

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

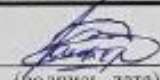
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

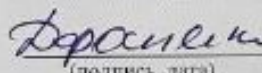
на тему

ПРИЛОЖЕНИЕ АНАЛИЗА
БЕЗОПАСНОСТИ PDF-ФАЙЛОВ

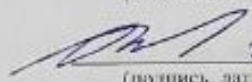
Студент

 26.05.2025 — Макаров Илья Сергеевич
(подпись, дата) (ФИО полностью)

Руководитель

 30.05.2025 — Дорошенко И.Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

 30.05.2025 — Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой

 М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от 2 июня 2025 г. № 3)

Секретарь ГЭК

 Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 06 » мая 2025 г

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Макаров Илья Сергеевич гр. 21ВВП1 факультета вычислительной техники

2. Тема работы Приложение анализа безопасности PDF-файлов.

Тема утверждена приказом ПГУ № 601/о от "06" мая 2025г.

3. Руководитель бакалаврской работы доцент кафедры ВТ ПГУ Дорошенко И.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать программное средство для анализа PDF-документов с целью выявления скрытого и вредоносного содержимого, включая работу с зашифрованными файлами и проверку целостности. Приложение должно обеспечивать визуализацию структуры документа, декодирование потоков и выявление угроз, таких как внедрённые скрипты и скрытые объекты.

Средства разработки: язык C#, платформа .NET Framework, технология WinForms.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: исследовать структуру PDF-документа, изучить существующие инструменты и библиотеки для работы с PDF, выбрать технологии и средства разработки. Изучить и спроектировать архитектуру приложения, протестировать на различных по наполненности PDF-документах.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	4%	06.05.2025	Фрошук
2	Анализ требований	4%	07.05.2025	Фрошук
3	Выбор технологий, применяемых при проектировании и разработке	2%	08.05.2025	Фрошук
4	Проектирование архитектуры	10%	09.05.2025 - 11.05.2025	Фрошук
5	Проектирование и Разработка ПО	50%	12.05.2025 - 24.05.2025	Фрошук
6	Тестирование и отладка	20%	25.05.2025 - 29.05.2025	Фрошук
7	Оформление выпускной работы	10%	30.05.2025	Фрошук

Дата выдачи задания "06" мая 2025 г.

Руководитель бакалаврской работы Фрошук 06.05.2025 Дорощенко И.Н.
(подпись, дата)

Задание к исполнению принял студент Макаров И.С. 06.05.2025
(подпись, дата)

Работу к защите допустить
Декан факультета Фионова Л.Р. 30.05.2025
(подпись, дата)

Реферат

Бакалаврская работа содержит 144 листа, 43 рисунка, 1 таблицу, 10 источников, 4 приложения.

PDF, АНАЛИЗ PDF-ДОКУМЕНТОВ, БЕЗОПАСНОСТЬ, ОБЪЕКТЫ PDF, ДЕКОДИРОВАНИЕ ПОТОКОВ, МЕТАДАННЫЕ, C#, WINFORM

Цель работы – разработать программное средство для анализа PDF-документов с целью выявления угроз внутри файлов.

Разработанное приложение предназначено для глубокого анализа внутренней структуры PDF-документов с целью выявления скрытого содержимого и выполнения встроенных JavaScript-скриптов. Программа обеспечивает просмотр и декодирование объектов как обычных, так и зашифрованных PDF-файлов, анализ метаданных, потоков данных и других ключевых компонентов документа.

В конечном итоге было создано приложение на языке программирования C# с использованием платформы .NET Framework и технологии WinForms. Разработка обеспечила выполнение всех заявленных требований, включая визуализацию структуры PDF, поддержку различных версий формата, дешифровку защищённых сегментов и пользовательский интерфейс для эффективного анализа документов.

					ПГУ1.090301.13.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Приложение анализа безопасности PDF-файлов Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Макаров И.С.		20.06.17					
Провер.		Дорошенко И.Н.		20.06.17					
Н. Контр		Бычков А.С.		20.06.17			2	145	
Утверд.		Митракин М.А.		20.06.17		ФВТ, каф. ВТ гр. 218ВП1			

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Анализ предметной области	7
1.1 Анализ актуальности темы	7
1.2 Основы формата PDF	9
1.2.1 Введение в формат PDF	9
1.2.2 История развития PDF и современные стандарты.....	10
1.2.3 Структура PDF-документа, компоненты и их роль.	12
1.2.4 Каталог PDF	14
1.2.5 Метаданные PDF.....	16
1.2.6 Безопасность PDF	19
1.3 Обзор существующих инструментов и библиотек для работы с PDF ...	25
1.3.1 RUPS (iText)	25
1.3.2 PDFDebugger (PDFBox)	26
1.3.3 IPA	28
1.3.4 MuPDF.....	29
1.3.5 Сравнительный анализ существующих инструментов и библиотек	30
1.4 Выбор технологий и средств разработки приложения	31
1.4.1 Выбор языка программирования	31
1.4.2 Выбор среды разработки.....	33
1.4.3 Выбор платформ реализации графического интерфейса	34
1.5 Определение требований к разрабатываемому ПО.....	35
1.5.1 Требования к программной части.....	35
1.5.2 Требования к аппаратуре и программному обеспечению	36
2 Проектирование и Разработка ПО.....	37
2.1 Архитектура приложения.....	37
2.1.1 Подсистема управления PDF.....	40

2.1.2 Ядро разбора и анализа	40
2.1.3 Подсистема визуализации и интерактивного анализа.....	40
2.2 Реализация приложения	41
2.2.1 Разбор структуры PDF-документа	42
2.2.2 Разбор объектов PDF и работа с ними	47
2.2.3 Обработка объектов XObject	54
2.2.4 Форматирование данных и вывод.....	57
2.3 Разработка пользовательского интерфейса.....	59
3 Результаты работы приложения	67
3.1 Тестирование корректности работы программы	67
3.2 Тестирование небезопасных PDF документов	73
3.2.1 Угрозы, выявляемые разработанным приложением.....	73
3.2.2 Иные угрозы	83
3.3 Дальнейшие развитие приложения	85
4 Безопасность жизнедеятельности.....	89
4.1 Требования к помещениям для работы с ПК	89
4.2 Требования к организации рабочих мест	91
Заключение	94
Список использованных источников	96
Приложение А – UML-диаграммы.....	98
Приложение Б – Листинг программы	103
Приложение В – Презентация.....	134
Приложение Г – Безопасность исполнения и защита от вредоносного кода	141

Перечень сокращений и обозначений

В данной работе применяют следующие сокращения и обозначения

ИИ – искусственный интеллект

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

AES (Advanced Encryption Standart) – симметричный алгоритм блочного шифрования

СМЫК (CYAN MAGENTA YELLOW BLACK) – четырёхцветная автотипия, субтрактивная схема формирования цвета

DMS (Document Management System) – система управления документами

GDPR (General Data Protection Regulation) – общий регламент по защите данных

IT (Information Technology) - информационные технологии

RGB (RED GREEN BLUE) – аддитивная цветовая модель

SHA (Secure Hash Algorithm) – алгоритм хеширования

UI (User Interface) – пользовательский интерфейс

XML (eXtensible Markup Language) – расширяемый язык разметки

Введение

В эпоху цифровых технологий формат PDF стал основным инструментом для хранения и обмена документацией в самых разных сферах, от юридической до технической. Его уникальная способность заключается в возможности обеспечивать стабильное отображение контента независимо от устройства или операционной системы, что делает его чрезвычайно популярным в документообороте. Однако с расширением области использования PDF возрастает и угроза безопасности, связанная с возможностью скрытого внедрения вредоносных программ или фальсификаций данных. В результате, даже технически защищённые или подписанные файлы могут скрывать опасные элементы.

Цель данной работы – разработка программного решения, которое будет анализировать PDF-документы с целью выявления аномалий и скрытых угроз. В ходе работы планируется создать инструмент, способный декодировать потоки данных, изучать их контекст и определять наличие враждебных объектов, что позволит автоматизировать процесс защиты информации и повысить доверие к используемым в документообороте форматам.

Практическая значимость работы заключается в создании эффективного инструмента для специалистов в области информационной безопасности, который позволит оперативно выявлять и устранять угрозы, скрытые в структуре PDF-документов, а также улучшить методы защиты и верификации информации в корпоративных и государственных системах.

В ходе работы будут получены навыки анализа формата PDF, разработки инструментов для работы с различными типами данных и создания алгоритмов для поиска скрытых угроз в документах. Эти навыки полезны для будущей профессиональной деятельности в области информационной безопасности и разработки программного обеспечения, а также могут быть использованы в задачах цифровой криминалистики и аудита документооборота.

1 Анализ предметной области

Предметная область исследования охватывает спецификацию PDF как многоуровневого контейнерного формата, основанного на взаимосвязанных объектах, потоках и кросс-ссылках, каждый из которых может содержать собственные метаданные, фильтры компрессии и шифрования. Ключевой задачей является разработка алгоритмов автоматического декодирования и инспекции потоков для выявления аномалий и несоответствий между заявленным и фактическим содержимым. Эффективная реализация таких методов позволяет повысить точность верификации целостности и подлинности PDF-файлов в условиях больших объёмов документных данных.

1.1 Анализ актуальности темы

Развитие электронного документооборота распространение формата PDF ставят перед современными специалистами по информационной безопасности и цифровой криминалистике новую задачу – обеспечение прозрачности и контроля над скрытым содержимым документов. Актуальность темы глубинного анализа PDF-документов обусловлена гигантскими объёмами обмена этим форматом в самых разных отраслях и растущей сложностью встроенных возможностей. Ежедневно по всему миру создаётся и передаётся свыше 250 млн PDF-файлов, причём значительная доля деловой и технической документации, финансовых отчётов и юридических соглашений хранится именно в этом формате. При этом за последние пять лет число зарегистрированных уязвимостей в популярных PDF-ридерах выросло более чем на 60 %, а атаки с использованием вредоносных JavaScript-скриптов внутри PDF-файлов составляют до 25 % всех фишинговых кампаний – что подчёркивает критическую потребность в надёжных инструментах для глубокого изучения внутренней структуры документа [1].

PDF является не только универсальным инструментом для обмена информацией, но и представляет собой сложную структуру, включающую

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

объектные и кросс-ссылочные потоки, зашифрованные сегменты, таблицы шрифтов и кодировок. Эти особенности, а также возможности динамической вставки контента через Object Streams и Optional Content Groups создают благодатную почву для скрытых изменений и фальсификаций. Встроенные механизмы, такие как скрытые слои и инкрементные обновления потоков, позволяют злоумышленникам маскировать изменения, изменять смысл соглашения или загружать на машину пользователя вредоносный код или исполняемый файл, даже в документ после цифровой подписи, что превращает формально «подписанный» документ в уязвимый объект для атак.

С точки зрения информационной безопасности, это превращает каждую корпоративную или официальную PDF-переписку в потенциальную точку проникновения для атакующих. Именно поэтому решение задачи глубокого анализа структуры PDF-документов представляет собой важный шаг в направлении повышения уровня защиты данных и повышения доверия к использованию данного формата в различных сферах деятельности.

Решение задачи глубокого анализа PDF-структур актуально для:

- инженеров по информационной безопасности, проводящих пентесты и аудит внутренних систем;
- экспертов цифровой криминалистики, исследующих подлинность и историю изменений документов;
- разработчиков DMS и compliance-офицеров, отвечающих за соблюдение стандартов GDPR, SOX;
- преподавателей и исследователей, обучающих методам реверс-инжиниринга и киберзащиты.

Практическая значимость исследования подтверждается возможностью интеграции результатов в систему автоматизированного аудита, что позволит исключить ручную работу при анализе документов и повысить скорость обнаружения угроз PDF. Таким образом, тема проекта представляет собой

гармоничное соединение теоретических основ формата PDF и современных производственных потребностей IT-сообщества.

1.2 Основы формата PDF

1.2.1 Введение в формат PDF

Portable Document Format (рисунок 1) был создан как универсальный формат обмена документами, сохраняющий оригинальное форматирование, шрифты и графику на любых устройствах и в любых операционных системах. Ключевой особенностью PDF является его инкапсуляционная модель, объединяющая текст, векторную и растровую графику, мультимедийные элементы, интерактивные формы и метаданные в единый структурированный двоичный файл, который рендеринг-движки интерпретируют предсказуемым образом.



Рисунок 1 – Логотип PDF файла компании Adobe

В отличие от гибких текстовых и веб-форматов, PDF реализует концепцию «фиксированной вёрстки»: при открытии документа все элементы – текст, изображения, векторные объекты и макеты страниц – отображаются точно так же, как в исходном файле, вне зависимости от размера экрана или доступного программного обеспечения. В основе формата лежит модульная структура, в которой документ разбивается на независимые «косвенные объекты»: словари (служебные сведения и метаданные), массивы, ссылки и потоки данных. Потоки могут содержать команды рисования (текстовые операторы, графические примитивы), сжатые или зашифрованные бинарные данные, а также встроенные

мультимедиа. Такая архитектура позволяет выборочно анализировать и модифицировать отдельные части файла - от страницы и текстового блока до конкретного встроенного изображения.

