данном стандарте описывается структура процессов жизненного цикла. Он будет использоваться на протяжении разработки и сопровождения модуля анализа для повышения качества программного продукта.

**ГОСТ Р 51275-2006. Защита информации. Объект информатизации. Факторы, воздействующие на информацию. Общие положения.**

В данном стандарте описываются основные положения, которые необходимо учесть для безопасной работы с информацией. Он будет использоваться для обеспечения безопасности данных исследований.

**ГОСТ Р ИСО/МЭК 27001-2006. Информационные технологии. Методы безопасности. Система управления безопасностью информации. Требования.**

Данный стандарт будет применяться для определения требований по разработке, функционированию, внедрению, анализу и поддержки модуля.

# **ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ ПРОЦЕССА ИССЛЕДОВАНИЯ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ МЕХАНИЗМА ДИНАМИЧЕСКОГО АНАЛИЗА**

## **ОПРЕДЕЛЕНИЕ ПОДХОДОВ К РЕШЕНИЮ ПОСТАВЛЕННОЙ ЗАДАЧИ**

Системный подход – в основе этой методологии объект рассматривается как целостная система. С его помощью производится исследование объекта, чтобы раскрыть его основные взаимосвязанные механизмы, выявить многообразие типов связей. Все сложные системы необходимо рассматривать в двух аспектах: в статическом и динамическом состояниях. С помощью предметного анализа определяется то, из каких компонентов состоит система, а также какие связи эти компоненты имеют между собой [14].

Архитектурный подход – обеспечивает подход к разработке автоматизированных систем, благодаря которому основное внимание уделяется созданию и постоянному развитию архитектуры предприятия. Базой для данного подхода является целостное многоаспектное представление о разрабатываемой системе и подсистемах. Основная идея данного подхода заключается в необходимости использования целостного подхода к описанию производственных систем, чтобы обеспечить высокой ценностью и гибкостью сферу бизнеса. В данном подходе рассматривается фундаментальная организация системы, в которой содержится совокупность компонент, у которых определены связи с внешней средой, а также принципы, которыми необходимо руководствоваться при разработке [8].

Процессный подход – одна из концепций управления, которая окончательно сформировалась в 80-х годах прошлого века. В соответствии с этой концепцией вся деятельность организации рассматривается как набор процессов. Для того чтобы управлять, необходимо управлять процессами [13].

Менеджмент качества – это деятельность, направленная на планирование, целеполагание, обеспечение, контроль, улучшение качества продукции и услуг, производимых конкретным предприятием [11].

Функциональное моделирование – процесс моделирования функций, которые выполняются в рассматриваемой информационной системой. При этом моделировании создается описательное структурированное графическое изображение, на котором показывается: что, кто и как делает при функционировании объекта [18].

Методология IDEF0 предписывает построение иерархической системы диаграмм – единичных описаний фрагментов системы. Сначала проводится описание системы в целом и ее взаимодействия с окружающим миром (контекстная диаграмма), после чего проводится функциональная декомпозиция – система разбивается на подсистемы, и каждая подсистема описывается отдельно (диаграммы декомпозиции). Затем каждая подсистема разбивается на более мелкие и так далее до достижения нужной степени подробности.

Основой функциональной модели составляют структурированные описания различных функций и информационных связей между элементами образовательной системы, с помощью которых можно:

* Представить и описать полный спектр процессов кого-либо учреждения на любом уровне детализации.
* Обеспечить точное и лаконичное описание моделируемых объектов.
* Оптимизировать взаимодействие и взаимопонимание между специалистами, занятыми анализом и проектированием различных процессов.

Физическое проектирование базы данных – создание схемы базы данных для конкретной СУБД.

Разработка через тестирование — техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам [14].

## **ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ МОДЕЛЕЙ МОДУЛЯ ИССЛЕДОВАНИЯ ВРЕДОНОСНОГО ПО НА ОСНОВЕ МЕХАНИЗМА ДИНАМИЧЕСКОГО АНАЛИЗА**

Словарь функциональной модели по разработке функциональных моделей модуля исследования вредоносного анализа на основе механизма динамического анализа состоит из блоков (таблица 2.1) и списка информационных потоков (таблица 2.2).

Внешними входными информационными потоками для модели функциональной модели процесса являются:

1. Выполнение исследуемой программы, которое генерирует события;
2. Модули динамического анализа, выполняющие анализ;
3. Параметры выполнения.

Внешними выходными информационными потоками для модели является:

1. Вердикт;
2. Связи событие-аналитики;
3. Дерево процессов.

Внешними управляющими потоками для модели являются:

1. Аналитики;
2. Регламент работы модуля;
3. Правила вынесения вердикта.

Основным механизмом для модели является модуль управления.

На рисунках 2.1-2.5 представлены диаграммы функциональной модели.

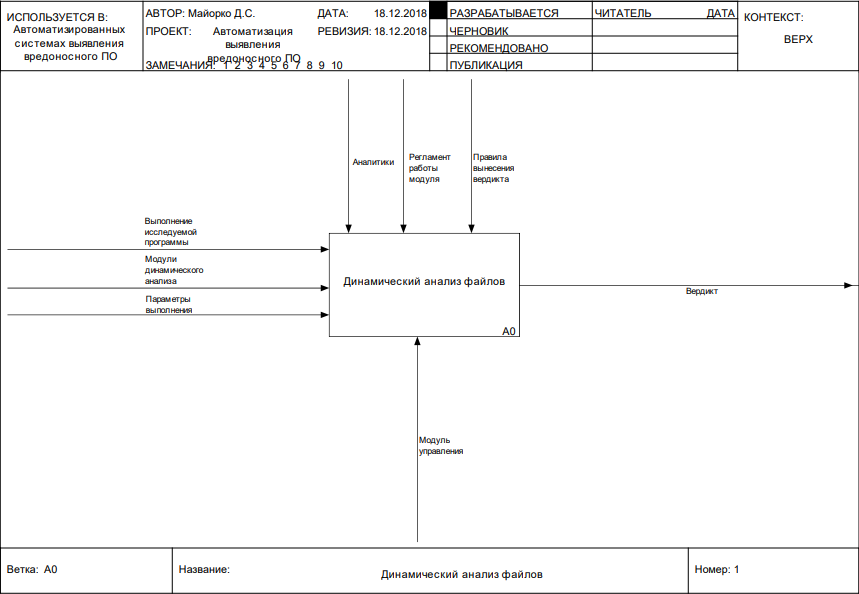


Рис. 2.1. Контекстная диаграмма

Процедуру динамического анализа можно разбить на пять функциональных блоков (рис. 2.2):

1. Запись результатов исследования;
2. Управление модулями анализа;
3. Частичный (пакетный) анализ;
4. Целостный (итоговый) анализ;
5. Вынесение вердикта.

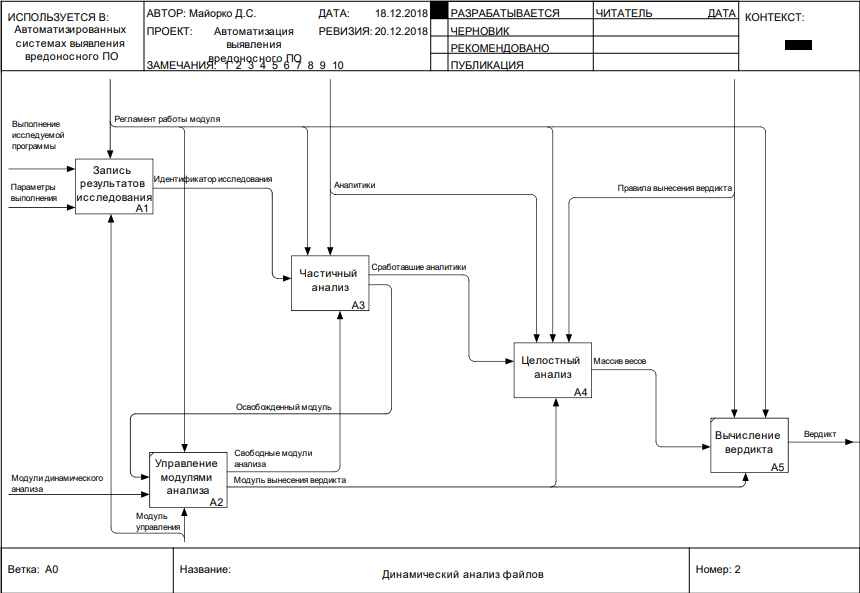


Рис. 2.2. Диаграмма процессов динамического анализа файлов

Блок частичного анализа состоит из пяти этапов (рис. 2.3):

1. Получение информации об исследовании;
2. Обработка аналитик;
3. Применение аналитик к событиям;
4. Анализ сложных аналитик;
5. Передача аналитик модулю целостного анализа.

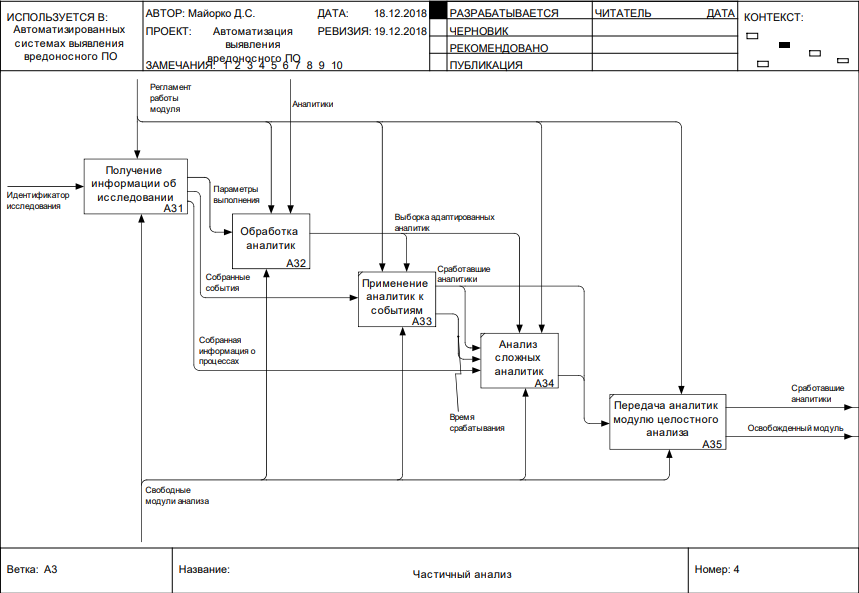


Рис. 2.3. Диаграмма процессов блока "Частичный анализ"

Блок обработки аналитик состоит из трех этапов (рис. 2.4):

1. Обработка параметров выполнения;
2. Фильтрация аналитик;
3. Адаптация аналитик.

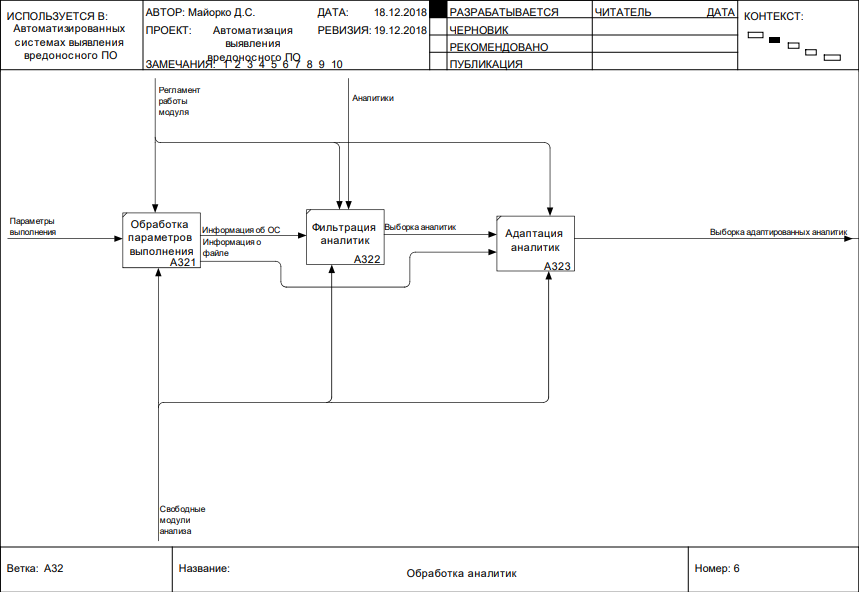


Рис. 2.4. Диаграмма процессов блока "Обработка аналитик"

Блок целостного анализа состоит из трех этапов (рис. 2.5):

1. Выделения групп аналитик;
2. Создания иерархий срабатываний;
3. Выделения используемых весов.