Среди ключевых преимуществ PDF формата выделяют:

- независимость от платформы – документы идентично отображаются на MS Windows, macOS, мобильных устройствах и в веб-браузерах;

- поддержка многослойности – возможность разделения контента на слои (например, текст и аннотации), что удобно для сложной полиграфии и интерактивных карт;

- интерактивность – интеграция гиперссылок, заполнения форм, 3D-объектов, скриптов и мультимедиа.

Со временем PDF эволюционировал в открытый международный стандарт (ISO 32000), обзавёлся отраслевыми профилями для архивирования, полиграфии и доступности, а также встроенными механизмами безопасности: шифрованием (RC4, AES-128), цифровыми подписями и управлением правами (печать, копирование, редактирование). Сегодня PDF – это не просто универсальный контейнер для статичных отчётов, а мощная платформа для интерактивных форм, мультимедиа и защищённого обмена документами в самых разных отраслях.

1.2.2 История развития PDF и современные стандарты

Формат PDF был впервые представлен компанией Adobe Systems совместно с выпуском Acrobat 1.0 в январе 1993 г. На ранних этапах (PDF 1.0–1.2, 1993–1996 гг.) основное внимание уделялось встраиванию шрифтов, поддержке векторной графики и базовым схемам сжатия данных. Существенный импульс популяризации формата дал выпуск в 1994 г. бесплатного Acrobat Reader, что позволило миллионам пользователей просматривать PDF-документы без приобретения коммерческой лицензии.

На рисунке 2 представлены основные версии формата PDF в его ранней эволюции.



Рисунок 2 – История версий PDF

К рубежу тысячелетий PDF эволюционировал: версия 1.3 добавила закладки и аннотации, а в 1.4 появились прозрачность и усовершенствованная работа с цветом (CMYK). В январе 2008 г. спецификация PDF 1.7 была официально принята в качестве международного стандарта ISO 32000-1, что открыло формат для свободного внедрения сторонними разработчиками без лицензионных ограничений. Следующим этапом стало опубликование ISO 32000-2 (PDF 2.0) в июле 2017 г.: новая редакция ввела поддержку современных алгоритмов сжатия (JPEG 2000, JBIG2), расширенные механизмы безопасности (AES-256), семантическую разметку Tagged PDF и интеграцию мультимедиа и 3D-контента.

Помимо базовых версий, для решения прикладных задач разработаны специализированные подстандарты:

- PDF/A (ISO 19005) – для долгосрочного архивирования без динамического контента;
- PDF/X (ISO 15930) – для полиграфии с жёсткими требованиями к цветовым профилям и встроенным шрифтам;

– PDF/UA (ISO 14289) – для обеспечения доступности документов людям с ограниченными возможностями;

– PDF/E (ISO 24517) – для инженерных и технических приложений.

В итоге PDF прошёл путь от закрытого формата для копирования оригинального вида страниц до открытого стандарта, в котором сейчас есть разные версии под архивирование, печать, доступность и другие задачи.

1.2.3 Структура PDF-документа, компоненты и их роль.

В основе любого PDF-файла лежит последовательность взаимосвязанных секций, каждая из которых отвечает за определённую задачу при загрузке и отображении документа. На рисунке 3 продемонстрирована основная структура PDF-документа, начиная от заголовка до завершения обработки файла.

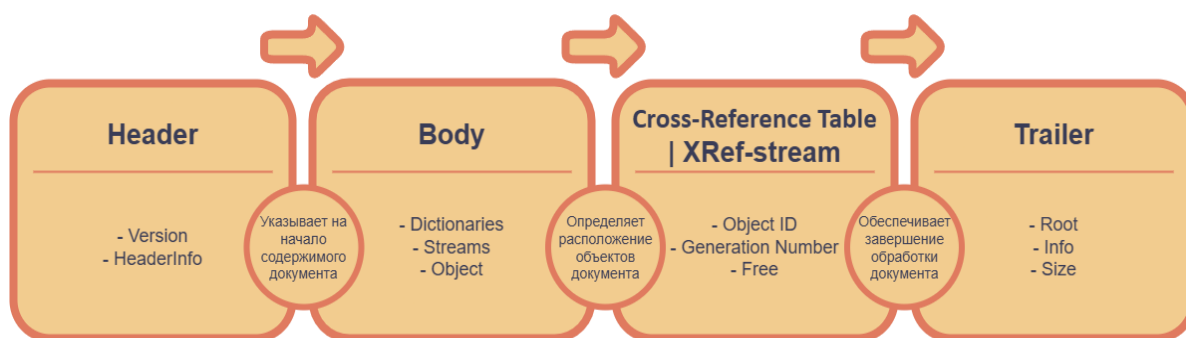


Рисунок 3 – Структура PDF-файла

Секции, составляющие структуру PDF:

– заголовок (Header) – Содержит подпись «%PDF-1.x» тем самым сообщает рендеру, что перед ним PDF-документ, и какую версию спецификации следует использовать. Эта информация необходима для корректного разбора всех последующих структур – от способов обработки сжатия до поддержки интерактивных элементов;

– тело документа (Body) формируется набором «косвенных» объектов, каждый из которых несёт свою функциональную нагрузку. Всего разделяются на 3 критерии:

а) словари (dictionaries) — ассоциативные структуры, описывающие служебные параметры. Они агрегируют метаданные и ссылки на дочерние объекты (каталог, шрифты, дерево страниц, параметры страницы, шифрование и т. д.);

б) потоки (streams) — большие сегменты данных, содержащие команды рисования текста и графики, а также двоичные блоки (сжатые изображения, XMP-метаданные);

в) простые объекты (числа, строки, имена, булевы значения, массивы) — используются для задания примитивных значений внутри словарей и в аргументах операторов.

Такая разбивка на объекты позволяет программам, работающим с форматом PDF, обращаться к конкретным частям документа по необходимости, не загружая всю структуру целиком;

– таблица кросс-ссылок (Cross-Reference Table или XRef-stream) представляет собой либо текстовую таблицу, либо поток XRef-stream. Служит индексом, где для каждого объекта хранится его смещение внутри файла. В старом стиле – это текстовая XRef-таблица, в PDF 1.5+ – специальный поток. Используя это, рендер моментально получает доступ к любому элементу без последовательного сканирования и значительно ускоряет открытие и навигацию по документу;

– трейлер (Trailer) объединяет ключевую служебную информацию:

а) /Size — общее число записей в таблице XRef (максимальный номер объекта + 1);

б) /Root — ссылка на корневой каталог (Catalog), откуда начинается построение дерева страниц;

в) /Info — ссылка на словарь метаданных (заголовок, автор, тема и пр.);

г) /Encrypt — ссылка на словарь параметров шифрования (если документ защищён);

д) /ID — массив из двух идентификаторов, обычно одинаковых, служащих для контроля целостности и версионности;

е) /Prev — указатель на смещение предыдущей XRef-таблицы при использовании инкрементальных обновлений (append-updates). Это позволяет читать историю изменений документа;

ж) /XRefStm — при использовании потоковых кросс-ссылок (PDF 1.5+) указывает на объект-поток, в котором хранится XRef, вместо классической текстовой таблицы.

Именно с трейлера начинается построение внутренней модели PDF: он указывает, с какого объекта нужно стартовать, чтобы собрать содержимое всего документа.

1.2.4 Каталог PDF

Каталог PDF (/Catalog) является важным компонентом структуры PDF-документа. Он представляет собой корневой объект, который устанавливает точку входа в логическую иерархию документа и управляет его основной структурой. Каталог задаёт основы для рендеринга страниц и организации ресурса, а также определяет, какие компоненты будут использованы для отображения контента на страницах документа.

Каталог — это словарь, который служит своего рода директивой для рендеринга документа, указывая на все важнейшие элементы, такие как страницы, метаданные, оглавление, формы и другие. Он связывает все ресурсы и функциональные элементы в документе, создавая основу для его логической и визуальной структуры. Основными элементами каталога являются:

– /Pages: важнейший элемент каталога, который отвечает за организацию страниц документа. Это объект, содержащий дерево страниц, где каждая страница представлена как отдельный узел в этом дереве. /Pages группирует страницы и управляет их иерархией, передавая по дереву ресурсы, такие как шрифты, палитры и макеты, которые необходимы для отображения контента.

– /Outlines: элемент, который если он присутствует в документе, содержит структуру оглавления документа, представляющую собой древовидную структуру с гиперссылками на различные разделы. /Outlines позволяет организовать навигацию по документу, что особенно полезно в больших и сложных PDF-документах, предоставляя пользователю удобный способ перехода между разделами.

– /Metadata: это ссылка на метаданные документа, которые могут содержать важную информацию о документе, такую как автор, дата создания, ключевые слова и другие атрибуты.

– /AcroForm: элемент, который является контейнером для данных о формах, если в PDF-документе содержатся интерактивные формы. Он хранит информацию о полях формы, а также о связанных с ними аннотациях. В контексте /AcroForm могут присутствовать такие элементы, как:

а) /Annot — общий контейнер для аннотаций в PDF-документе;

б) /Widget — подтип аннотации, который используется для описания конкретных элементов формы, таких как текстовые поля, кнопки, флажки и другие элементы интерфейса, с которыми может взаимодействовать пользователь.

Важной частью каталога является дерево страниц, которое описывает, как страницы связаны между собой. Каталог ссылается на объект /Pages, который является контейнером для страниц документа. В свою очередь, /Pages может включать другие объекты, такие как страницы, например, /Page или группы страниц. Это создаёт иерархию, которая позволяет эффективно управлять страницами и их ресурсами.

Каждая /Page описывает отдельную страницу документа. Этот объект содержит информацию о том, где размещены элементы на странице, как они ориентированы. Непосредственно с каждой страницей тесно связаны еще два компонента страницы:

– контент-стримы (/Contents) содержат инструкции для рендеринга текста, графики и других элементов на странице. Эти инструкции представляют собой последовательность команд, которые определяют, как отображать контент на странице. Контент-стримы могут включать в себя операции рисования текста, линий, кривых и других графических элементов, а также команды для вывода изображений и других медиа-ресурсов;

– словарь /Resources, который связывает страницы с ресурсами, необходимыми для рендеринга. Ресурсы, которые хранятся в /Resources, включают в себя:

а) /Font — шрифты, используемые на странице;

б) /XObject — объекты, такие как изображения и формы, которые могут быть вставлены в документ;

в) /ColorSpace — цветовые пространства, используемые для отображения графики.

1.2.5 Метаданные PDF

В современных информационных системах PDF-документы нередко выступают не просто контейнером для визуального представления страниц, но и хранилищем важной служебной информации и средств защиты содержимого. Метаданные в PDF служат для хранения сведений о документе — его названии, авторе, дате создания и других атрибутах, которые облегчают поиск, каталогизацию и автоматическую обработку. В современном PDF существуют два основных способа хранения метаданных: через простой словарь /Info и через более гибкий XML-блок XMP.

Словарь /Info — это начальный и самый простой контейнер метаданных, который всегда присутствует в PDF (если документ хоть раз сохранялся с заполненными полями). Он хранится не в самом контенте страниц, а в трейлере, и представляет собой набор пар «ключ → значение», где:

– /Title - заголовок документа. Отображается в заголовке окна ридера и используется при создании ярлыка или имени по умолчанию для сохранения файла.

– /Author - имя автора или организации, ответственной за содержание. Часто берётся из свойства «Автор» в текстовом процессоре и позволяет быстро находить документы по авторству.

– /Subject - краткое описание темы или назначения документа. Полезно для каталогизации и при автоматическом групповом отображении списка PDF.

– /Keywords - список ключевых слов, разделённых запятыми или точкой с запятой. Используется для поиска и фильтра по тематике.

– /Creator - программа, в которой исходный документ был создан (например, Microsoft Word, Adobe InDesign). Позволяет определить исходный формат и версию ПО.

– /Producer - программа или библиотека, которая собственно сгенерировала PDF (Adobe Acrobat Reader, LibreOffice, ABBYY FineReader и сторонние SDK). Помогает отследить цепочку обработки файла.

– /CreationDate - дата и время первого сохранения PDF, в формате D:YYYYMMDDHHmmSSOHH'mm' (ISO-подобная строка с указанием часового пояса).

– /ModDate - дата и время последнего изменения. Если документ редактировался несколько раз, именно это поле обновляется при каждом сохранении.

Кроме этих стандартных полей, PDF не запрещает добавлять собственные ключи в словарь /Info, но рекомендовано придерживаться официальных названий, чтобы обеспечить совместимость. При чтении и изменении /Info важно помнить, что каждое поле хранится в виде PDF-строки (без структурированного вложенного формата), тем самым накладывая ограничения на длину, кодировку и наличие управляющих символов.