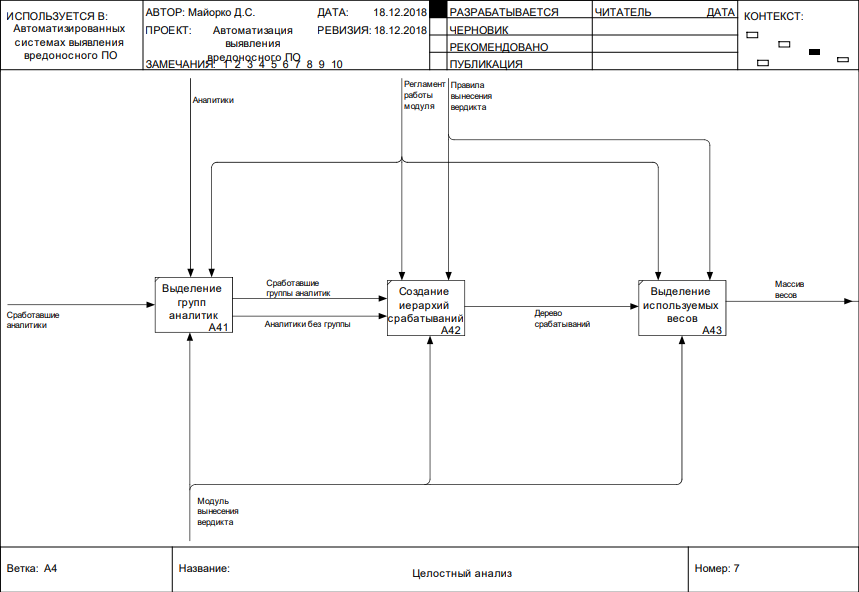


Рис. 2.5. Диаграмма блока "Целостный анализ"

Таблица 2.1.

Список функциональных блоков

| **Номер** | **Наименование** | **Определение** |
| --- | --- | --- |
| A0 | Динамический анализ файлов | Система производит анализ исследуемого файла на предмет вредоносных событий. |
| A1 | Запись результата исследования | Запись результатов проведения исследования: записанных событий, генерируемых файлом. |
| A2 | Управление модулями анализа | На данном этапе происходит процесс управление ресурсами модулей: оставляются запросы на проведение анализа пакета, целостного анализа, либо вынесения вердикта. |
| A3 | Частичный анализ | Происходит анализ собранного пакета событий |
| A4 | Целостный анализ | Комплексный анализ исследования по выявленным аналитикам |
| А5 | Вычисление вердикта | Вычисление вердикта с учетом используемых правил |
| A31 | Получение информации об исследовании | Получение основной информации о запуске исследования (длительность, архитектура проведения, дополнительные параметры, имя исполняемого файла) |
| A32 | Обработка аналитик | Изменение аналитик под собранные данные требуемого формата |
| A33 | Применение аналитик к событиям | Выделение подошедших аналитик |
| A34 | Анализ сложных аналитик | Обработка собранных данных для анализа процессов, файловой структуры, загрузки системы и прочего. |
| A35 | Передача аналитик модулю целостного анализа | Отправка собранных данных в модуль целостного анализа |
| А41 | Выделение групп аналитик | Выделение групп сработавших аналитик |
| А42 | Создание иерархий срабатываний | Создание иерархии сработавших аналитик и групп |
| А43 | Выделение используемых весов | Выделение весов на основе правил анализа |
| А321 | Обработка параметров выполнения | Обработка параметров выполнения для подготовки аналитик |
| А322 | Фильтрация аналитик | Фильтрация аналитик по параметрам для подбора необходимых аналитик с повышением приоритета вредоносных аналитик |
| А323 | Адаптация аналитик | Подготовка аналитик к выполнению с выбранными параметрами |

Таблица 2.2.

Список информационных потоков

| **Наименование** | **Тип** | **Определение** |
| --- | --- | --- |
| Выполнение исследуемой программы | Input | Все персональные данные обучающегося |
| Модули динамического анализа | Input | Набор программных решений для функционирования системы |
| Параметры выполнения исследования | Input | Лицо, которое занимается процессом оформления материальной поддержки |
| Аналитики | Control | Лицо, занимающееся реализацией процессов в профкоме |
| Регламент работы модуля | Control | Алгоритмы, работающие в профкоме, которые регулируют процессы оформления материальной поддержки и расчет выплат |
| Правила вынесения вердикта | Control | Федеральный закон РФ, обеспечивающий безопасность персональных данных пользователя в сети |
| Модуль управления | Mechanism | Инструкции по заполнению различных форм в системе |
| Вердикт | Output | Инструкции по заполнению различных форм в системе |

Диаграммы потоков данных (Data Flow Diagrams) являются основным средством моделирования функциональных требований к проектируемой системе. Требования представляются в виде иерархии процессов, связанных потоками данных. Диаграммы потоков данных показывают, как каждый процесс преобразует свои входные данные в выходные, и выявляют отношения между этими процессами.

Наименования таблиц базы данных информационной системы приводятся в формате «БД: Таблица». На рисунках 2.6-2.7 представлены диаграммы потоков данных функциональных блоков системы.

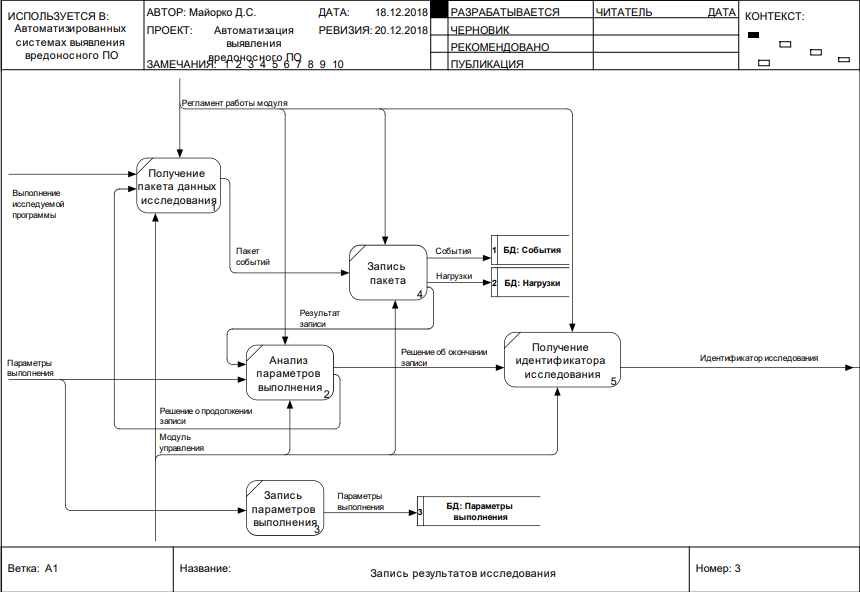


Рис. 2.6. Диаграмма потоков данных блока "Запись результатов исследования"

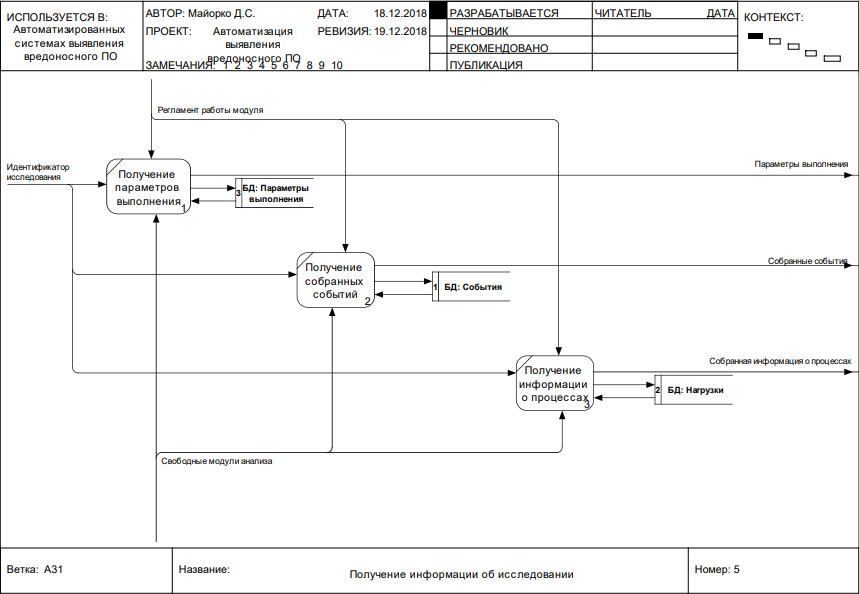


Рис. 2.7. Диаграмма потоков данных блока "Получение информации об исследовании"

Модель отображает физическую структуру базы данных: таблицы, которые будут в ней использоваться, связи между ними, а также список атрибутов каждой таблицы.

Для функционирования системы в базе данные данных присутствуют следующие таблицы и их атрибуты:

1. Таблица «Параметры выполнения» содержит информацию о параметрах выполнения исследования.

Таблица 2.3.

Структура таблицы «Параметры выполнения»

| **Столбец** | **Тип данных** | **Нуль?** | **Ключ** |
| --- | --- | --- | --- |
| IDИсследования | Int | Not null | Первичный |
| Длительность | Int | Not null |  |
| АрхитектураПроведения | Int | Not null |  |
| НазваниеПакета | Varchar |  |  |
| НазваниеИсполняемогоФайла | Varchar | Not null |  |
| ДополнительныеПараметрыИсследования | Varchar |  |  |

2. Таблица «События» содержит информацию о полученных событиях исследования. Так как физическое проектирование производится на основе базы MongoDb, для таблиц «События» и «Нагрузки» выделены только поля обязательные для функционирования. Остальные поля добавляются в зависимости от присылаемых данных виртуальной машиной. Модуль анализа формирует аналитики вне зависимости от посланных данных.

Таблица 2.4.

Структура таблицы «События»

| **Столбец** | **Тип данных** |
| --- | --- |
| Номер | Int |
| IDИсследования | Int |
| Модуль | Int |
| Событие | Int |
| Pid | Int |
| ОписаниеАналитик | string[] |

3. Таблица «Нагрузки» содержит информацию о нагрузках во время проведения исследования.

Таблица 2.5.

Структура таблицы «Нагрузки»

| **Столбец** | **Тип данных** |
| --- | --- |
| IDИсследования | Int |
| ВремяПолучения | Int |
| Процессы | Процесс[] |
| Процесс.Id | Int |
| Процесс.CPU | Double |
| Процесс.Память | Double |
| Процесс.GPU | Double |
| Процесс.Диск | Double |

Внешними ключами для всех таблиц будет IDИсследования. Связываться данные таблицы будут через прослойку в самом модуле динамического анализа. Таким образом схема данных будет выглядеть соответственно рисунку 2.8.

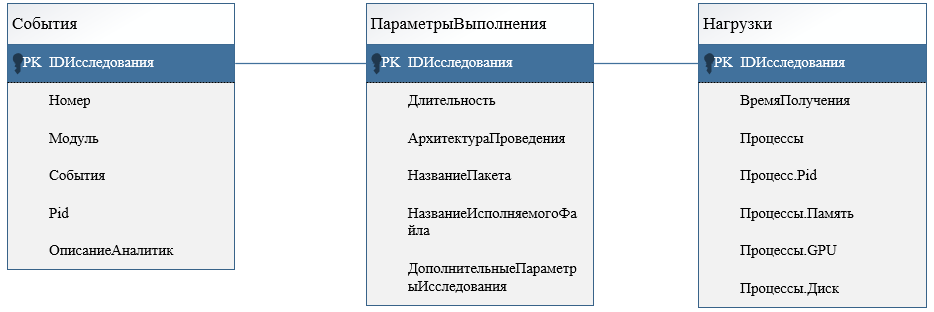


Рис. 2.8. Схема данных

# **РАЗРАБОТКА МОДУЛЯ ВЫЯВЛЕНИЯ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ МЕХАНИЗМА ДИНАМИЧЕСКОГО АНАЛИЗА**

## **ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ РЕАЛИЗАЦИИ.**

Для написания сервиса анализа используется фреймворк .Net Core 2.2. NET Core является кроссплатформенным (Windows, Mac, Linux) аналогом .NET Framework с открытым исходным кодом. Он содержит среду CoreCLR — кросс-платформенную реализацию CLR, в которую входит JIT-компилятор — RyuJIT. Также .NET Core включает в себя CoreFX — набор библиотек, функциональность которых частично совпадает с BCL: API .NET Core предоставляет подмножество API .NET Framework, но также и предоставляет ряд собственных API. .NET Core был создан в 2014 году.

База данных для сбора событий – MongoDB. Это документоориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных.

Для хранения общих данных исследования используется MSSQL Server. Данная система является системой управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия.

## **РЕАЛИЗАЦИЯ МОДУЛЯ ВЫЯВЛЕНИЯ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ МЕХАНИЗМА ДИНАМИЧЕСКОГО АНАЛИЗА**

Разработанный модуль выявления вредоносного программного обеспечения на основе механизма динамического анализа содержит множество методологических новшеств, описанных ниже.

1. Введено новое понятие в систему — аналитическая группа. Данная сущность позволяет определять группы аналитик и выставлять веса не только конкретным аналитикам, но и их множеству.

2. Введено новое понятие — иерархия аналитик. Создание иерархии аналитик (определение родительских и дочерних аналитик) позволяет более точно определить несущественные аналитики и потенциально связанные для корректной классификации.

3. Введено новое понятие — нагрузки. Анализ нагрузок на систему позволяет более эффективно отлавливать новые классы вирусов, такие как майнеры и шифраторы.

4. Добавлен новый алгоритм выставления вердиктов на основе регрессионного анализа. Производится оценка весов классификаций, выполненных с помощью аналитик благодаря чему:

* снижается нагрузка на систему;
* повышается скорость обнаружения вредоносного программного обеспечения;
* снижается число затрачиваемых ресурсов на анализ;
* снижается число ложных срабатываний;
* повышается число истинных срабатываний.

5. Добавлена возможность анализа последовательностей выпадения событий.

Также, было добавлено редактирование весов аналитик на основе машинного обучения линейной регрессии.

Для метода оценки был выбрана функция наименьшей абсолютной ошибки из-за уменьшенной чувствительности к выбросам, которые могут быть на большой выборке исследований.

Методом получения весов был выбран стохастический градиентный спуск для оптимизации времени обучения. При сравнении градиентного спуска и стохастического градиентного спуска можно увидеть следующие преимущества:

* Быстрее выполняется один шаг;
* Не требуется хранение выборки в памяти;
* Подходит для обучения во время выполнения.

Размер шага поставлен в обратную зависимость от количества шагов для решения проблемы неправильного выбора шага инициации. Реализация данной задачи была выполнена с помощью фреймворка Accord.NET.

Обучающая выборка состояла из десяти тысяч файлов при использовании четырехсот аналитик. В процессе обучения выборка была разделена по пять тысяч файлов: обучающую и проверяющую. В результате получены веса срабатываний, используемые для вынесения вердиктов. Вероятность вынесения верного вердикта на тестирующей выборке 96%, что означает повышение точности анализа. В сочетании с другими видами и алгоритмами анализа данный результат можно считать приемлемым.

Кроме методологических изменений, модуль также содержит новые функции:

1. Производится построение дерева процессов для анализа и визуализации нагрузки на систему;
2. Производится архивирование отработанных исследований для сокращения потребляемого места;
3. Был реализован модуль конвертации аналитик, способный переводить их реализацию под большинство баз данных с минимальными изменениями;
4. Реализована возможность горизонтального масштабирования и параллельной обработки исследований, благодаря перевода системы на режим пакетной обработки данных;
5. Реализована возможность конфигурации модуля для определения параметров выполнения;
6. Реализована возможность автоиндексации полей, производительность поиска по которым становится проблемой для системы;
7. Реализована платформонезависимость модуля;
8. Реализована независимость модуля от конкретной структуры присылаемых данных;
9. Реализован отложенный анализ, при котором вердикт выдается сразу при получении достаточного уровня, но события полностью не проанализировались.

Необходимо заметить, что весь реализованный функционал разрабатывался с применением техники разработки через тестирование. Результатом данной особенности является стопроцентное покрытие кода тестами.

Также, при разработке модуля были созданы и дополнены несколько отображающих страниц:

1. На странице создания исследования (рис. 3.1) можно дать задачу на проведение динамического исследования файла. На ней можно выбрать предварительно загруженные файлы, мобильные приложения, режим запуска исследования (ручной или по созданному сценарию), продолжительность исследования, среду исполнения и дополнительные параметры исследования (скрипты пользователей, команды выполнения, автоматическое тестирование).

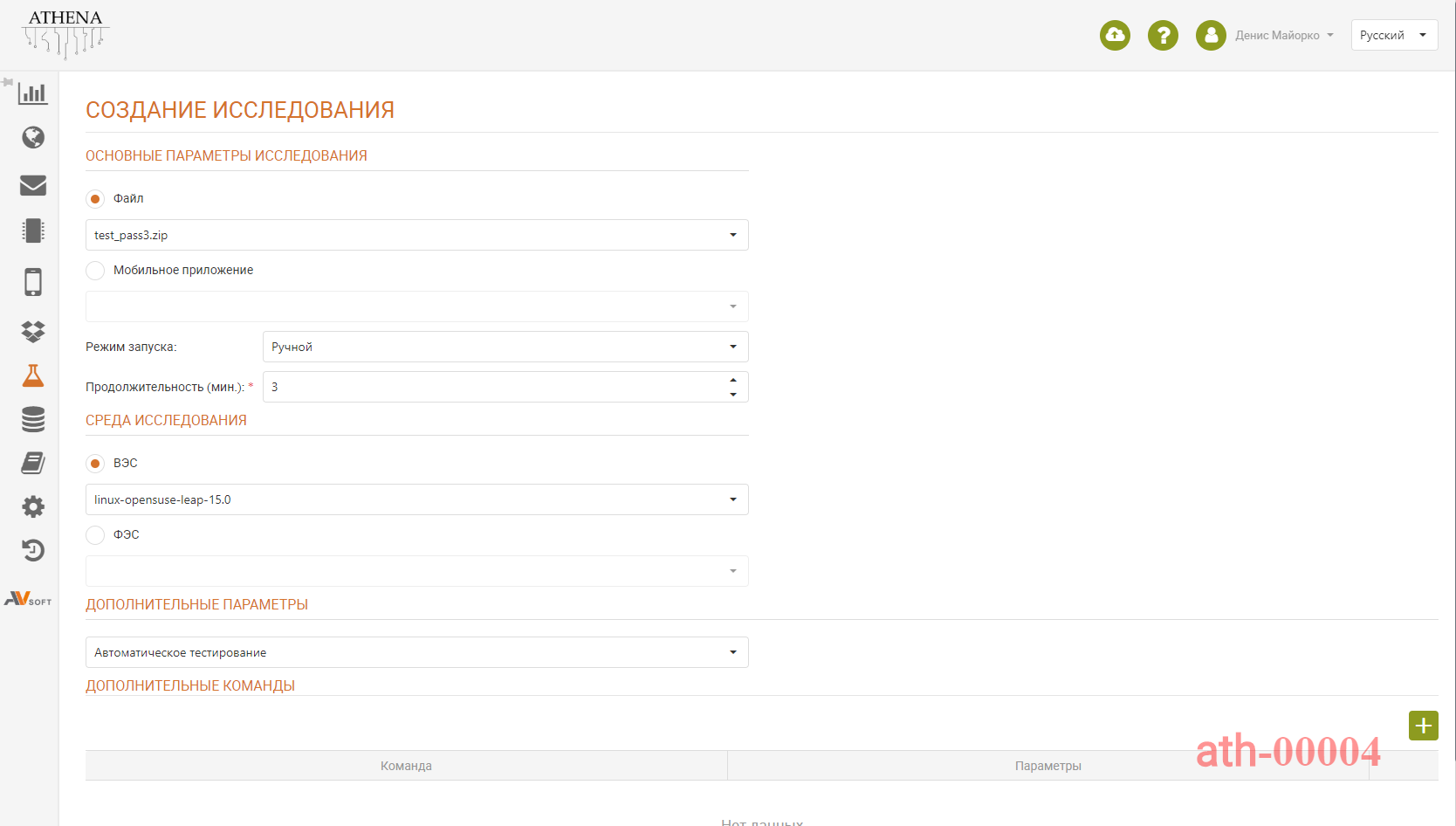


Рис. 3.1. Страница создания исследования

Дополнительные параметры (рис. 3.2) представляют собой задания, дающиеся на выполнения виртуальной машине в определенный момент. В данный момент доступен обширный список команд для мобильных устройств.

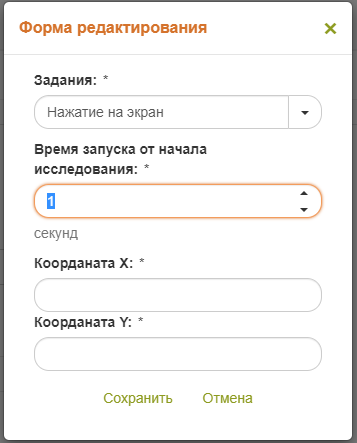


Рис. 3.2. Добавление дополнительных параметров исследования

2. Страница раздела динамических исследований (рис. 3.3) служит для отображения списка созданных исследований и их результатов.