Чтобы преодолеть эти ограничения, в PDF введён механизм XMP (Extensible Metadata Platform). Внутри потока /Metadata располагается XML-блок, построенный по схеме RDF, где можно указывать не только базовые свойства, но и любые пользовательские поля, создавать собственные пространства имён и связывать документ с внешними системами (например, корпоративными реестрами или облачными архивами). Помимо этого, XMP-метаданные позволяют:

- описывать сложные структуры (авторы, роли, лицензии);
- гарантировать взаимосвязь с внешними системами через URI;
- сохранять полное представление Unicode-строк.

XMP-блок обычно расположен сразу после трейлера и выглядит как `<x:xmpmeta>...</x:xmpmeta>`. При наличии XMP современные приложения предпочитают читать именно его, оставляя словарь /Info для совместимости с более старым ПО. Важным остается помнить, что при реализации приложения редактора PDF нужно обеспечить синхронизацию между этими двумя уровнями метаданных. Изменение значения в словаре /Info должно сопровождаться правкой XMP-блока, иначе возникнет рассинхронизация, что может привести к проблемам при проверке цифровой подписи или при проверке на соответствие стандарту PDF/A. Для стандарта PDF/A наличие корректно оформленного XMP-блока не просто рекомендовано, а обязательно: он обеспечивает долговременную читаемость без потери информации о версии формата и составе документа.

Практическая значимость метаданных выходит за рамки простого описания:

- автоматизация: по XMP-свойствам системы мгновенно группируют и сортируют сотни тысяч документов;
- поиск: поисковые движки корпоративных порталов учитывают поля и расширенные теги XMP, что ускоряет выдачу нужных файлов;

– контроль версий: в XMP можно хранить историю правок, авторов и причину изменений;

– безопасность и целостность: цифровые подписи (PKCS#7/CMS) обычно включают именно XMP-метаданные в область вычисления хеша, что гарантирует неизменность критичных для подлинности полей.

1.2.6 Безопасность PDF

Современный формат PDF сочетает в себе мощные механизмы криптографической защиты, включая шифрование и цифровые подписи, а также механизмы, влияющие на безопасность в процессе изменений, такие как инкрементальные обновления. Эти технологии позволяют не только надёжно ограничивать доступ к конфиденциальной информации, но и обеспечивать целостность и подлинность документа. Благодаря гибкой архитектуре, PDF может адаптироваться к различным сценариям использования от защиты персональных данных до юридически значимого электронного документооборота.

1.2.6.1 Инкрементальные обновления (append-update)

Одной из ключевых особенностей формата PDF является возможность внесения изменений в документ без полной перезаписи всего файла. Этот механизм, известный как инкрементальное обновление, представляет собой метод добавления новых данных в конец существующего файла, что позволяет сохранять старые данные и не удалять их, а помечать как неактивные. Это означает, что при изменении или добавлении элементов в документ сохраняется его исходная структура, а изменения добавляются в отдельной секции в конце файла.

На рисунке 4 показан механизм инкрементального обновления, который включает несколько этапов. Когда в документ вносятся изменения, создается новая таблица перекрестных ссылок (xref) и обновляется трейлер файла, который управляет метаданными о структуре документа. Вместо того чтобы переписывать весь файл, как это происходит при обычном сохранении, только

новые или измененные объекты добавляются в файл, что значительно экономит время и ресурсы при сохранении документа.

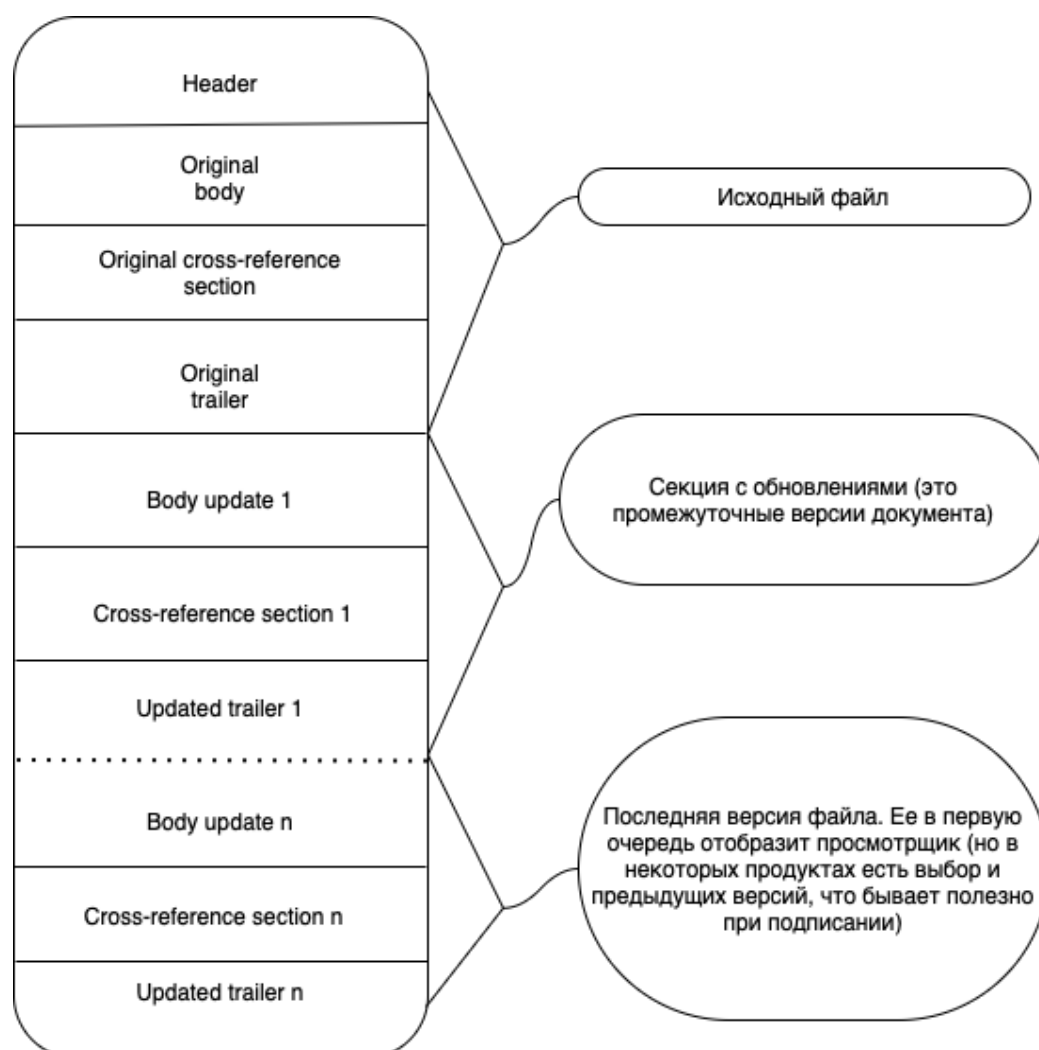


Рисунок 4 – Механизм инкрементального обновления PDF

Такой подход имеет важные практические преимущества. Он позволяет редактировать документы в условиях, когда полная перезапись файла невозможна или нежелательна. Особенно это важно при работе с документами через веб-сервисы (HTTP), где минимизация времени загрузки становится критическим фактором для повышения производительности и ускорения обмена данными. Также этот подход полезен при внедрении PDF в операционные системы, где файлы могут быть встроены в другие приложения и требуют быстрого доступа без долгих задержек.

Особенность инкрементальных обновлений заключается в том, что при внесении изменений в документ старые объекты не исчезают, а становятся помеченными как устаревшими и больше не используются. Это позволяет сохранить историческую информацию о всех изменениях, сделанных в документе, что может быть полезно для анализа и восстановления старых версий документа. Например, при добавлении новой аннотации в документ, новая аннотация записывается как отдельный объект, и затем обновляется страница, на которой эта аннотация появляется. В xref-таблице указывается смещение новых объектов в файле, что позволяет эффективно управлять доступом к ним и ускоряет процесс открытия и работы с документом.

Таким образом, инкрементальное обновление не только повышает производительность работы с PDF, но и поддерживает гибкость при обработке документов в различных приложениях, обеспечивая высокий уровень совместимости и удобства при редактировании. Но несмотря на его преимущества, такой подход может представлять определенные риски с точки зрения безопасности и целостности данных, так как старые, неактивные объекты остаются в файле и могут быть использованы для манипуляций или скрытого изменения содержания документа.

1.2.6.2 Шифрование и криптографическая защита

Одной из ключевых возможностей современных PDF-файлов является защита содержимого посредством шифрования и гибкого управления правами доступа. Уже в версии PDF 1.3 появилась поддержка стандартного шифрования RC4, а начиная с PDF 1.6 был внедрён более надёжный алгоритм AES-128. В дальнейшем, в PDF 2.0, криптографическая защита была значительно усилена: формат получил поддержку AES-256 в режиме CBC с PKCS#7-дополнением и унифицированный подход к реализации всех криптографических операций. Это позволило PDF окончательно перейти от устаревших и уязвимых схем к современным и стойким решениям.

Исторически шифрование в PDF строилось на поточных шифрах RC4 с длиной ключа 40 или 128 бит. RC4 благодаря своей скорости и простоте обеспечивал базовую защиту потоков и строк, однако со временем продемонстрировал серьёзные уязвимости, особенно при использовании коротких ключей. В результате произошло постепенное смещение в сторону блочных шифров. AES-CBC с обязательным инициализационным вектором для каждого потока стал не просто предпочтительным, а необходимым требованием. Такое решение обеспечило значительно более высокую криптографическую стойкость, особенно при использовании 256-битных ключей, и сделало PDF-файлы устойчивыми к целому ряду известных атак.

Сама защита документа задаётся через словарь /Encrypt в трейлере или в отдельном косвенном объекте, на который он ссылается. В этом словаре указываются:

- /Filter - тип используемого шифрования (обычно /Standard для RC4/AES);
- /V и /R - версия и алгоритмический раунд, влияющие на длину ключей и способ их генерации;
- /O и /U - так называемые «владелец» и «пользователь» ключи, по которым проверяется введённый пароль;
- /P - битовая маска прав доступа, где каждый бит разрешает или запрещает операции (печать, копирование, изменение, добавление аннотаций и др.).

Дополнительную гибкость формату PDF придаёт механизм Crypt Filters — настраиваемых цепочек шифрования. С их помощью можно указать, какие части документа шифруются, каким алгоритмом и в каком режиме. Это позволяет, например, оставить метаданные открытыми для поисковой индексации, одновременно полностью защищая контент-потоки и графику. Таким образом, документ может адаптироваться под различные

сценарии использования, включая частичный доступ или селективное раскрытие данных.

При первом открытии зашифрованного PDF на основе переданного пользователем пароля и встроенных ключей вычисляется master key, который затем модифицируется для каждого косвенного объекта в документе (с учётом его номера и поколения). Это позволяет шифровать и дешифровать строки и потоки независимо, обеспечивая высокую степень защиты. Если введён «владелец» пароль, пользователь получает полный доступ - тогда как «пользовательский» пароль позволяет выполнить только те действия, которые не запрещены флагами /P: печать (с высоким или низким качеством), копирование текста и графики, изменение содержимого, работу с формами и аннотациями.

Помимо базового шифрования содержимого, формат PDF 2.0 ввёл возможность шифровать метаданные и XMP-блок - поток /Metadata, где хранится XML с расширяемой информацией о документе. Это особенно важно, если метаданные содержат конфиденциальные ссылки, пользовательские поля или описания, которые не должны быть доступны без авторизации. В словаре /Encrypt можно указать, какие именно потоки зашифрованы, а какие остаются открытыми, что позволяет гибко настраивать политику безопасности: например, оставить доступными только поля /Info, но скрыть полный XMP-блок.

Невозможно не отметить и то, как шифрование взаимодействует с инкрементальными обновлениями. Каждый новый фрагмент XRef и трейлера может добавлять или менять словарь /Encrypt. Для корректного распознавания всех версий шифрования в трейлере указываются ключи /Prev (смещение предыдущей версии XRef) и /Encrypt — таким образом, при сохранении новой версии документа приложение по цепочке сможет взять правильные параметры шифрования.

В итоге грамотно настроенное шифрование и управление правами в PDF позволяет надёжно защищать конфиденциальную информацию, сохраняя при этом гибкость доступа для разных групп пользователей.

1.2.6.3 Цифровые подписи и целостность

Цифровая подпись в PDF основана на контейнере PKCS#7/CMS и позволяет одновременно подтвердить авторство документа и зафиксировать его неизменность. При сохранении подписи создаётся объект со словарём /Signature, который содержит следующие ключи:

- /Filter и /SubFilter – указывают формат подписи (обычно /Adobe.PPKLite и adbe.pkcs7.detached);

- /ByteRange – четыре числа, задающие диапазоны байтов в файле, покрываемые подписью (область подписи сама при подсчёте хеша игнорируется);

- /Contents – бинарный блок PKCS#7 с зашифрованным хешем документа и цепочкой сертификатов;

- /M, /Name, /Location – время подписи, имя подписанта и, при желании, место или причина.