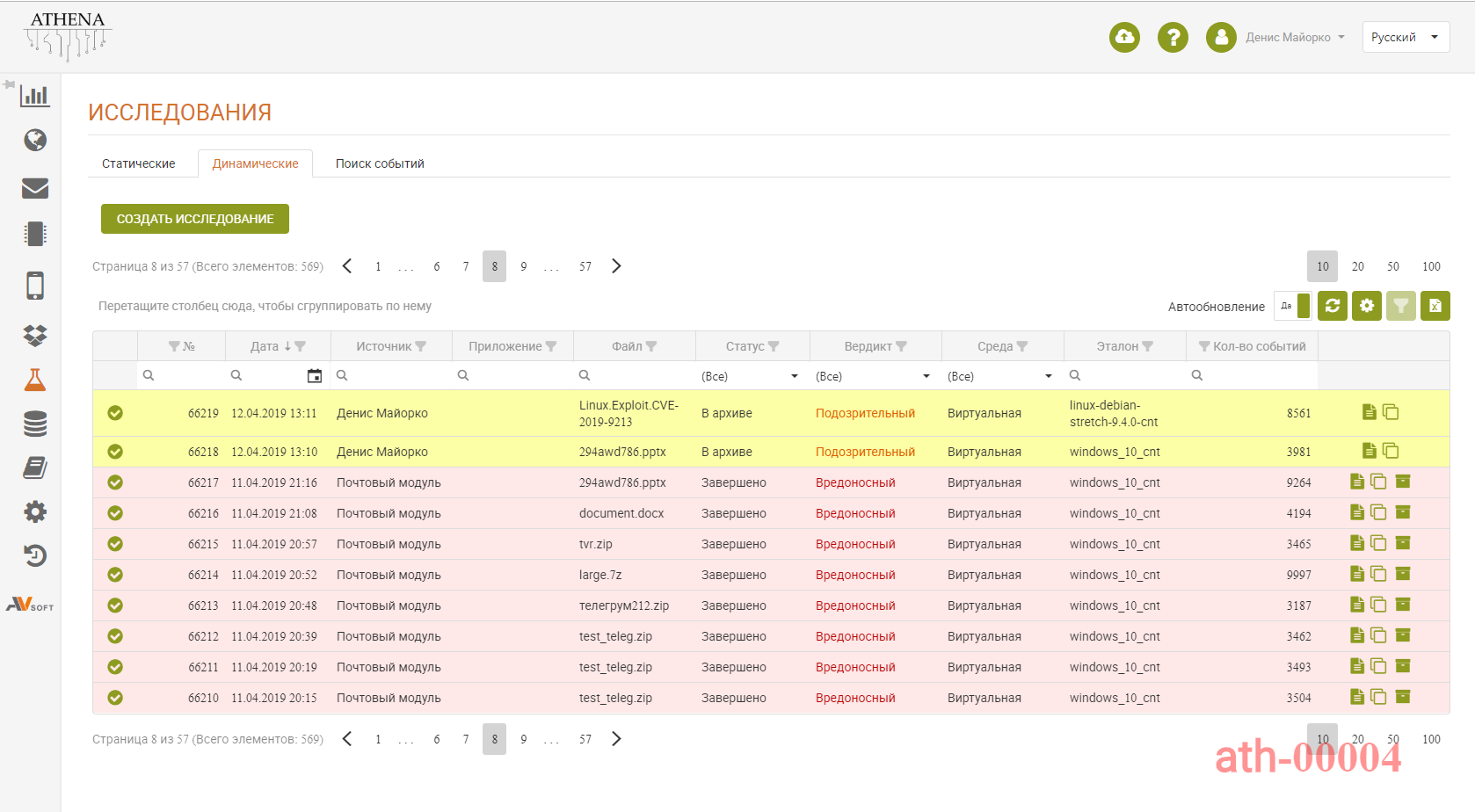


Рис. 3.3. Страница списка исследований

3. На странице отчета по исследованию (рис. 3.4-3.7) можно увидеть результат работы модуля:

* Список загруженных событий;
* Найденные аналитики;
* Найденные группы аналитик;
* Среднюю и пиковую нагрузку на процессор;
* Среднюю и пиковую нагрузку на диск;
* Среднюю и пиковую нагрузку на видеокарту по выполняемому файлу или каждому процессу в отдельности.

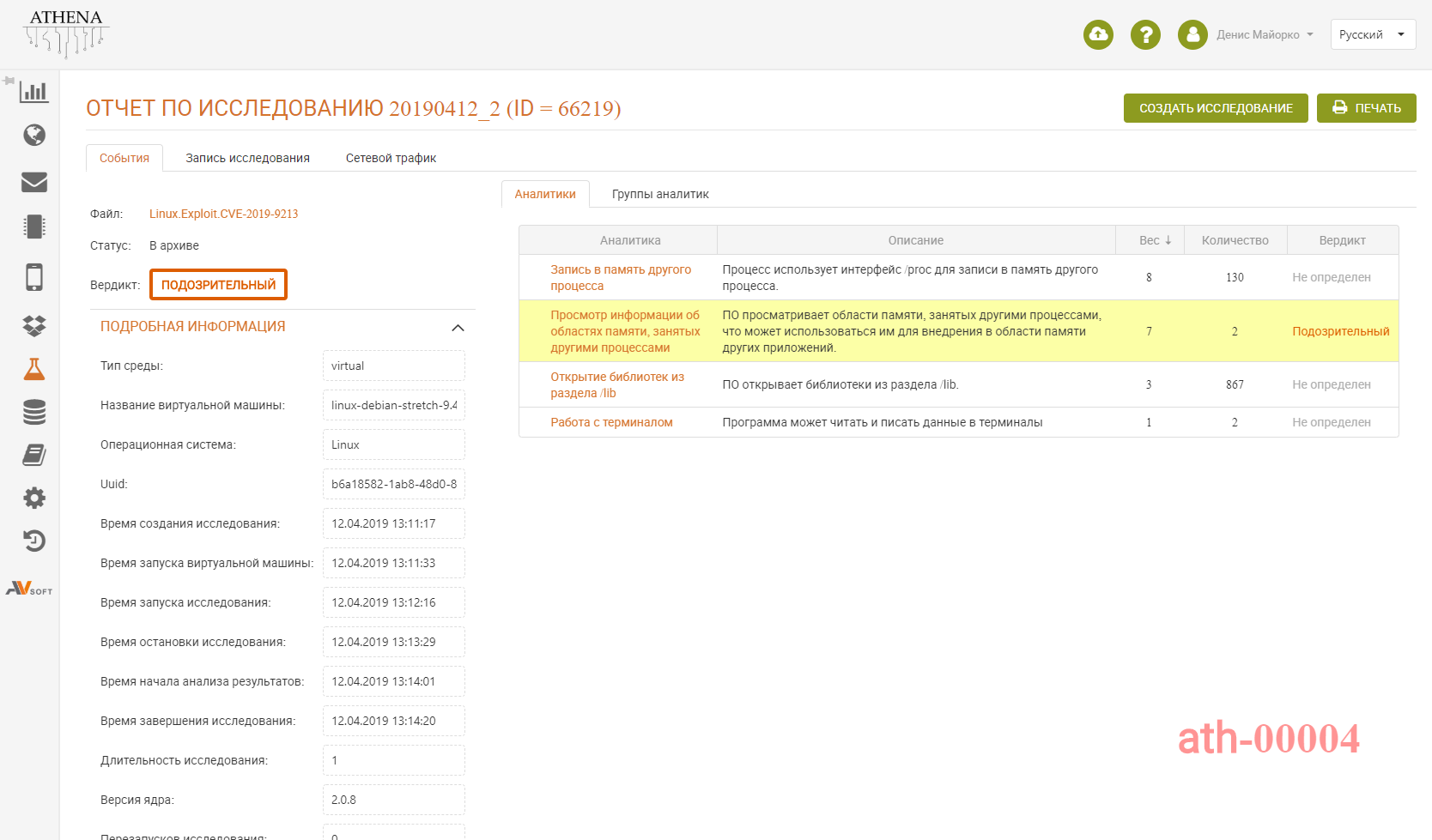


Рис. 3.4.Основная информация и аналитики страницы отчета по исследованию

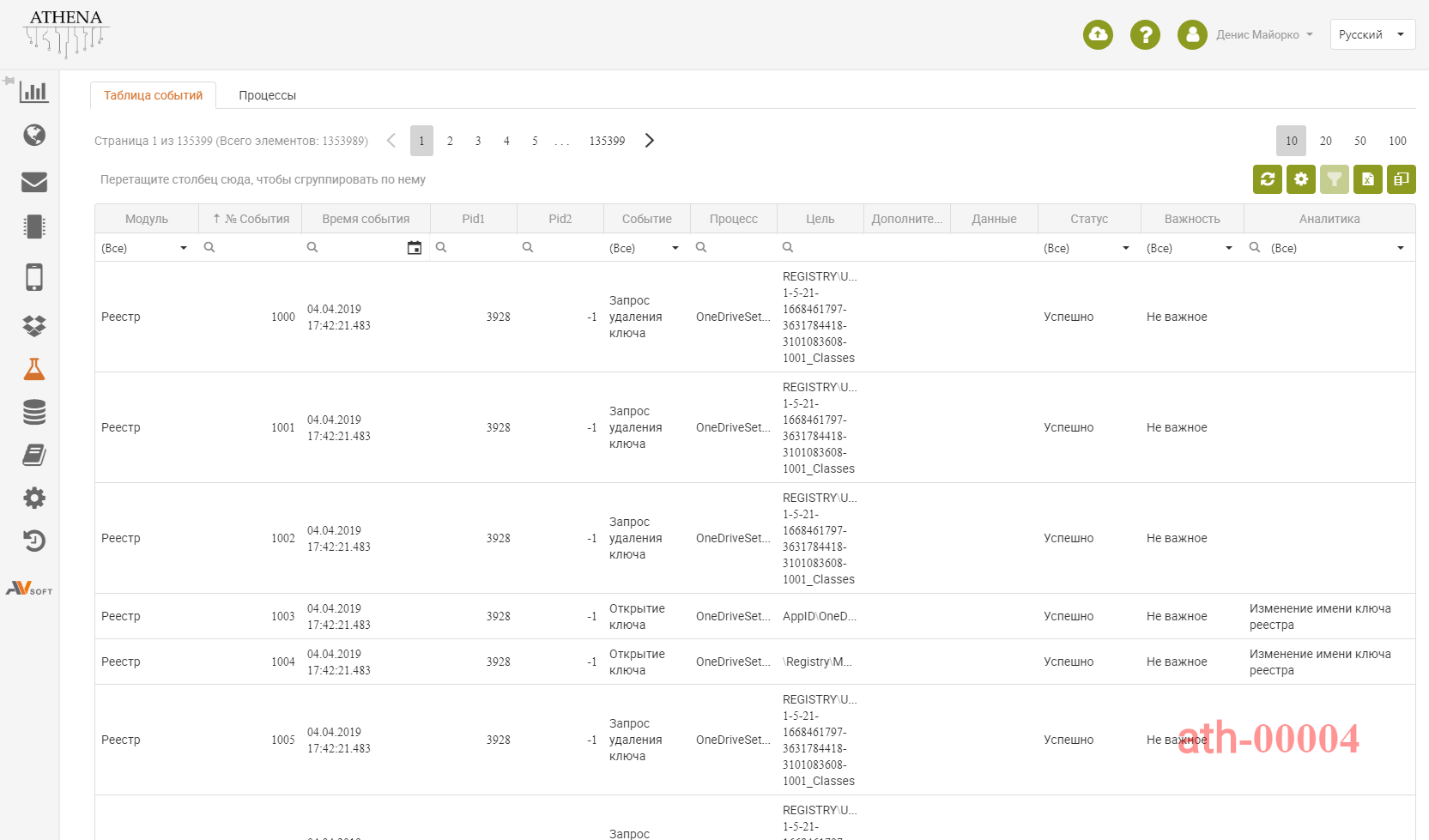


Рис. 3.5. Таблица событий страницы отчета по исследованию

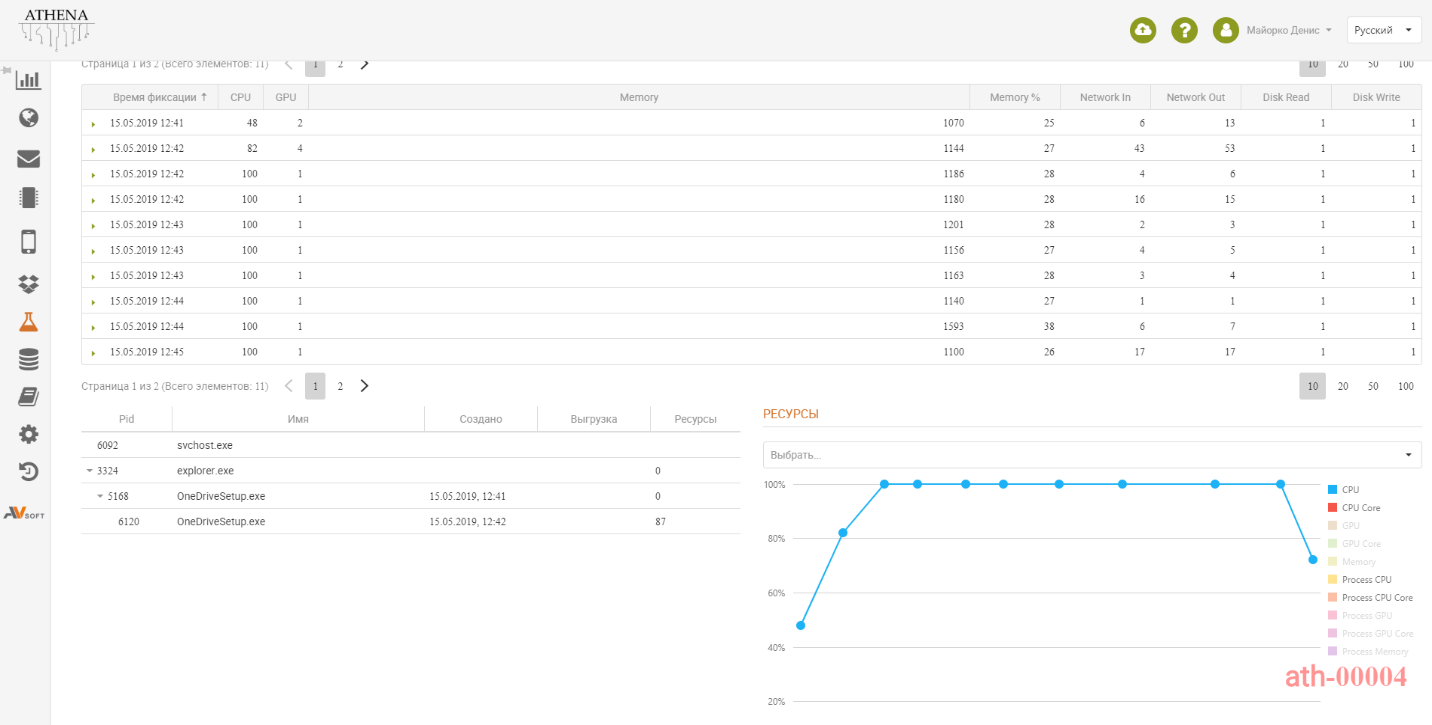


Рис. 3.6. Проанализированные процессы страницы отчета по исследованию

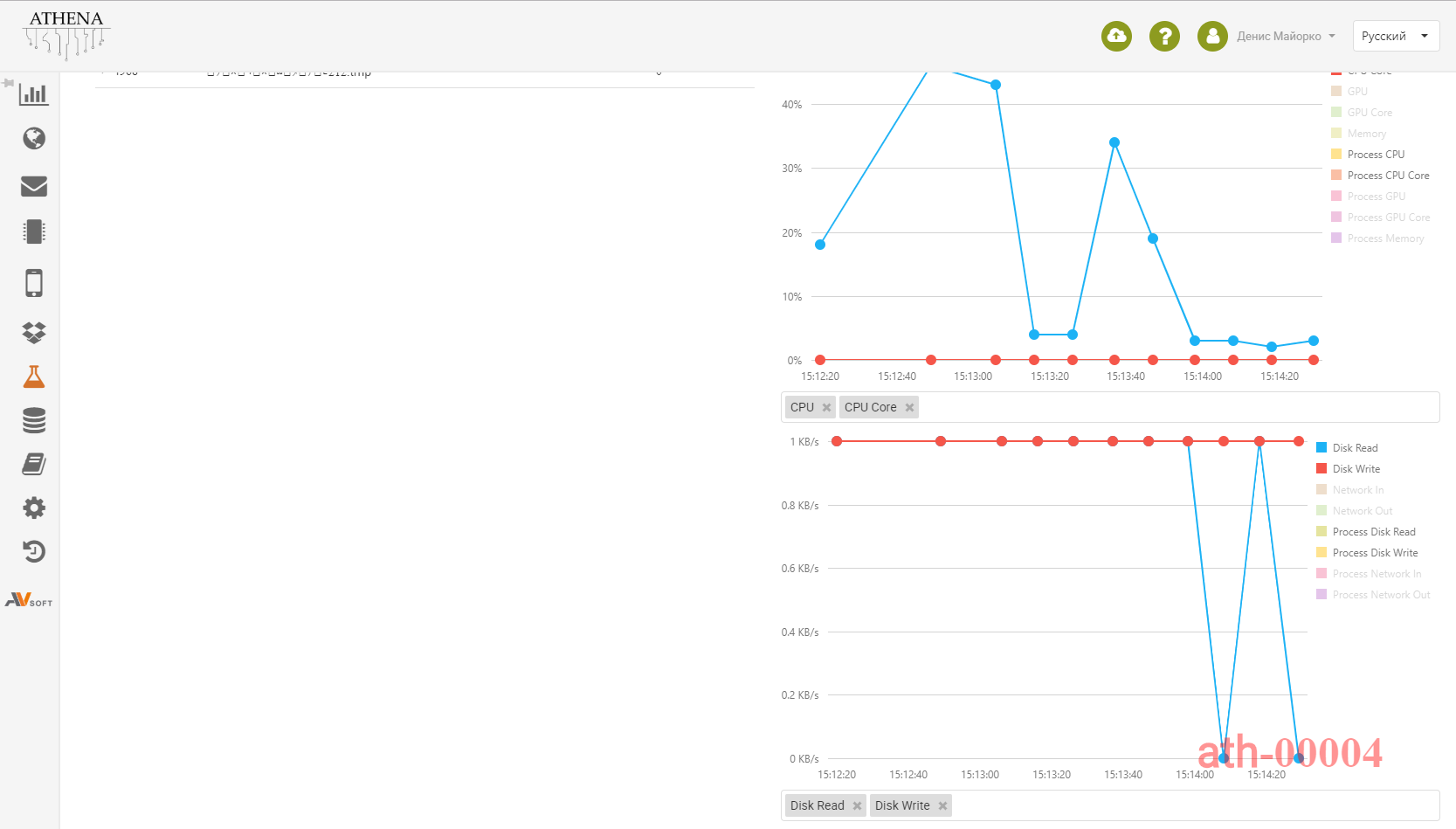


Рис. 3.7. Диаграммы процессов страницы отчета по исследованию

4. Страница формирования печатного отчета (рис. 3.8) позволяет сформировать отчет по данным исследования, пригодный для печати. Есть возможность выделения, группировки и форматирования всех полей исследования, которые необходимо вывести.

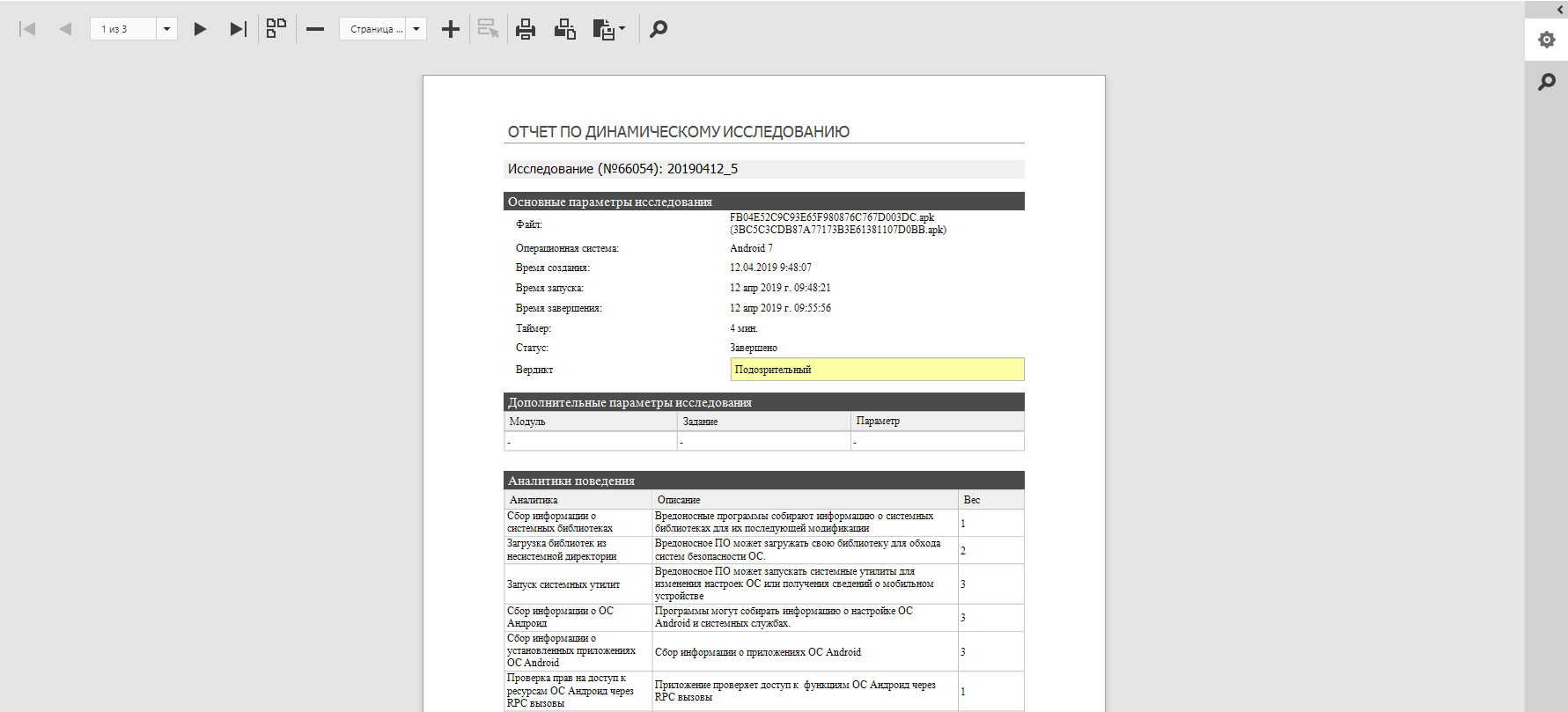


Рис. 3.8. Страница печатного отчета

5. Страница аналитик (рис. 3.9), отображающая список созданных аналитик, статистику проявления, проявление в безопасных, подозрительных и вредоносных исследованиях.