При проверке целостности, с цифровыми подписями, система собирает все байты из указанных в /ByteRange областей, вычисляет хеш (чаще всего SHA-256), а затем с помощью публичного ключа из сертификата сверяет этот хеш с тем, что заложен в /Contents. Проверяется валидность сертификата подписанта (срок действия, доверенные корневые центры, опционально OCSP/CRL-ответы). Если хоть один байт документа изменён, хеш не совпадёт – подпись станет недействительной.

Для юридически значимых документов недостаточно один раз проверить подпись – нужно гарантировать верификацию через годы. PDF поддерживает встраивание OCSP/CRL-ответов прямо в поток /Contents или в отдельный контейнер DSS (Document Security Store). Это позволяет подтвердить подпись без внешних запросов, а также возможность отмечать документы как «утверждённый» или «для отдельных разделов».

Сами подписи могут быть как «невидимыми» (хранятся исключительно в /Contents и ни на что не влияют визуально), так и «видимыми»: для последних создаётся аннотация /Widget, в которой задаётся положение на странице, рамка и графический штамп или текст подписи. При этом техническая схема проверки остаётся той же – только к подписи добавляется визуальный компонент, который пользователь видит при просмотре.

PDF поддерживает размещение нескольких подписей в одном файле: каждая новая подпись вносится через инкрементальное обновление и получает собственный объект /Signature с новым /ByteRange. При проверке приложения идут по цепочке подписей в порядке их добавления, последовательно считая и сверяя хеши для каждого диапазона. Благодаря этому можно, например, застраховать документ сначала подписью автора, затем подписью проверяющего, а в конце – приёмщик. Если кто-то что-то изменит, можно легко понять, на каком этапе это произошло.

1.3 Обзор существующих инструментов и библиотек для работы с PDF

1.3.1 RUPS (iText)

Инструмент **RUPS (Reading and Updating PDF Syntax)**, представленный на рисунке 5, разрабатывается компанией iText Group NV и представляет собой Java-приложение на базе Swing, позволяющее визуально «заглянуть внутрь» PDF-документа и последовательно просмотреть все его косвенные объекты, словари и потоки содержимого. RUPS строится поверх ядра iText и обеспечивает удобную навигацию по дереву объектов, позволяя раскрывать вложенные объекты /Catalog, /Pages, /XObject, а также XFA-спецификации. Кроме того, он поддерживает открытие зашифрованных PDF-файлов в режиме «read only» при вводе user-пароля, что гарантирует неизменность оригинала документа. Однако при всех своих возможностях RUPS остаётся достаточно тяжеловесным: он

требует установки JRE и обладает сравнительно медленным временем запуска, что ограничивает его применение для массовой пакетной обработки больших документов. Также инструмент ориентирован прежде всего на ручную работу в GUI и предлагает лишь скудную поддержку автоматизации через командную строку. Отсутствие полноценного API вынуждает пользователей переключаться между несколькими утилитами для интеграции в свои конвейеры обработки.

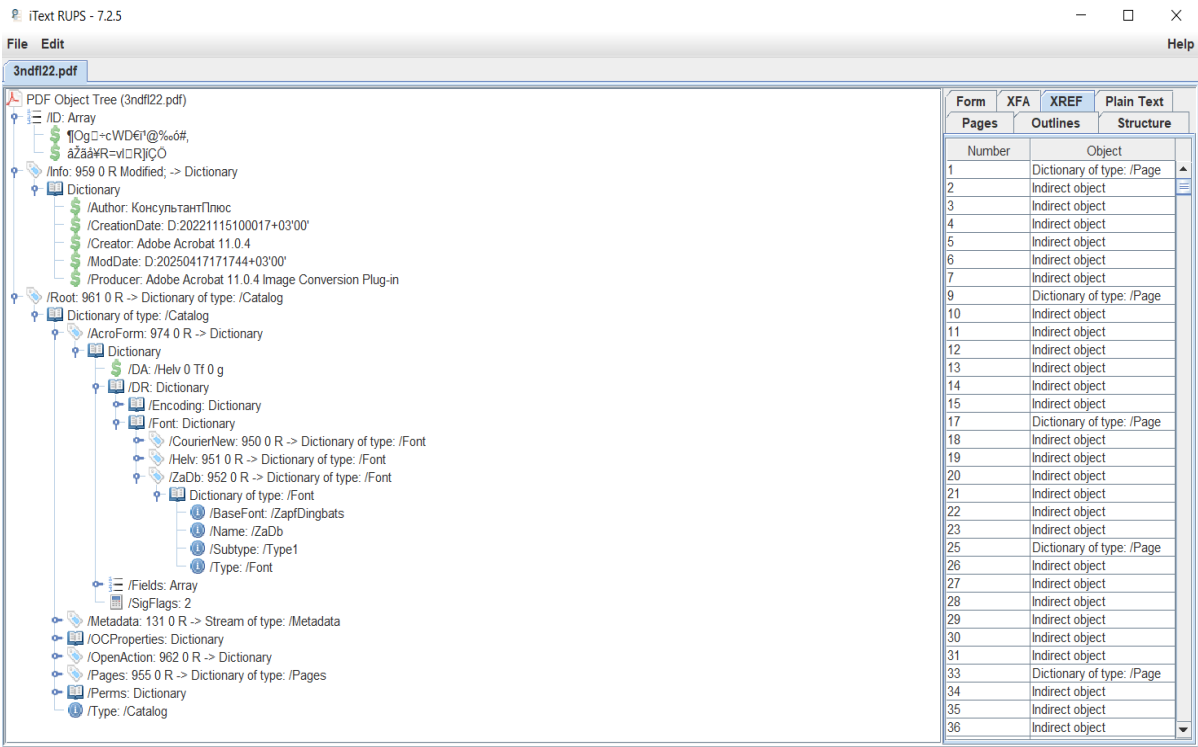


Рисунок 5 – Приложение RUPS от iText

1.3.2 PDFDebugger (PDFBox)

Инструмент [PDFDebugger](#) входит в состав утилит Apache PDFBox и представляет собой отдельное приложение для интерактивного анализа внутренней структуры PDF-документов. При старте PDFDebugger отображается дерево косвенных объектов (COSObjects), где каждый узел соответствует отдельному объекту PDF, будь то словарь, массив или поток содержимого. Одновременно в соседней панели выводится расшифрованное содержимое выбранного элемента: словари показываются в читаемом виде, а потоки

автоматически декодируются. К безусловным плюсам PDFDebugger относится интуитивно понятный графический интерфейс, при котором всё управление сводится к кликам по узлам дерева, без необходимости углубляться в программирование или скрипты. Инструмент одинаково хорошо работает на разных платформах и поддерживает открытие зашифрованных PDF-файлов в режиме только для чтения при вводе пароля пользователя. В то же время при работе с большими документами (особенно содержащими множество изображений и сложных потоков) может проявляться заметная задержка при загрузке узлов и переключении между ними — это связано с кэшированием и декодированием ресурсов «на лету», что требует значительных объёмов памяти и CPU. Экспорт данных ограничен стандартными способами «сохранить как...» или копированием в буфер, а отчёты в привычных форматах CSV/JSON не предусмотрены, что осложняет автоматизацию пакетного анализа.

Интерфейс приложения PDFDebugger представлен на рисунке 6.

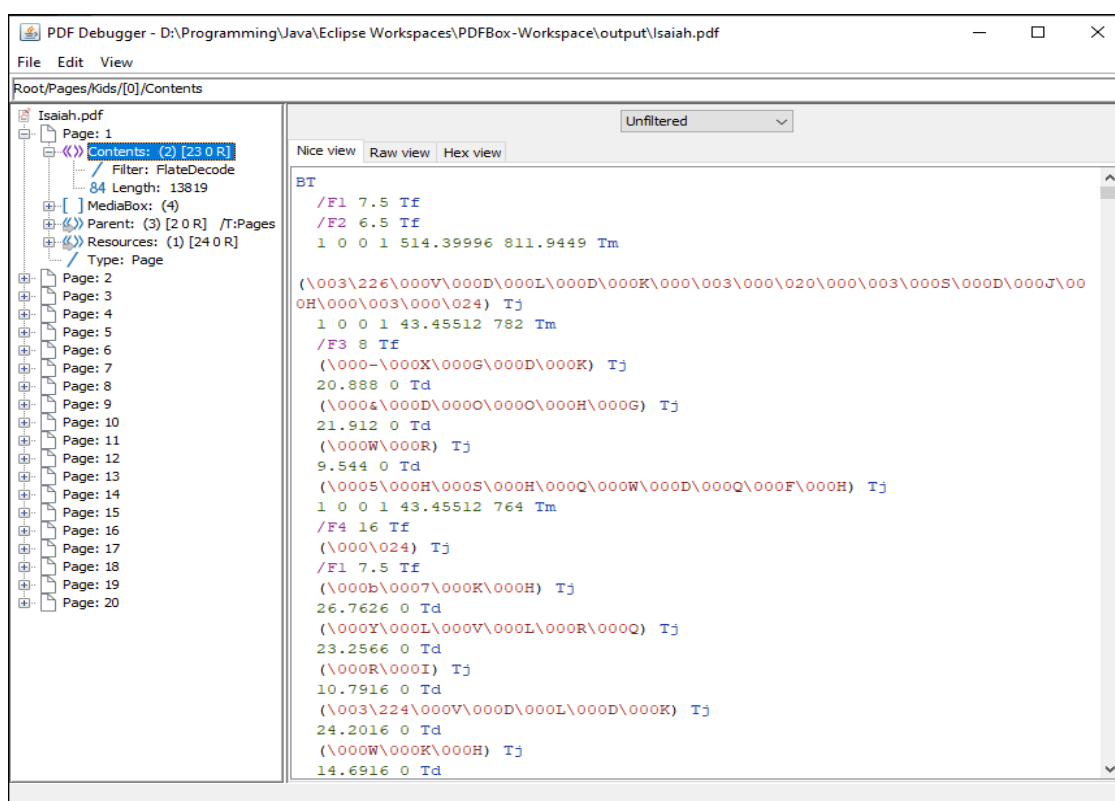


Рисунок 6 – Приложение PDFDebugger

1.3.3 IPA

Interactive PDF Analysis (IPA) — проект с открытым кодом под GPL-2.0, размещённый на GitHub, создавался как специализированное средство для безопасности и цифровой криминалистики PDF. В графическом интерфейсе IPA, представленном на рисунке 7, реализовано дерево объектов PDF с возможностью экспорта «сырых» байтов потока, фильтрации объектов при помощи YARA-правил и поиска вредоносных паттернов, что делает его инструментом первой необходимости при анализе подозрительных документов. В основе разбора pdf лежит Rust-библиотека pdf-rs, а для GUI используется egui, что обеспечивает приложению нативную скорость и относительную лёгкость без JVM. С другой стороны, IPA пока не поддерживает дешифровку защищённых PDF и не интегрируется с внешними автоматизированными конвейерами анализа — всё делается вручную через интерфейс, что ограничивает сценарии массового сканирования больших объёмов документов. Кроме того, проект относительно молод, что накладывает риски неполного функционала и нерегулярных обновлений.