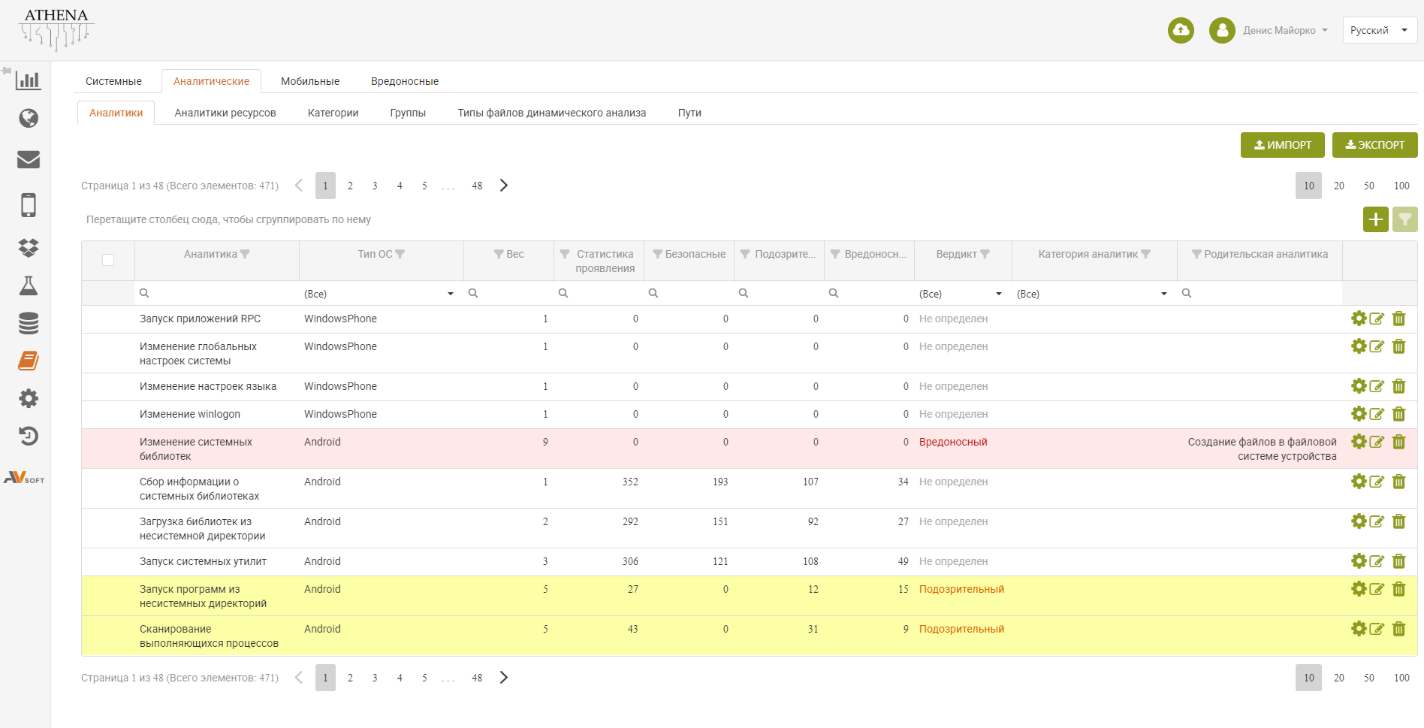


Рис. 3.9. Страница аналитик

6. Страница аналитических групп (рис. 3.10), содержащая в себе описание и состав групп аналитик.

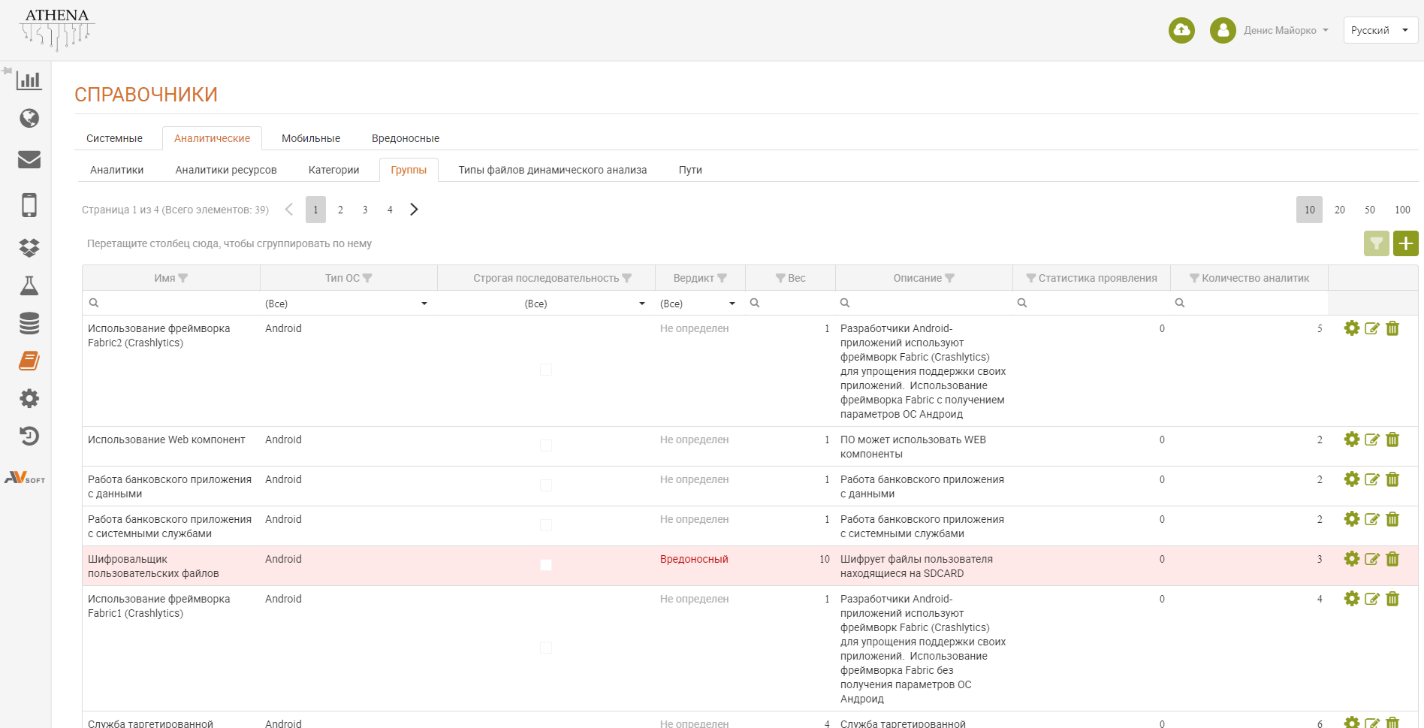


Рис. 3.10. Страница аналитических групп

7. Страница создания аналитик (рис. 3.11), содержащая поля для редактирования аналитики.

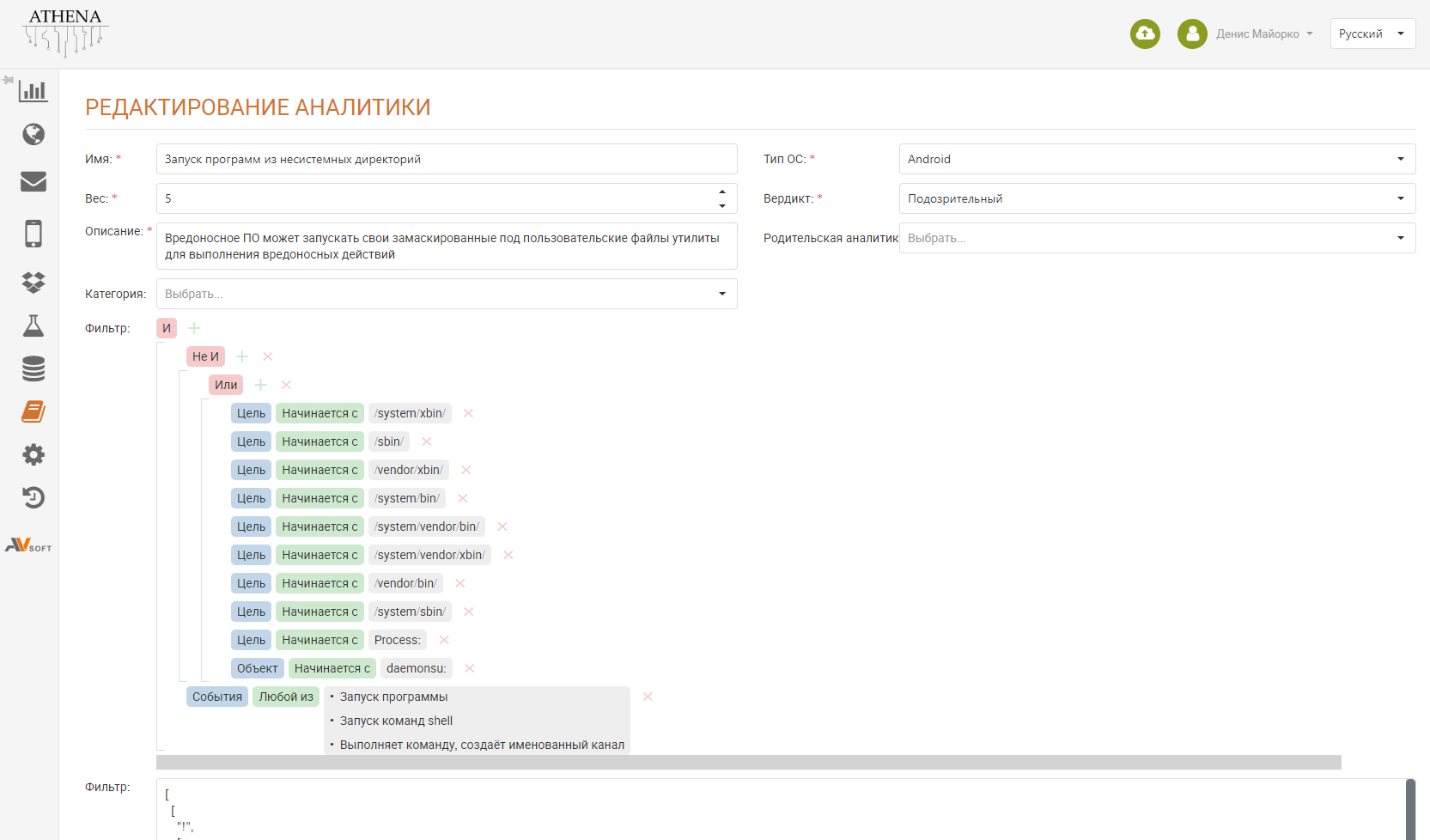


Рис. 3.11. Страница создания исследования

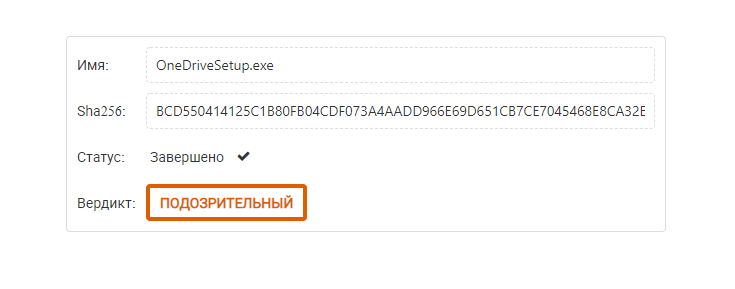
8. Страница загрузки файла (рис. 3.12-3.13).

Рис. 3.12. Страница загрузки подозрительного файла

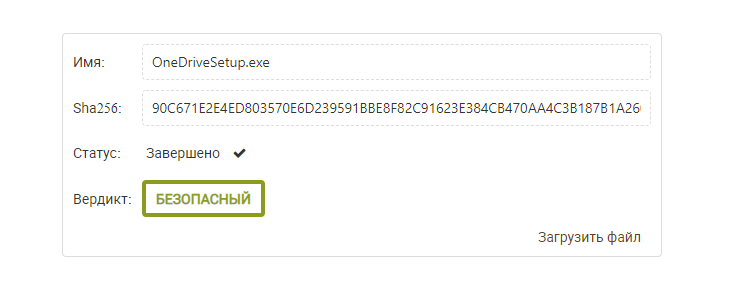


Рис. 3.13. Страница загрузки безопасного файла

# ЗАКЛЮЧЕНИЕ

В ходе исследования были проанализированы существующие системы выявления вредоносного ПО на основе механизма динамического анализа. Были рассмотрены три системы: Athena, KATA, TDS Polygon. Были рассмотрены методы анализа Big Data.

Была разработана функциональная модель процесса и методика выявления вредоносного ПО.

Был реализован модуль выявления вредоносного ПО, выполняющий поставленные задачи. В данный момент модуль интегрирован в систему Athena.

Результатом интеграции модуля было:

1. относительное увеличение частоты истинных срабатываний на 56% на выборке из 1000 файлов;

2. уменьшение частоты ложных срабатываний на 41% на выборке из 1000 файлов;

3. уменьшение потребляемых ресурсов процедурой анализа на одном исследовании на 70%;

4. увеличение скорости анализа на одном исследовании на 10%.

В дальнейшем существует возможность расширения и дополнения модуля, добавление анализируемых полей, моделей и систем.