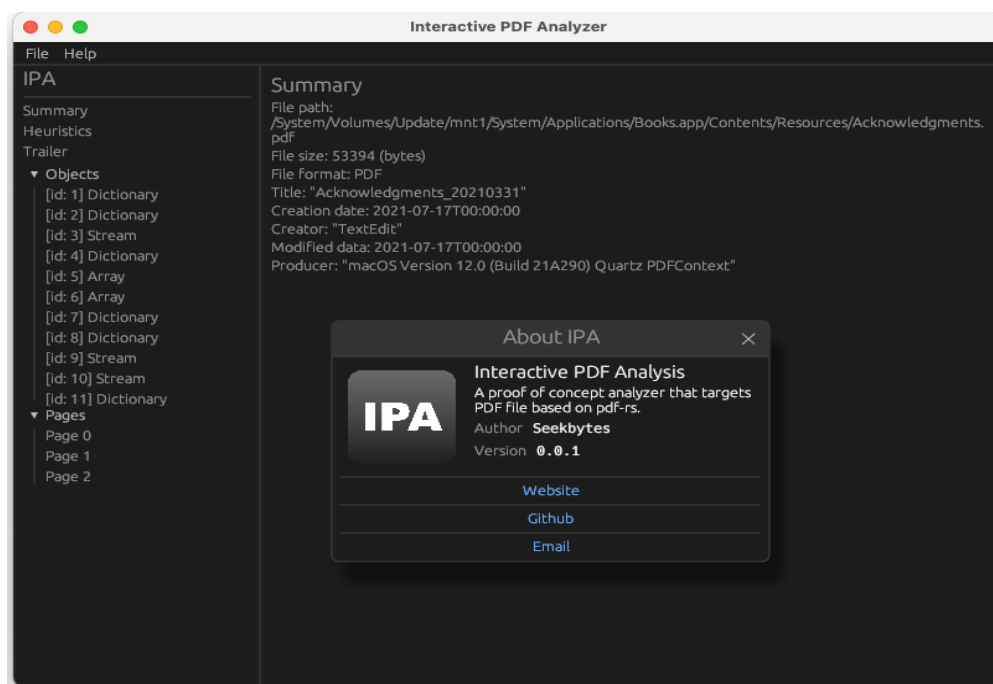
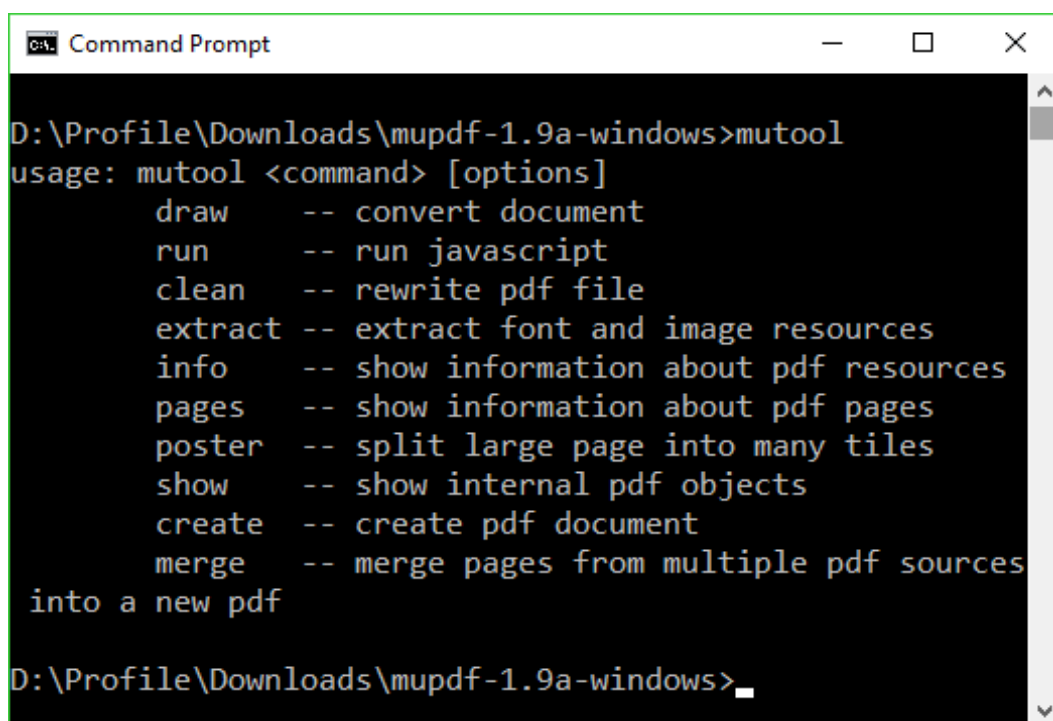


Рисунок 7 – Приложение IPA

1.3.4 MuPDF

MuPDF – это высокопроизводительная библиотека на языке C, разработанная компанией Artifex Software, предназначенная для разбора и рендеринга PDF, XPS и EPUB документов. Графическое приложение, `mupdf-gl` предоставляемое MuPDF (рисунок 8), обеспечивает лишь визуализацию страниц без какого-либо доступа к внутренним объектам или потокам файла. Все средства для глубокого анализа синтаксиса PDF реализованы в консольной утилите `mutool`: команда `mutool show` выводит указанные косвенные объекты и их потоки в текстовом виде, автоматически декодируя содержимое и заменяя непечатные символы точками. При помощи параметра `clean -d` утилита переписывает PDF в распакованном формате, что упрощает чтение потоков в обычном текстовом редакторе. Такая архитектура гарантирует минимальный размер кода и высокую скорость пакетной обработки, однако полная структура документа доступна только через командную строку, что снижает удобство для пользователей, не знакомых с CLI.



```
Command Prompt

D:\Profile\Downloads\mupdf-1.9a-windows>mutool
usage: mutool <command> [options]
    draw    -- convert document
    run     -- run javascript
    clean   -- rewrite pdf file
    extract -- extract font and image resources
    info    -- show information about pdf resources
    pages   -- show information about pdf pages
    poster  -- split large page into many tiles
    show    -- show internal pdf objects
    create  -- create pdf document
    merge   -- merge pages from multiple pdf sources
into a new pdf

D:\Profile\Downloads\mupdf-1.9a-windows>_
```

Рисунок 8 – Приложение mutool

1.3.5 Сравнительный анализ существующих инструментов и библиотек

В процессе исследования существующих аналогов было обнаружено, что ни один из них не обладает полным набором необходимых функций. В таблице 1 приведены основные критерии анализа приложений-аналогов.

Таблица 1 – Достоинства и недостатки приложений-аналогов

Поддержка функционала	RUPS	PDFDebugger	MuPDF	IPA
Таблица объектов с типами, позициями	-	-	-	-
Дерево объектов	+	+	-	+
Поиск и отображение ссылок (Refs)	+	+	+/-	+
Просмотр содержания потока (text / hex)	-	-	+/-	-
Работа с ObjStm и XRef stream	+	+/-	+	+
Поддержка дешифровки (шифрование)	+	+	+	-
Сохранение потока в файл / Копирование в буфер	-	-	-	+/-
Сводка по документу (metadata, trailer)	+	-	-	+
Подходит для непрограммистов	+	+	-	+

1.4 Выбор технологий и средств разработки приложения

В данном разделе будут описаны технологии и средства разработки, использованные при разработке программного обеспечения шифрования файлов.

1.4.1 Выбор языка программирования

Существует множество языков программирования, которые могут использоваться для написания приложения и работы с содержимым файла.

Рассмотрим несколько из них:

1.4.1.1 Язык программирования C/C++

C и C++ – это «ближайшие к железу» языки, на которых написаны такие движки, как MuPDF и Poppler. Их сила в прямом доступе к памяти и максимально высокой скорости работы с большими бинарными файлами. Однако необходимость ручного управления памятью, низкоуровневый синтаксис и обилие указателей усложняют разработку и отладку. Для проекта, где важно быстро получить наглядный инструмент и избежать тонкой настройки сборки под разные платформы, C/C++ требовал бы слишком больших усилий и времени на поддержку.

1.4.1.2 Язык программирования Java

Java действует поверх виртуальной машины, что даёт кроссплатформенность и автоматическое управление памятью. Библиотеки iText и Apache PDFBox умеют генерировать и разбирать PDF в самых сложных сценариях, и на их основе построены многие корпоративные решения. Но JVM-over-head, особенности лицензирования (особенно в коммерческих версиях iText) и относительная тяжеловесность Java-приложений на Windows делают её менее предпочтительной, когда приоритетом является лёгкое развёртывание и высокая отзывчивость UI на одной платформе.

1.4.1.3 Язык программирования Python

Python известен лаконичностью и скоростью прототипирования: пакеты PyPDF2, PDFMiner или pdfcrowd позволяют в несколько строк начать работу с

содержимым PDF. Тем не менее интерпретируемая природа языка и глобальная блокировка потоков (GIL) ограничивают производительность при обработке больших потоков данных, а поддержка некоторых низкоуровневых фильтров и шифров в существующих библиотеках пока менее зрелая, чем в «компилируемых» решениях.

1.4.1.4 Язык программирования C#

C# сочетает в себе управляемую среду CLR, удобный синтаксис и продвинутые инструменты разработки: LINQ, обобщённые коллекции, мощные дебаггеры и профилировщики в Visual Studio. На платформе .NET уже созданы зрелые проекты для работы с PDF — PDFsharp и PdfiumViewer показывают, что C# отлично подходит для разбора, отображения и манипуляции PDF-структурами без серьёзных потерь в производительности. При этом язык позволяет писать код быстрее, чем на C/C++, и при этом не теряет в производительности, как в случае Python или Java. Благодаря встроенной поддержке WinForms и WPF вместе с Visual Studio удалось быстро создать удобный интерфейс для анализа и визуализации структуры PDF.

В экосистеме .NET существуют два направления: классический .NET Framework, проверенный десятилетиями, с богатейшей поддержкой WinForms, WCF, COM-Interop и интеграцией в Microsoft Windows, и более молодой .NET Core, лёгкий и кроссплатформенный, но на момент старта проекта ещё не охвативший все Windows-специфичные технологии.

.NET Framework предоставляет полный набор дизайнеров форм, надёжные профилировщики, обширные сторонние библиотеки и что особенно важно, эта программная платформа гарантированно установлена на большинстве рабочих машин и обладает богатой поддержкой Windows-специфичных технологий, что устраняло необходимость дополнительной настройки и установки зависимостей. Это значительно ускоряет процесс разработки и повышает стабильность работы приложения в целевой среде.

Таким образом, исходя из баланса производительности, удобства разработки и стабильности, была выбрана связка C# и .NET Framework как оптимальное решение для создания полноценного инструмента глубокого анализа PDF-документов.

1.4.2 Выбор среды разработки

1.4.2.1 JetBrains Rider

Кроссплатформенная IDE с интеллектуальным анализом кода и встроенной подсветкой архитектурных проблем. Несмотря на высокую производительность работы с C#, для учебного проекта потребовалось бы дополнительное лицензирование и настройка, а также привыкание к пользовательскому интерфейсу, отличному от привычной Visual Studio.

1.4.2.2 Visual Studio Code

Легковесный редактор с расширениями для C# и разными плагинами. Он отлично подходит для небольших скриптов и веб-разработки, но не предоставляет «из коробки» визуального конструктора форм и полноценного отладчика для классических desktop-приложений на .NET Framework.

1.4.2.3 SharpDevelop

Ранее популярная бесплатная альтернатива Visual Studio, хорошо справляется с базовыми задачами .NET-разработки. На сегодняшний день обновления выходят крайне редко, и поддержка современных шаблонов проектов и пакетов NuGet уже ограничена, что создаёт риски при использовании актуальных библиотек.

1.4.2.4 Visual Studio

Visual Studio остаётся основной средой для разработки приложений на .NET Framework благодаря глубокому набору инструментов: визуальным редакторам форм, мощному отладчику, встроенным профилировщикам, гибкой работе с NuGet и интеграции с системами контроля версий. Важное остается за тем, что данная IDE располагает удобным интерфейсом и высокой производительностью по сравнению с вышеописанными средами. Главное

преимущество — возможность быстро и надёжно создавать, и отлаживать как логику разбора PDF, так и интерфейс анализа в единой среде.

Таким образом, выбор пал на Visual Studio, которая остаётся ведущей IDE для разработки на .NET Framework. Данная среда разработки предлагает мощный и удобный визуальный конструктор форм, продвинутый отладчик, её глубокая интеграция с системами контроля версий обеспечивают комфортную и эффективную разработку пользовательского интерфейса. Использование единой среды позволит существенно ускорить процесс создания приложения.

1.4.3 Выбор платформ реализации графического интерфейса

1.4.3.1 Universal Windows Platform

UWP — платформа для универсальных приложений Windows Store с современными контролами и безопасной моделью развёртывания. Подходит для приложений из Microsoft Store, но не всегда соответствует классическому десктопному сценарию лабораторных и студенческих утилит.

1.4.3.2 Windows Presentation Foundation

WPF - фреймворк с декларативным описанием UI на XAML и продвинутой системой привязок. Для небольшого инструмента с традиционным окном анализа требует освоения нового подхода к разметке и MVVM-паттерна.

1.4.3.3 Avalonia

Кроссплатформенный аналог WPF, позволяющий запускать одно и то же приложение на Microsoft Windows, Linux и macOS. Хотя дизайн на XAML знаком, стабильность сборок и поддержка пакетного менеджмента под .NET Framework были на момент старта проекта недостаточно зрелыми.

1.4.3.4 WinForms (.NET Framework)

Классическая библиотека для быстрого создания десктопных приложений с простым визуальным дизайнером. Позволила в короткие сроки реализовать таблицы объектов PDF и окна просмотра потоков без лишней сложности.

Для реализации GUI была выбрана платформа WinForms, которая благодаря простому визуальному дизайнеру и богатой экосистеме на

.NET Framework позволит в короткие сроки создать удобный и интуитивно понятный интерфейс.

1.5 Определение требований к разрабатываемому ПО

1.5.1 Требования к программной части

В рамках исследования на основе выбранных технологий и инструментов разработки необходимо создать полноценное приложение с графическим интерфейсом для глубокого анализа PDF-файлов на уровне низкоуровневых объектов. Приложение должно демонстрировать все прямые и вложенные объекты документа в табличном виде, отображать номера объектов, типы и подтипы (например, /Font, /Encoding, /XObject, /Metadata), позиции и длины потоков, а также информацию о родительских ObjStm и индексах внутри них. Пользователь должен иметь возможность:

- просматривать сводку документа: трейлер, фильтры, версии PDF, количество объектов и страниц;
- детализировать любой отдельный объект: отображать его словарь (Dictionary), массивы (Array), числовые и логические значения.
- открывать и декодировать поток (Stream) в текстовом или шестнадцатеричном представлении;
- работать с зашифрованными PDF-файлами: вводить пользовательский пароль, просматривать и расшифровывать защищённые сегменты;
- экспортировать данные: сохранять содержимое потока или полного Summary в файл, копировать произвольный текст или hex-код в буфер обмена;
- удобно фильтровать и искать объекты по номеру, типу или субтипу.

Реализация всего перечисленного функционала позволит специалистам, пользователям по информационной безопасности и разработчикам систем документооборота – проводить анализ PDF-документов, оперативно выявлять скрытый или изменённый контент, контролировать целостность подписанных и

зашифрованных блоков, а также интегрировать результаты в автоматизированные рабочие процессы.

1.5.2 Требования к аппаратуре и программному обеспечению

Для комфортной и стабильной работы приложения необходимо следующее:

– аппаратные требования:

а) процессор: многоядерный CPU с частотой не ниже 2,0 ГГц (рекомендуется Intel Core i3-8100 или эквивалент);

б) оперативная память: минимум 4 ГБ (для больших PDF-файлов — 8 ГБ и более);

в) место на диске: не менее 200 МБ свободного пространства для установки и временных файлов.

– программные требования:

а) ОС: Microsoft Windows 10 (x86 или x64), с актуальными обновлениями безопасности.

б) Платформа: .NET Framework начиная с версии 4,5;

в) IDE (для разработки и отладки): Visual Studio 2015 или новее с компонентом “.NET Desktop Development”;

г) Дополнительные компоненты: при работе с зашифрованными PDF-файлами рекомендуется наличие актуальных криптопровайдеров Windows.

Таким образом, использование такого приложения предоставляет визуализацию всех ключевых элементов PDF-документа, обеспечивает корректную работу с объектными потоками и XRef-stream, реализовывает дешифровку при вводе пароля и предоставить пользователю мощный и интуитивно понятный набор средств для аудита и анализа PDF-файлов любого уровня сложности.

2 Проектирование и Разработка ПО

2.1 Архитектура приложения

Выбор правильной архитектуры приложения играет критическую роль: от неё зависит не только надёжность и производительность, но и дальнейшая возможность масштабирования, поддержки и расширения функционала. Хорошо продуманная структура кода позволяет команде быстрее внедрять новые возможности, легко адаптироваться к изменениям спецификации PDF и сохранять качество при росте проекта.

Приложение представляет собой настольный инструмент для глубокого анализа PDF-файлов: от их валидации и расшифровки до детального просмотра каждого косвенного объекта, содержимого потоков и XObject-форм. Цель – дать пользователю полный контроль над внутренней структурой документа, не прибегая к сторонним редакторам, и наглядно отобразить связи между страницами, графикой, текстовыми фрагментами и метаданными. Для этого система разделена на три взаимосвязанных подсистемы, каждая из которых выполняет свою задачу, но при этом легко масштабируется и модифицируется благодаря применению проверенных паттернов проектирования.

					ПГУ1.090301.05.001 ПЗ	Лист
						37
Изм.	Лист	№ докум.	Подпись	Дата		

На рисунке 9 представлена UML-диаграмма вариантов использования, иллюстрирующая основные сценарии взаимодействия с системой представлена. Она демонстрирует основные функции: открытие PDF, просмотр структуры и объектов, декодирование потоков, отображение трейлера и кросс-ссылок, а также сохранение содержимого. Такая визуализация помогает понять, какие действия доступны пользователю и как они связаны между собой.

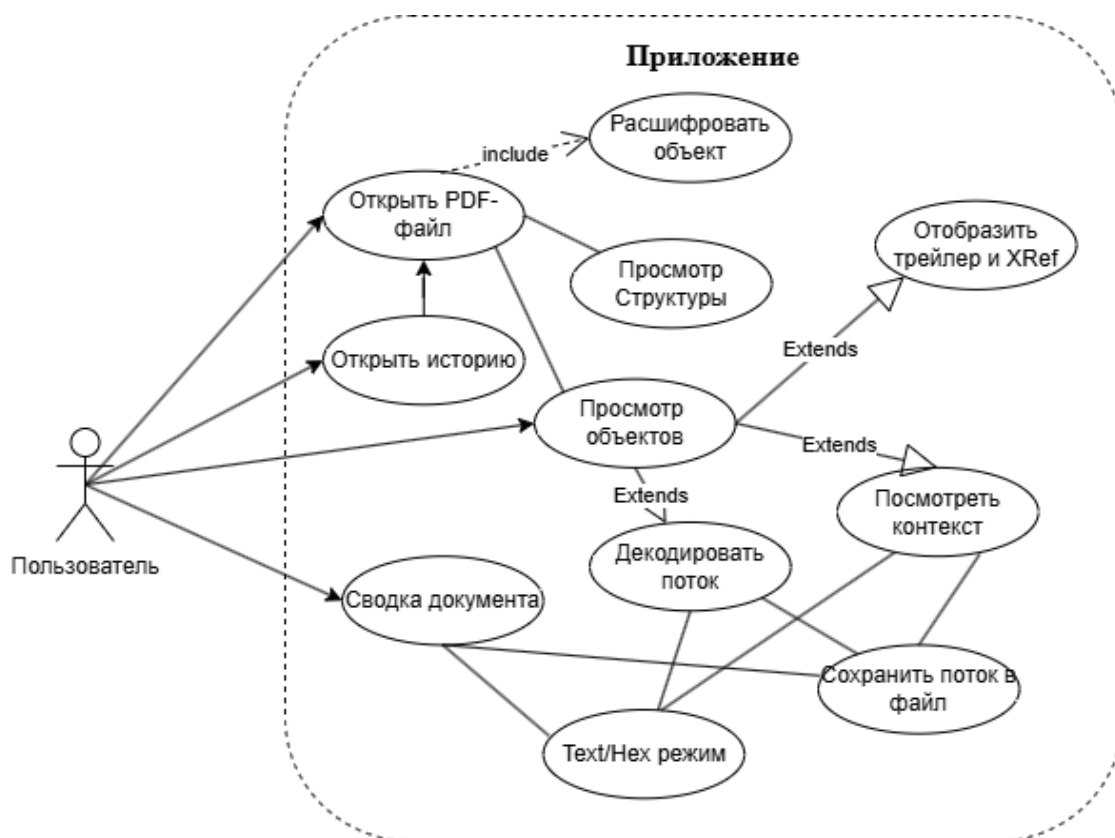


Рисунок 9 - UML диаграмма вариантов использования

В представленной диаграмме «*include*» применяется для тех действий, которые всегда должны выполняться, например, чтобы открыть файл, надо обязательно его прочитать и частично расшифровать. В свою очередь «*extend*» используют для дополнительных, расширяющих возможностей, которые не всегда нужны, например, если пользователь решает посмотреть детали конкретного объекта или сохранить поток в файл.

На рисунке 10 приведена UML-диаграмма, наглядно показывающая слои приложения и основные точки их взаимодействия. Она хорошо иллюстрирует слоистую архитектуру, её чёткое разделение ответственности, лёгкость сопровождения и возможность расширения функционала без кардинальных переделок.

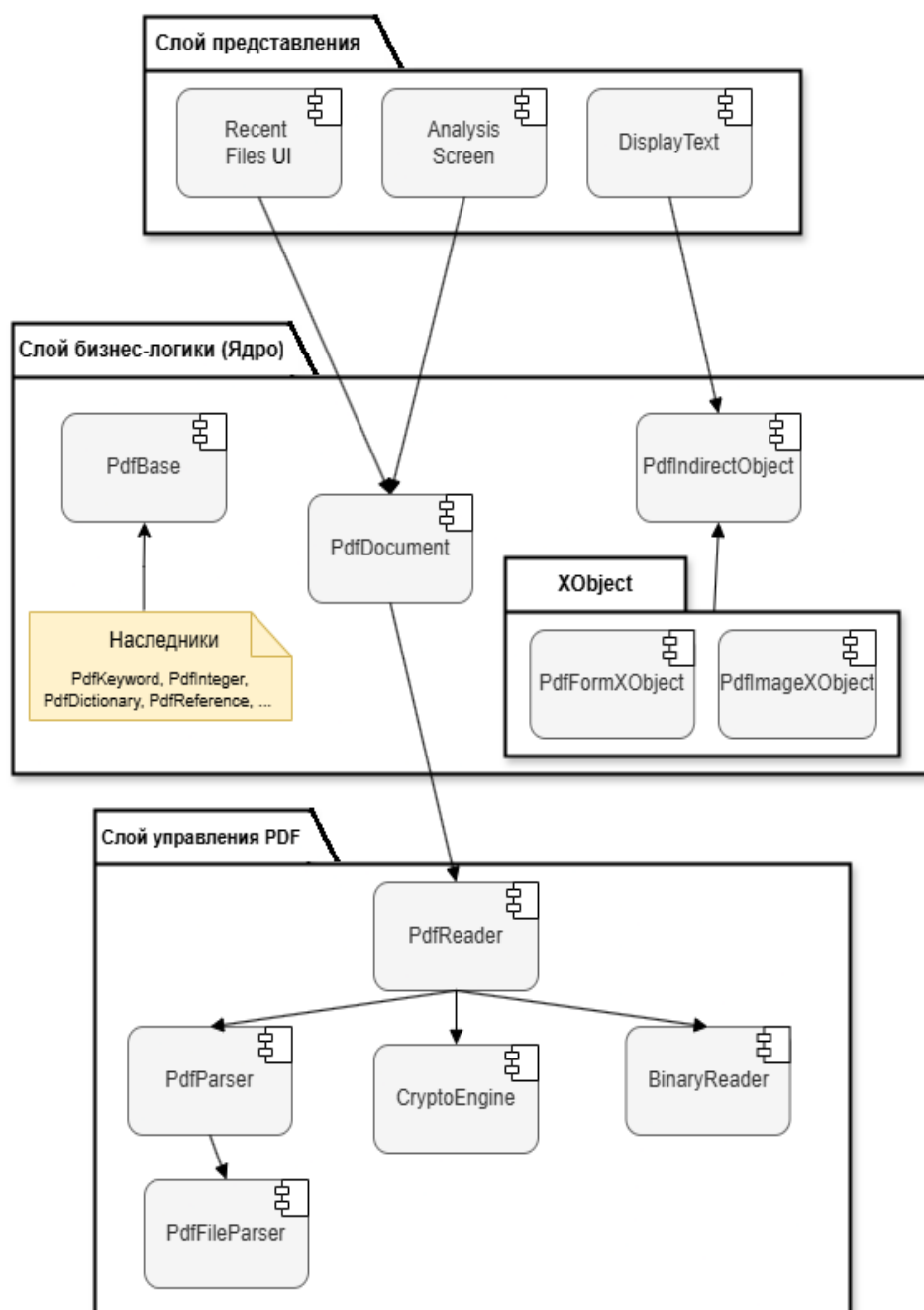


Рисунок 10 - UML диаграмма компонентов

2.1.1 Подсистема управления PDF

Подсистема реализует низкоуровневые механизмы работы с самим файлом PDF. Она включает компоненты PdfFileParser и PdfParser, которые отвечают за чтение байтов из потока, токенизацию и первичную классификацию элементов (числа, строки, словари, потоки). Ключевая задача – построить корректную таблицу кросс-ссылок (xref-таблицы и xref-стримы) и предоставить методы установки/сброса позиции в потоке.

2.1.2 Ядро разбора и анализа

Именно здесь сосредоточены основные алгоритмы разбора и анализа структуры PDF. Сердце приложения последовательно обрабатывает файл, создавая для каждого объекта модель PdfIndirectObject с полями: номер, поколение, смещение, тип (Dictionary, Stream, Free) и субтип (/Page, /XObject, /Annot и пр.). Сначала строится таблица кросс-ссылок (xref-таблицы и xref-стримы), затем для потоковых объектов применяются фильтры FlateDecode, LZWDecode. Потоки, помеченные как страницы, десериализуются в последовательность операторов PDF (BT/ET, Tj, Do и т. д.), что позволяет отобразить текст и графику внутри каждой страницы. Параллельно ядро собирает информацию о дереве страниц с наследованием ресурсов, а также вытаскивает словарь /Info и XMP-метаданные.

Построение дерева страниц с наследованием ресурсов (/Pages → /Page) происходит по принципам Composite: каждый узел может быть либо контейнером страниц, либо листом, и по спецификации PDF, когда рекурсивно «наследуются» словари ресурсов, пока не будет достигнут конкретный лист.