# СПИСОК ЛИТЕРАТУРЫ

1. Athena. Основная информация [В Интернете]. - 25 08 2018 г.. - http://avsw.ru/.
2. KATA. Информация для ознакомления [В Интернете]. - 25 08 2018 г.. - https://www.all-smety.ru/upload/KATA%20-%20Kaspersky\_Anti\_Targeted%20\_Attack\_Platform\_WhitePaper\_RU.PDF.
3. RFC8259 [В Интернете] // tools.ietf.org. - 04 02 2019 г.. - https://tools.ietf.org/html/rfc8259.
4. TDS. Знакомство с системой [В Интернете]. - 25 08 2018 г.. - https://www.anti-malware.ru/reviews/TDS-Group-IB-detection-of-targeted-attacks.
5. Анализ методов обнаружения вредоносных программ [В Интернете] / авт. Лысенко А. В. Кожевникова И. С., Ананьин Е. В., Никишова А. В. // Молодой ученый. - 2016 г.. - 21 04 2019 г.. - https://moluch.ru/archive/125/34803/.
6. Анатомия таргетированной атаки. [В Интернете] / авт. Левцов В. Демидов Н. // Системный администратор. - 2016 г.. - http://samag.ru/archive/article/3188.
7. Антивирусная защита: Учебное пособие для защиты информационных ресурсов [В Интернете] / авт. А.В. Фролов // Frolov-lib.ru. - 2004 г.. - 2019 04 21 г.. - http://frolov-lib.ru/books/av/ch05.html.
8. Архитектурный подход [В Интернете] // Энциклопедия информационного общества. - 2019 04 15 г.. - http://wiki.iis.ru/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%BD%D1%8B%D0%B9\_%D0%BF%D0%BE%D0%B4%D1%85%D0%BE%D0%B4.
9. Большие данные [В Интернете] // Википедия. - 15 02 2019 г.. - https://ru.wikipedia.org/wiki/%D0%91%D0%BE%D0%BB%D1%8C%D1%88%D0%B8%D0%B5\_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5.
10. Импортозамещение ПО [В Интернете] // ИнфраОфис. - 15 03 2019 г.. - https://infraoffice.pro/noimport.
11. Менеджмент качества [В Интернете] // Управление производством. - 21 01 2019 г.. - http://www.up-pro.ru/encyclopedia/management-kachestva.html.
12. Новые виды вредоносных программ и методы их обнаружений [В Интернете]. - 25 08 2018 г.. - https://habr.com/company/microsoft/blog/352376/.
13. ПРОЦЕССНЫЙ ПОДХОД [В Интернете] // КПМС. - http://www.kpms.ru/General\_info/Process\_approach.htm.
14. Разработка через тестирование [В Интернете] // Википедия. - 04 02 2019 г.. - https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0\_%D1%87%D0%B5%D1%80%D0%B5%D0%B7\_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5.
15. Регрессионный анализ [В Интернете] // Википедия. - 04 02 2019 г.. - https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9\_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7.
16. Системный подход [В Интернете] // Википедия. - 08 03 2019 г.. - https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D1%8B%D0%B9\_%D0%BF%D0%BE%D0%B4%D1%85%D0%BE%D0%B4.
17. Сравнение производительности MongoDB vs PostgreSQL. Часть I: No index [В Интернете] // Habr. - 03 02 2019 г.. - https://habr.com/ru/post/197590/.
18. ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ [В Интернете] // ITстан. - http://www.itstan.ru/funk-strukt-analiz/funkcionalnoe-modelirovanie.html.

# **ПРИЛОЖЕНИЕ**

## **ЛИСТНИНГ КОДА**

namespace Athena.Extreme.ApiControllers

{

[Authorize]

[Produces("application/json")]

[Route("api/researches/dynamic/report/{id}")]

public class DynamicResearchReportMongoController : Controller

{

private readonly ICrypter \_crypter;

private readonly IMongoService \_mongoService;

public DynamicResearchReportMongoController(IMongoService mongoService, ICrypter crypter)

{

\_crypter = crypter;

\_mongoService = mongoService;

}

[Route("events")]

[HttpGet]

public object GetDynamicResearchEvents(DataSourceLoadOptions loadOptions, int id, [FromQuery]string analytic, [FromQuery] bool isArchived)

{

var collection = isArchived ? \_mongoService.GetArchiveEventsCollection() : \_mongoService.GetEventsCollection();

var query = collection.AsQueryable().Where(x => x.RschId == id);

if (analytic != null)

query = query.Where(x => x.AnalyticDescription.Contains(analytic));

return DataSourceLoader.Load(query, loadOptions);

}

[Route("processes")]

[HttpGet]

public object GetDynamicResearchProcesses(DataSourceLoadOptions loadOptions, int id, [FromQuery] bool isArchived)

{

var collection = isArchived ? \_mongoService.GetArchiveProcessesCollection() : \_mongoService.GetProcessesCollection();

var query = collection.AsQueryable()

.Where(x => x.ResearchId == id)

.Select(x => new

{

x.Id,

x.ResearchId,

x.ResTime,

x.Total.GPU,

x.Total.CPU,

x.Total.DiskRead,

x.Total.DiskWrite,

x.Total.Memory,

x.Total.MemoryPercent,

x.Total.NetworkIn,

x.Total.NetworkOut,

x.Processes

});

return DataSourceLoader.Load(query, loadOptions);

}

[Route("processesTree")]

[HttpGet]

public object GetDynamicResearchProcessesTree(DataSourceLoadOptions loadOptions, int id, [FromQuery] bool isArchived)

{

var collection = isArchived ? \_mongoService.GetArchiveProcessesTreeCollection() : \_mongoService.GetProcessesTreeCollection();

var query = \_mongoService.GetProcessesTreeCollection().AsQueryable()

.Where(x => x.ResearchId == id)

.Select(x => new

{

x.Pid,

x.Name,

x.CreationTime,

x.UnloadTime,

x.ChildrenProcesses,

x.Resources

});

return DataSourceLoader.Load(query, loadOptions);

}

[Route("processesSelect")]

[HttpGet]

public object GetDynamicResearchProcessesSelect(DataSourceLoadOptions loadOptions, int id, [FromQuery] bool isArchived)

{

var collection = isArchived ? \_mongoService.GetArchiveProcessesTreeCollection() : \_mongoService.GetProcessesTreeCollection();

var query = collection.AsQueryable()

.Where(x => x.ResearchId == id);

var processes = DynamicResearchReport.GetProcesses(query);

return DataSourceLoader.Load(processes, loadOptions);

}

[Route("systemResources/{processId?}")]

public object GetProcessesSystemResources(DataSourceLoadOptions loadOptions, int id, int? processId, [FromQuery] bool isArchived)

{

var collection = isArchived ? \_mongoService.GetArchiveProcessesCollection() : \_mongoService.GetProcessesCollection();

var query = collection.AsQueryable()

.Where(x => x.ResearchId == id).ToList()

.Select(x => new SystemResources

{

ResearchId = x.ResearchId,

ResTime = x.ResTime,

CPU = x.Total.CPU,

CPUCore = x.Total.Core,

GPU = x.Total.GPU,

GPUCore= x.Total.GPUCore,

MemoryPercent = x.Total.MemoryPercent,

Memory = x.Total.Memory,

Process = processId.HasValue ? x.Processes.FirstOrDefault(y => y.Pid == processId.Value) : null

});

return DataSourceLoader.Load(query, loadOptions);

}

[Route("networkResources/{processId?}")]

public object GetProcessesNetworkDiskResources(DataSourceLoadOptions loadOptions, int id, int? processId, [FromQuery] bool isArchived)

{

var collection = isArchived ? \_mongoService.GetArchiveProcessesCollection() : \_mongoService.GetProcessesCollection();

var query = collection.AsQueryable()

.Where(x => x.ResearchId == id).ToList()

.Select(x => new NetworkAndDisk

{

ResearchId = x.ResearchId,

ResTime = x.ResTime,

NetworkIn = x.Total.NetworkIn,

NetworkOut = x.Total.NetworkOut,

DiskRead = x.Total.DiskRead,

DiskWrite = x.Total.DiskWrite,

Process = processId.HasValue ? x.Processes.FirstOrDefault(y => y.Pid == processId.Value) : null

});

return DataSourceLoader.Load(query, loadOptions);

}

}

}

using Athena.Service.Analytic.Mongo.Models;

using MongoDB.Bson;

using MongoDB.Driver;

using Serilog;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Threading.Tasks;

namespace Athena.Service.Analytic.Mongo

{

public class MongoIndexesUpdater : IMongoIndexesUpdater

{

private readonly IMongoDbProvider \_mongoDbProvider;

private readonly ILogger \_logger;

public MongoIndexesUpdater(IMongoDbProvider mongoDbProvider, ILogger logger)

{

\_mongoDbProvider = mongoDbProvider;

\_logger = logger;

}

public async Task<HttpStatusCode> UpdateAsync(string[] requiredIndexes)

{

\_logger.Information("Starting indexes updating");

IEnumerable<string> indexestoadd;

var createdcount = 0;

try

{

var collection = \_mongoDbProvider.Database.GetCollection<MongoResearchEvent>("ResearchEvents");

\_logger.Information("-> Getting indexes list..");

var indexList = await collection.Indexes.ListAsync();

\_logger.Information("-> Indexes got");

\_logger.Information("-> Detecting missing indexes");

IEnumerable<BsonDocument> currentindexbson;

if (indexList.MoveNext())

currentindexbson = indexList.Current;

var currentindexes = indexList.Current.Select(x => x.FirstOrDefault(y => y.Name == "key").Value.ToBsonDocument().Names.First());

indexestoadd = requiredIndexes.Except(currentindexes);

if (indexestoadd.Count()==0)

{

\_logger.Information("-> All indexes exists");

\_logger.Information("Indexes updated");

return HttpStatusCode.OK;

}

\_logger.Information("-> Missing indexes detected");

\_logger.Information($"-> Creating missing indexes [Indexes=[{string.Join(", ",indexestoadd)}]]");

foreach (var index in indexestoadd)

{

\_logger.Information($"--> Creating index {index}..");

try

{

var indexToCreate = Builders<MongoResearchEvent>.IndexKeys;

var indexModel = new CreateIndexModel<MongoResearchEvent>(indexToCreate.Ascending(index));

await collection.Indexes.CreateOneAsync(indexModel);

createdcount++;

}

catch (Exception ie)

{

\_logger.Error(ie,$"--> Unexpected error! Index [{index}] wasn't created. [ErrorMessage={{0}}]");

continue;

}

\_logger.Information($"--> Index {index} created");

}

\_logger.Information("-> Missing indexes created");

}

catch (Exception e)

{

\_logger.Error($"Unexpected error! Indexes wasn't created. [ErrorMessage={{{e.ToString()}}}]");

return HttpStatusCode.InternalServerError;

}

\_logger.Information("Indexes updated");

if (createdcount < indexestoadd.Count())

return HttpStatusCode.PartialContent;

return HttpStatusCode.OK;

}

}

}

using Athena.Service.Analytic.Mongo.Parsers.Common.Block;

using Athena.Service.Analytic.Mongo.Parsers.Common.Enums;

using Athena.Service.Analytic.Mongo.Parsers.DevExtreme.Expressions;

using NUnit.Framework;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Athena.Service.Analytic.UnitTests.ParserDetailedTests.ToJson

{

public class BlockTest

{

[TestCase(1, LogicStatus.And, "")]

[TestCase(1, LogicStatus.Undefined, "")]

[TestCase(2, LogicStatus.Undefined, "")]

[TestCase(2, LogicStatus.And, "{'$and': [")]

[TestCase(2, LogicStatus.Or, "{'$or': [")]

public void GetLogicOpeningStringTest(int count, LogicStatus logic, string expectedResult)