2.1.3 Подсистема визуализации и интерактивного анализа

В интерфейсе применён упрощённый Model-View-Controller (MVC):

- Model — это PdfDocument с таблицей объектов и потоками;
- View — WinForms-экраны AnalysisScreen и DisplayText;

– Controller — события UI (Open, View, Save) и команды, реализованные через Command Pattern, что позволяет легко добавлять новые действия (например, «Export to XML» или «Search in Contents»).

Главное окно с DataGridView показывает все объекты: их номера, типы, субтипы, смещения и размеры потоков. Пользователь может сортировать, фильтровать и выбирать строки – при этом Observer Pattern обеспечивает мгновенное обновление статуса кнопок «View Stream»/«View Contents» в зависимости от типа выбранного объекта.

Окно «Object Summary» демонстрирует словарь и метаданные объекта, а при переходе в режим просмотра потока («Stream», «Hex», «Contents») применяется State Pattern: каждый режим имеет собственную логику рендеринга и разбиения на строки или шестнадцатеричный дамп, но все они работают через единый интерфейс отображения.

2.2 Реализация приложения

При разработке приложения был выбран подход, использующий обработку PDF-документов на уровне структуры их объектов. Каждый элемент документа, будь то текст, изображение или метаданные, имеет свою структуру, и задача приложения заключается в правильном извлечении и интерпретации этих данных. Важным аспектом является работа с шифрованием, которое позволяет обрабатывать документы, защищённые от несанкционированного доступа. В дальнейшем будет рассматриваться структура PDF-документа, а также как приложение выполняет разбор файла, расшифровку и извлечение данных.

На диаграмме, представленной на рисунке 11, изображена схема алгоритма анализа PDF документа, которая демонстрирует процесс реализации приложения последовательно. Схема структурирует весь процесс в виде последовательных этапов, начиная с открытия и проверки PDF-файла, поиска трейлера и таблицы xref, и заканчивая извлечением объектов и созданием отчетов. Прежде чем перейти к разбору реализации приложения, эта диаграмма помогает наглядно

понять, как оно работает, шаг за шагом, обрабатывая различные элементы документа.

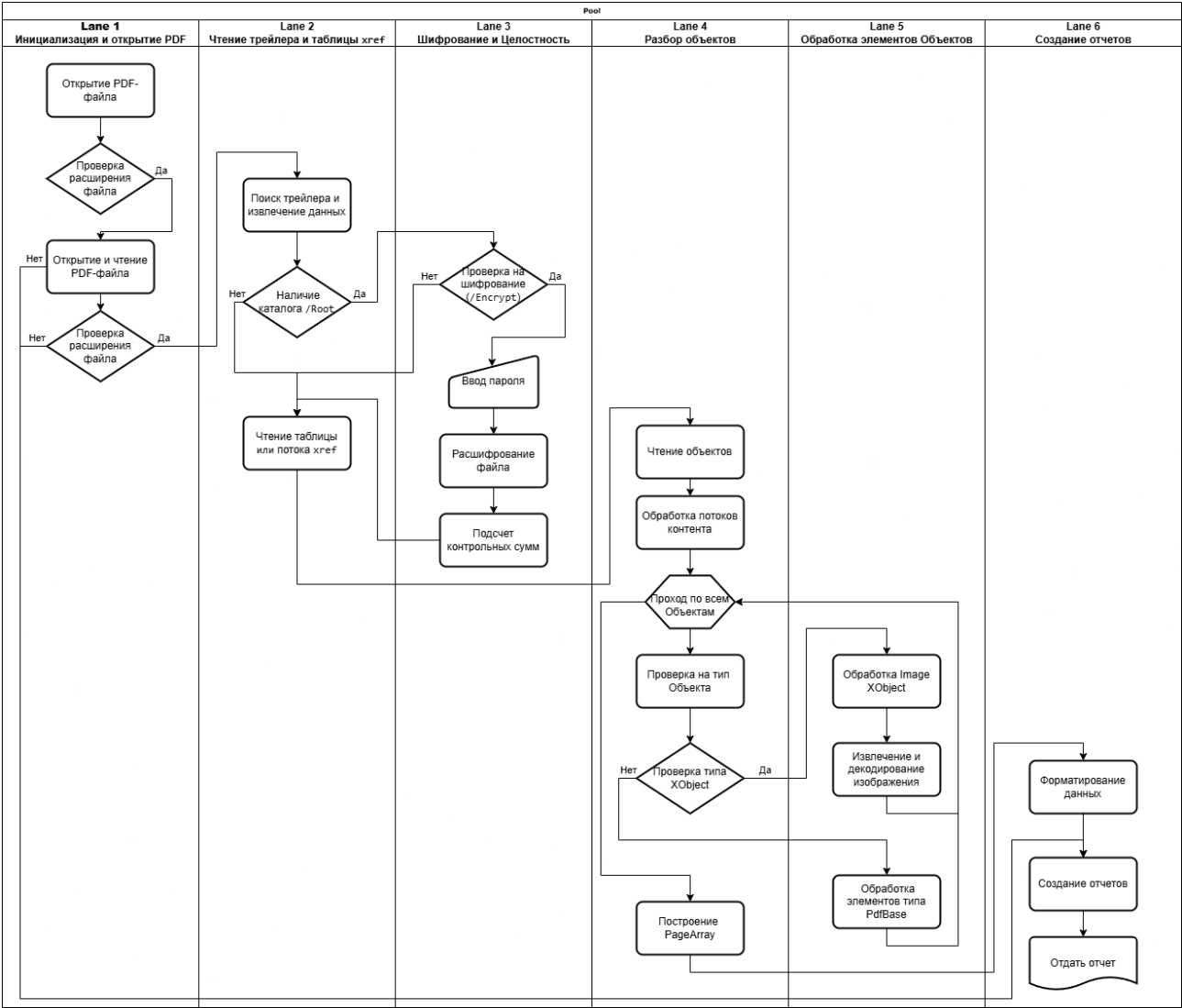


Рисунок 11 - Процесс извлечение и анализа PDF-документа

2.2.1 Разбор структуры PDF-документа

Для успешного анализа и обработки PDF-документа необходимо сначала правильно разобрать его структуру. Каждый из элементов может быть представлен в виде отдельных объектов в файле. Основной задачей на этом этапе является корректная обработка этих объектов и извлечение нужной информации.

2.2.1.1 Чтение PDF-файла и проверка заголовка

Первым создается объект PdfBinaryReader, который отвечает за чтение байтов из PDF-файла. Он использует FileStream для доступа к файлу и позволяет нам работать с его содержимым в виде байтов. Проверяется, что файл имеет расширение .PDF и что файл существует в файловой системе.

Далее проверяем заголовок зная, что каждый PDF-документ начинается с фиксированной сигнатуры %PDF-х.х, где х.х — это версия PDF. Задача приложения — удостовериться, что файл начинается с правильного заголовка, что позволяет подтвердить, что перед нами действительно PDF-документ. Ниже приведен пример программы, которая выполняет данную проверку:

```
if (Buffer[Ptr + 0] != '%' || Buffer[Ptr + 1] != 'P' ||  
Buffer[Ptr + 2] != 'D' || Buffer[Ptr + 3] != 'F')  
throw new PdfException("Не валидный PDF file (плохая сигнатура:  
не соответствует %PDF-1.х)");
```

После проверки заголовка начинается поиск ключевых структур в документе, включая трейлер, который указывает на корневой объект PDF, и таблицу xref, которая описывает расположение объектов.

2.2.1.2 Поиск трейлера и таблицы xref

Трейлер в PDF-файле — это специальная структура, которая обычно расположена в конце документа и содержит ссылки на критически важные элементы, например, особенно важный вам сначала ключ /Root, а также ссылку на таблицу xref или поток xref. Типичная структура трейлера в файле выглядит так:

```
trailer  
<<  
  /Size 25  
  /Root 4 0 R  
  /Encrypt 10 0 R  
  /Prev 123456  
>>
```

Трейлер находится в конце PDF-документа, и для его нахождения необходимо переместить указатель в конец файла. Там должен быть

специальный маркер startxref, который указывает на позицию начала таблицы xref или потока xref. Это является одной из особенностей формата PDF, как раз наличие косвенных объектов и таблицы перекрёстных ссылок, которая отображает номера объектов на смещения в файле. Для более старых версий PDF используется текстовая XRef-таблица и обычный трейлер, а начиная с версии PDF 1.5 применяются сжатые XRef-потоки (XRef-stream).

После нахождения позиции начала таблицы, начинается чтение данных трейлера. Процесс поиска трейлера выполняется с помощью метода TrailerDict.FindValue(), который ищет в нём ключи. Пример использования в проекте:

```
public PdfBase TrailerDict { get; private set; }

PdfBase Temp = TrailerDict.FindValue("/Root");
PdfBase TempEncryptionDict = TrailerDict.FindValue("/Encrypt");
```

Вторым по очереди важности является как раз поиск ключа /Encrypt и если в трейлере присутствует он, то это означает, что PDF зашифрован. В таком случае, необходимо извлечь ключи шифрования и проверить права доступа.

После нахождения трейлера, он проверяется на наличие ключей /XRef и /XRefStm. Если трейлер содержит ключ /XRef, то мы извлекаем позицию таблицы xref, которая указывает на место в файле, где начинается таблица объектов. После проверяется, если в нем есть ссылка на предыдущие таблицы xref (/Prev). Это важно для восстановления связности в случае больших документов, разбитых на несколько частей.

```
if (TrailerDict.FindValue("/Prev").GetInteger(out int prev))
    return prev; // Если есть предыдущая таблица xref,
возвращаем ее
```

Если же в трейлере указан /XRefStm а не /Prev, это означает, что используется поток для представления таблицы xref. В этом случае получается ссылка на этот поток, который будет распарсен как объект, содержащий информацию о размещении объектов. В итоге строится карта «object number → file offset». Конкретно в реализации приложения применяется отложенный

доступ (lazy-loading): объекты создаются и разбираются только при первом обращении, что позволяет экономить ресурсы и ускорять работу с большими PDF-документами.

2.2.1.3 Чтение таблицы xref или потока xref

После того как был найден трейлер, и мы извлекли информацию о таблице xref или потоке xref, следующим шагом является их чтение. В этом процессе важно учитывать, что PDF может использовать два типа таблиц: традиционную таблицу xref и потоковую таблицу. Xref содержит информацию о том, где в файле расположены объекты PDF, а также обеспечивают эффективный доступ к этим объектам.

Если в трейлере указан ключ /XRef, это означает, что используется традиционная таблица xref, в которой содержатся данные о положении каждого объекта в документе. В этом случае нужно переместить указатель в файл на позицию таблицы xref и начать разбор её содержимого.

Типичная структура таблицы xref:

```
xref
0 1
0000000000 65535 f
1 2
0000000010 00000 n
0000000064 00000 n
```

Здесь указано, что объект с номером 0 является удалённым (помечен как f), а объекты с номерами 1 и 2 активны и расположены на позициях 10 и 64 в файле.

Для извлечения данных из таблицы xref, приложение читает данные о каждой строке таблицы, восстанавливает объекты на основе их позиций и номеров поколений:

```
PdfBase xrefPos = TrailerDict.FindValue("/XRef");
if (!xrefPos.IsEmpty)
{
    long xrefTablePosition = xrefPos.ToInteger();
    reader.SetFilePosition(xrefTablePosition);
}
```

Если объект активен, то его данные извлекаются, если объект помечен как удаленный, то мы пропускаем его.

Если же в трейлере указан ключ /XRefStm, это означает, что используется поток xref, а значит данные о размещении объектов сжаты, и нам необходимо распаковать их перед обработкой. Типичная структура потока xref:

```
xref
stream
<stream data>
Endstream
```

Поток может также требовать расшифровки, если файл защищен, перед тем как данные можно будет прочитать. После извлечения данных из потока или таблицы, информация о каждом объекте включает его позицию, поколение и статус, что позволяет точно восстановить объекты в правильном порядке.

Пример обротки потока в приложении:

```
PdfBase xrefStreamPos = TrailerDict.FindValue("/XRefStm");
if (!xrefStreamPos.IsEmpty)
{
    long xrefStreamPosition = xrefStreamPos.ToInteger();
    reader.SetFilePosition(xrefStreamPosition); // Перемещение
в позицию потока xref
    // Декодирование и обработка потока xref
}
```

Перед чтением содержимого может понадобиться декомпрессия или дешифровка. После распаковки и расшифровки данных мы восстанавливаем объекты на основе информации, содержащейся в потоке. Важно помнить, что ссылки могут быть косвенными и указывать на другие объекты в документе. Эти ссылки разрешаются через механизм разрешения ссылок (resolve the reference), который находит нужный объект по его номеру и поколению. Если объект был изменен (например, обновлен или заменен), нужно выбрать актуальную версию объекта, а не устаревшую.