{

//Arrange

var block = new TestingBlock

{

logic = logic

};

for (int i = 0; i < count; i++)

block.expressions.Add(new EqualityExpression("who", "=", "asdasd", null, null));

//Act

var result = block.GetLogicOpeningStringTest;

//Assert

Assert.AreEqual(expectedResult, result);

}

[TestCase(1, LogicStatus.And, "")]

[TestCase(1, LogicStatus.Undefined, "")]

[TestCase(2, LogicStatus.Undefined, "")]

[TestCase(2, LogicStatus.And, "]}")]

[TestCase(2, LogicStatus.Or, "]}")]

public void GetLogicClosingStringTest(int count, LogicStatus logic, string expectedResult)

{

//Arrange

var block = new TestingBlock

{

logic = logic

};

for (int i = 0; i < count; i++)

block.expressions.Add(new EqualityExpression("who", "=", "asdasd", null, null));

//Act

var result = block.GetLogicClosingStringTest;

//Assert

Assert.AreEqual(expectedResult, result);

}

[TestCase(true, LogicStatus.And, true, true, false, LogicStatus.Or)]

[TestCase(true, LogicStatus.Or, true, true, false, LogicStatus.And)]

[TestCase(false, LogicStatus.Or, false, false, false, LogicStatus.Or)]

[TestCase(false, LogicStatus.And, false, false, false, LogicStatus.And)]

public void InverseContentTest(bool blockNegative, LogicStatus logic, bool firstExpressionBlockNegative, bool secondExpressionBlockNegative, bool resultBlockNegative, LogicStatus resultLogic)

{

//Arrange

var block = new TestingBlock

{

blockNegative = blockNegative,

logic = logic

};

block.expressions.Add(new EqualityExpression("who", "=", "asdasd", null, null));

block.expressions.Add(new EqualityExpression("who", "<>", "asdasd", null, null));

//Act

block.InverseContentTest();

//Assert

Assert.AreEqual(firstExpressionBlockNegative, block.expressions[0].blockNegative);

Assert.AreEqual(secondExpressionBlockNegative, block.expressions[1].blockNegative);

Assert.AreEqual(resultBlockNegative, block.blockNegative);

Assert.AreEqual(resultLogic, block.logic);

}

}

public class TestingBlock : Block

{

public string GetLogicOpeningStringTest

{

get => GetLogicOpeningString;

}

public string GetLogicClosingStringTest

{

get => GetLogicClosingString;

}

public void InverseContentTest()

{

InverseContent();

}

}

}

namespace Athena.Service.Analytic.UnitTests.ParserDetailedTests.ToJson

{

public class EqualityExpressionTest

{

readonly Dictionary<string, string> Fields = new Dictionary<string, string>

{

{"event", "Event"},

{"who", "Who"},

};

[TestCase("who", "=", "[who]", "{$expr:{ $eq: [\"$Who\", \"$who\"] }}",

TestName = "ExpressionsToJson\_ParseEqualityExpressionToJson01\_ShouldReturnParsed")]

[TestCase("event", "=", "who", "{ \"Event\": { $eq: who }}",

TestName = "ExpressionsToJson\_ParseEqualityExpressionToJson02\_ShouldReturnParsed")]

public void ExpressionsToJson\_ParseEqualityExpressionToJson\_ShouldReturnParsed(string field, string @operator, string value, string expected)

{

//Arrange

var expression = new EqualityExpression(field, @operator, value, Fields, null);

// Action

var result = expression.ToJson();

// Assert

Assert.AreEqual(expected, result);

}

}

}

using Athena.Service.Analytic.Mongo.Parsers;

using Athena.Service.Analytic.Mongo.Parsers.Common;

using Athena.Service.Analytic.Mongo.Parsers.Common.Block;

using Athena.Service.Analytic.Mongo.Parsers.Common.Enums;

using Athena.Service.Analytic.Mongo.Parsers.DevExtreme;

using Athena.Service.Analytic.Mongo.Parsers.DevExtreme.Expressions;

using MongoDB.Bson;

using NSubstitute;

using NUnit.Framework;

using Serilog;

namespace Athena.Service.Analytic.UnitTests.ParserDetailedTests.ToJson

{

public class SupportFunctionsTest

{

readonly Dictionary<string, string> Fields = new Dictionary<string, string>

{

{"module", "Module"},

{"event", "Event"},

{"who", "Who"},

{"dest", "Dest"},

{"descr", "Description"},

{"rschId", "RschId"},

{"pid1", "Pid1"},

{"pid2", "Pid2"},

{"adddata1", "Adddata1"},

{"adddata2", "Adddata2"},

{"timeofevent", "TimeOfEvent"},

{"status", "Status"},

{"timediff", "TimeDiff"},

{"Significance", "Significance"},

{"Tid", "Tid"},

{"Uid", "Uid"},

{"Gid", "Gid"},

{"ErrorCode", "ErrorCode"},

{"AnalyticDecription", "AnalyticDescription"},

{"statusdescription", "StatusDescription"}

};

[TestCase("Who", new[] { "Event", "adddata" }, false,

TestName = "SupportFunctions\_IsIntField01\_ShouldReturnIsIntField")]

[TestCase("event", new[] { "Who", "Event", "adddata" }, false,

TestName = "SupportFunctions\_IsIntField02\_ShouldReturnIsIntField")]

[TestCase("Who", null, false,

TestName = "SupportFunctions\_IsIntField03\_ShouldReturnIsIntField")]

[TestCase("Event", new[] { "Who", "Event", "adddata" }, true,

TestName = "SupportFunctions\_IsIntField04\_ShouldReturnIsIntField")]

public void SupportFunctions\_IsIntField\_ShouldReturnIsIntField(string field, IEnumerable<string> intFields, bool expected)

{

Assert.AreEqual(expected, SupportFunctions.IsIntField(field, intFields));

}

[TestCase(true, "$nin:",

TestName = "SupportFunctions\_GetInOperatorString01\_ShouldReturnNin")]

[TestCase(false, "$in:",

TestName = "SupportFunctions\_GetInOperatorString02\_ShouldReturnIn")]

public void SupportFunctions\_GetInOperatorString\_ShouldReturnStringValue(bool blockNegative, string expected)

{

//Act

var result = SupportFunctions.GetInOperatorString(blockNegative);

//Assert

Assert.AreEqual(expected, result);

}

[TestCase("who", null,

TestName = "SupportFunctions\_CreateInToOrBlock01\_ShouldReturnException")]

[TestCase("who", new string[] { },

TestName = "SupportFunctions\_CreateInToOrBlock02\_ShouldReturnException")]

[TestCase("who", new string[] { "", "" },

TestName = "SupportFunctions\_CreateInToOrBlock03\_ShouldReturnException")]

public void SupportFunctions\_CreateInToOrBlock\_ShouldReturnException(string field, string[] inside)

{

Assert.Catch<ArgumentNullException>(()=> { SupportFunctions.CreateInToOrBlock(field, inside, null, null); });

}

[TestCase(true,

LogicStatus.And,

TestName = "SupportFunctions\_SetInOrBlockLogic01")]

[TestCase(false,

LogicStatus.Or,

TestName = "SupportFunctions\_SetInOrBlockLogic02")]

public void SetInOrBlockLogic(bool blockNegative, LogicStatus expected)

{

//Arrange

var block = new DevExtremeBlock();

//Act

SupportFunctions.SetInOrBlockLogic(block, blockNegative);

//Assert

Assert.AreEqual(expected, block.logic);

}

[TestCase("[who]", "who",

TestName = "SupportFunctions\_GetBraketlessValue01\_ShouldReturnBraketlessValue")]

[TestCase("[who", "[who",

TestName = "SupportFunctions\_GetBraketlessValue02\_ShouldReturnValue")]

[TestCase("who]", "who]",

TestName = "SupportFunctions\_GetBraketlessValue03\_ShouldReturnValue")]

public void GetBraketlessValue(string value, string expected)

{

//Act

var result = SupportFunctions.GetBraketlessValue(value);

//Assert

Assert.AreEqual(expected, result);

}

[TestCase("who", "Who",

TestName = "SupportFunctions\_GetMongoFieldNameValue01\_ShouldReturnMongoValue")]

[TestCase("[who", "[who",

TestName = "SupportFunctions\_GetMongoFieldNameValue02\_ShouldReturnValue")]

public void GetMongoFieldNameValue(string field, string expected)

{

//Act

var result = SupportFunctions.GetMongoFieldName(field, Fields);

//Assert

Assert.AreEqual(expected, result);

}

[TestCase("who", "who",

TestName = "SupportFunctions\_GetMongoFieldNameValue03\_ShouldReturnValue")]

public void GetMongoFieldNameValueNull(string field, string expected)

{

//Act

var result = SupportFunctions.GetMongoFieldName(field, null);

//Assert

Assert.AreEqual(expected, result);

}

[TestCase(true, "{'$not':[",

TestName = "SupportFunctions\_GetNotExprOpeningString01")]

[TestCase(false, "",

TestName = "SupportFunctions\_GetNotExprOpeningString02")]

public void SupportFunctions\_GetNotExprOpeningString(bool blockNegative, string expected)

{

//Act

var result = SupportFunctions.GetNotExprOpeningString(blockNegative);

//Assert

Assert.AreEqual(expected, result);

}

[TestCase(true, "]}",

TestName = "SupportFunctions\_GetNotExprClosingString01")]

[TestCase(false, "",

TestName = "SupportFunctions\_GetNotExprClosingString02")]

public void SupportFunctions\_GetNotExprClosingString(bool blockNegative, string expected)

{

//Act

var result = SupportFunctions.GetNotExprClosingString(blockNegative);

//Assert

Assert.AreEqual(expected, result);

}

[TestCase(true, "{'$not':",

TestName = "SupportFunctions\_GetNotOpeningString01")]

[TestCase(false, "",

TestName = "SupportFunctions\_GetNotOpeningString02")]

public void SupportFunctions\_GetNotOpeningString(bool blockNegative, string expected)

{

//Act

var result = SupportFunctions.GetNotOpeningString(blockNegative);

//Assert

Assert.AreEqual(expected, result);

}

[TestCase(true, "}",

TestName = "SupportFunctions\_GetNotClosingString01")]

[TestCase(false, "",

TestName = "SupportFunctions\_GetNotClosingString02")]

public void SupportFunctions\_GetNotClosingString(bool blockNegative, string expected)

{

//Act

var result = SupportFunctions.GetNotClosingString(blockNegative);

//Assert

Assert.AreEqual(expected, result);

}

[TestCase("Who", new[] { "Event", "adddata" }, "{$toLower:\"$Who\"}",

TestName = "SupportFunctions\_ToLowString01")]

[TestCase("Who", null, "{$toLower:\"$Who\"}",

TestName = "SupportFunctions\_ToLowString02")]

[TestCase("Event", new[] { "Who", "Event", "adddata" }, "{$substr:[\"$Event\", 0, -1 ]}",

TestName = "SupportFunctions\_ToLowString03")]

public void SupportFunctions\_ToLowString(string value, IEnumerable<string> intFields, string expected)

{

//Act

var result = SupportFunctions.ToLowString(intFields, value);

//Assert

Assert.AreEqual(expected, result);

}

[TestCase("Who", new[] { "Event", "adddata" }, "Who",

TestName = "SupportFunctions\_ToString01")]

[TestCase("Who", null, "Who",

TestName = "SupportFunctions\_ToString02")]

[TestCase("Event", new[] { "Who", "Event", "adddata" }, "{$substr:[\"$Event\", 0, -1 ]}",

TestName = "SupportFunctions\_ToString03")]

public void SupportFunctions\_ToString(string value, IEnumerable<string> intFields, string expected)

{

//Act

var result = SupportFunctions.ToString(intFields, value);

//Assert

Assert.AreEqual(expected, result);

}

}

}