Кроме базового разбора и поэтапного прохода по потокам, в приложении используется простой трюк, чтобы всё шустро работало и не падало. Например,

если вдруг текстовая XRef-таблица битая или вовсе пропала, можно просто пройти по файлу и искать в нём строки формата «123 0 obj ... endobj». По найденным смещениям собираются ссылки на объекты и документ всё равно удаётся разобрать.

Также бывает так, что не нужно сразу лезть во весь документ, достаточно узнать только заголовок, список страниц или пару картинок. В таких случаях сначала читается только начало файла, трейлер и XRef-таблица, а потом, когда реально понадобится, добирается до нужных объектов.

Эти два дополнительных подхода повышают устойчивость и производительность анализа, особенно для больших и частично повреждённых PDF-файлов.

Далее, уже для разобранного xref, где уже не важно поток это или таблица, программа следует этим шагам при восстановлении объектов:

- для каждой ссылки на объект, извлеченной из таблицы xref или потока xref, определяется актуальная версия объекта, используя его номер и поколение;
- после того как актуальная версия объекта найдена, она восстанавливается для дальнейшей обработки;
- все объекты, которые были восстановлены, могут быть использованы для анализа или извлечения их данных (например, извлечение текста, изображений, метаданных).

2.2.2 Разбор объектов PDF и работа с ними

После того как мы разобрали структуру PDF-документа, включая его заголовок, трейлер и таблицу xref, следующим этапом является разбор объектов, из которых состоит PDF. Объекты в PDF-документе могут быть разных типов: строки, числа, словари, массивы, ссылки на другие объекты, а также потоки данных. Каждый из этих объектов имеет свои особенности и требует соответствующей обработки.

2.2.2.1 Извлечение и разбор объектов (PdfObject)

Объект, каждый из которых может быть, как косвенным, так и прямым. Косвенные объекты (indirect objects) могут ссылаться на другие объекты, в то время как прямые объекты находятся непосредственно в документе. Для корректного извлечения данных необходимо правильно разобрать каждый объект, определить его тип и извлечь соответствующие данные.

Перед непосредственным определением типа объекта происходит лексическое сканирование: поток байт посимвольно разбивается на токены — числа, имена (начинающиеся с «/»), строки в скобках и ключевые слова вроде obj/endobj или stream/endstream. Такой подход позволяет представить сырой поток байт в виде удобных для обработки лексем.

Далее выполняется синтаксический разбор — сопоставление полученных токенов грамматике PDF. Наиболее распространён метод «рекурсивного спуска», при котором грамматика разбивается на логические единицы. Для каждой единицы реализуется рекурсивная процедура, которая вызывается при встрече соответствующего токена. В результате по вложенным вызовам формируется иерархическая модель документа, отражающая структуру PDF.

После этого приложение использует методы, такие как ParseNextItem() или ParseObjectRefNo(), которые читают следующий элемент и определяют его тип, позволяя извлекать данные и корректно обрабатывать объекты

Основные типы объектов:

– словарь (Dictionary): структура данных, состоящая из пар "ключ-значение". Каждый словарь может содержать другие объекты, например, массивы, строки, числа и другие словари. Основное назначение словарей — хранить метаданные или параметры объектов. Для извлечения значения по ключу в словаре используется метод FindValue(), который ищет нужный ключ и возвращает соответствующее значение:

```
public PdfBase FindValue(string key) {  
    int index = KeyValueArray.BinarySearch(new  
    PdfKeyValue(key));
```

```

        return index < 0 ? PdfBase.Empty :
        KeyValueArray[index].Value;
    }

```

– массив (Array): представляет собой упорядоченную коллекцию объектов. В PDF массивы используются для хранения последовательностей объектов, например, объектов страницы или элементов на странице. В проекте обработка массива осуществляется с помощью доступа к его элементам через ToArrayItems, который преобразует данные в массив элементов для дальнейшей работы;

```

PdfBase[] ToArrayItems
{
    get => GetType() == typeof(PdfArray) ?
    ((PdfArray)this).Items.ToArray() : null;
}

```

– строка (String): могут быть представлены как текстовые данные или бинарные данные, закодированные в формате строк. Приложение извлекает строку и декодирует её, если это необходимо:

```

public string ToText()
{
    return this.ToString();
}

```

Также строки могут содержать специальные символы, такие как пробелы или символы новой строки, которые нужно учитывать при их обработке;

– число (Integer или Real): бывает целым или вещественным. Важно правильно извлекать их значения и использовать их в соответствующих контекстах, например, для представления координат или размеров;

```

public double ToNumber()
{
    if (GetType() == typeof(PdfInteger))
        return (double)((PdfInteger)this).IntValue;
    else if (GetType() == typeof(PdfReal))
        return ((PdfReal)this).RealValue;
    else
        return double.NaN;
}

```

– ссылка (Reference): косвенные объекты, которые могут указывать на другие объекты в документе. Чтобы обработать такие объекты, необходимо сначала разрешить ссылку, то есть найти объект по его номеру и поколению. В проекте это делается с помощью метода `ToPdfIndirectObject()`, который преобразует ссылку в объект;

```
public PdfIndirectObject ToPdfIndirectObject(PdfReference
reference)
{
    return ObjectTable[reference.ObjectNumber].FirstOrDefault(o
=> o.Generation == reference.Generation);
}
```

– поток (Stream): Это объект, который может содержать бинарные данные. Они могут быть как простыми, например, текст или изображения, так и более сложными, например, шрифты или ресурсы.

2.2.2.2 Обработка потока данных (Stream)

Потоки могут быть сжаты или зашифрованы, поэтому они требуют дополнительной обработки для извлечения и декодирования содержимого.

Для чтения потока используется метод `ReadStream()`, который возвращает данные потока в виде массива байтов;

```
public byte[] ReadStream()
{
    if(StreamLength == 0)
        return new byte[0];
    Reader.SetFilePosition(StreamFilePosition);
    byte[] ByteArray =
    Reader.PdfBinaryReader.ReadBytes(StreamLength);
    if(Reader.Encryption != null)
        ByteArray =
        Reader.Encryption.DecryptByteArray(ObjectNumber,
        ByteArray);
    return ByteArray;
}
```

После того как данные потока считаны, их нужно обработать. В PDF потоки часто используются для хранения сжатых или зашифрованных данных, поэтому извлеченные байты нужно будет декомпрессировать или расшифровать.

Для декомпрессии PDF использует различные фильтры для сжатия данных в потоках, чтобы уменьшить размер файла. Наиболее распространенные фильтры сжатия:

- FlateDecode – алгоритм использующий Deflate, который является стандартом для сжатия данных в PDF;

- LZWDecode – алгоритм сжатия, используемый в некоторых старых форматах, например, в GIF;

- ASCII85Decode – преобразует бинарные данные в текстовый формат ASCII85, что облегчает передачу и хранение данных в текстовых файлах. Этот фильтр часто используется для кодирования потоков перед сжатием или шифрованием;

- Predictor – улучшает эффективность сжатия, используя предсказание значений пикселей или данных на основе предыдущих значений. Обычно применяется совместно с фильтрами, такими как FlateDecode, для повышения степени сжатия;

- DCTDecode – реализует сжатие изображений с помощью алгоритма JPEG, основанного на дискретном косинусном преобразовании (DCT). Используется для эффективного хранения растровых изображений в PDF.

2.2.2.3 Расшифровка данных

Для корректной работы с сжатыми данными необходимо выполнить декомпрессию потока. Метод DecompressStream() используется для применения нужного фильтра и расшифровки данных:

```
public byte[] DecompressStream(byte[] byteArray) {  
    string[] filterNames = GetFilterNameArray();  
    if (filterNames.Contains("/FlateDecode"))  
        return FlateDecode(byteArray);  
    else if (filterNames.Contains("/LZWDecode"))  
        return LZWDecode(byteArray);  
    return byteArray;  
}
```

В случае с защищенными PDF-документами, потоки могут быть зашифрованы, и перед их использованием необходимо их расшифровать. PDF

использует симметричные алгоритмы шифрования, такие как AES или RC4 для защиты данных. Приложение в свою очередь из-за ограничений программной системы net framework реализует AES128 и Standard128(RC4).

Шифрование и дешифрование данных в PDF строится на базе ключей, которые генерируются с использованием алгоритма MD5. Для повышения безопасности ключ проходит 50 итераций MD5-хэширования – этот процесс значительно усложняет подбор ключа методом перебора и обеспечивает надежную защиту.

Основные этапы генерации ключа и расшифровки:

- получение пароля (пользовательского или владельца документа);
- обработка пароля: пароль преобразуется в бинарный формат и дополняется до нужной длины;
- генерация мастер-ключа: на основе пароля, метаданных документа и дополнительных параметров создаётся мастер-ключ (master key) с помощью многократного применения MD5-хэширования (50 раз подряд);

```
internal byte[] CreateEncryptionKey(int ObjectNumber)
{
    byte[] HashInput = new byte[MasterKey.Length + 5 +
    (EncryptionType == EncryptionType.Aes128 ? Salt.Length : 0)];
    int Ptr = 0;
    Array.Copy(MasterKey, 0, HashInput, Ptr, MasterKey.Length);
    Ptr += MasterKey.Length;
    // Добавляем номер объекта (3 байта) и поколение (2 байта, в
данном случае 0)
    HashInput[Ptr++] = (byte)ObjectNumber;
    HashInput[Ptr++] = (byte)(ObjectNumber >> 8);
    HashInput[Ptr++] = (byte)(ObjectNumber >> 16);
    HashInput[Ptr++] = 0;
    HashInput[Ptr++] = 0;
    // Если используется AES128, добавляем соль
    if (EncryptionType == EncryptionType.Aes128)
        Array.Copy(Salt, 0, HashInput, Ptr, Salt.Length);
    // Многократное хэширование MD5)
    byte[] EncryptionKey = MD5.ComputeHash(HashInput);
    if (EncryptionKey.Length > 16) Array.Resize(ref
EncryptionKey, 16);
    return EncryptionKey;
}
```

– создание ключа объекта: для каждого объекта документа генерируется уникальный ключ шифрования с учётом номера объекта и поколения;

– дешифровка данных: с использованием сгенерированного ключа и выбранного алгоритма шифрования происходит расшифровка потока;

Только после выполнения этих пунктов можно приступить к чтению всех последующих объектов. При этом, приложение хранит в памяти уже расшифрованный поток или строку, чтобы не выполнять повторно дорогостоящие криптооперации.

Для декодирования или расшифровки потока используется метод DecryptByteArray(), который применяет соответствующий алгоритм шифрования, используя ключи, извлеченные из PDF-документа:

```
internal byte[] DecryptByteArray(int objectNumber,
byte[] cipherText){
    byte[] encryptionKey =
CreateEncryptionKey(objectNumber);
    if (EncryptionType == EncryptionType.Aes128)
    {
        AES.Key = encryptionKey;
        byte[] iv = new byte[16];
        Array.Copy(cipherText, 0, iv, 0, 16);
        AES.IV = iv;
        byte[] encryptedData = new
byte[cipherText.Length - 16];
        Array.Copy(cipherText, 16, encryptedData, 0,
encryptedData.Length);
        return ProcessCrypto(encryptedData, () =>
AES.CreateDecryptor());
    }
    else
    {
        byte[] decryptedData =
(byte[])cipherText.Clone();
        EncryptRC4(encryptionKey, decryptedData);
        return decryptedData;
    }
}
```

В целом работа шифрования и декомпрессии в цепочке фильтров использует подход в приложении соответствует паттерну «цепочка обязанностей»: фильтры выстраиваются в том порядке, в каком они указаны в

/Filter, и каждый по очереди берёт на вход байты от предыдущего, выдавая результат следующему. При этом на каждом шаге обязательно выполняется проверка целостности, например, контрольная сумма Adler-32, обрабатываются возможные ошибки формата, чтобы единичная проблема в одном фильтре не уничтожила весь разбор.

2.2.3 Обработка объектов XObject

В структуре PDF-документа объекты типа XObject нужны для представления графических ресурсов, которые могут использоваться многократно. К XObject относятся в первую очередь изображения (ImageXObject) и формы (FormXObject) – группы операторов, описывающих графику, текст и прочие элементы. Обработка этих объектов требует специализированного подхода для извлечения содержимого и правильного представления данных.

При обнаружении объекта с типом /XObject приложение вызывает статический фабричный метод PdfXObject.Create(). Этот метод считывает ключ /Subtype из словаря объекта и, в зависимости от значения, создает конкретный класс для работы с этим XObject:

- если /Subtype равен /Form, создается объект PdfFormXObject;
- если /Subtype равен /Image, создается объект PdfImageXObject;
- если ничего из этого, то для остальных типов создается универсальный класс GenericPdfXObject, который выводит базовую информацию.

Такой подход позволяет централизованно управлять созданием обработчиков и легко расширять поддержку новых типов XObject.

2.2.3.1 Работа с классом XObjectForm

Объекты типа Form в PDF представляют собой формы специальные потоки операторов, описывающих графические элементы, текст и трансформации. В рамках приложения реализация обработки Form XObject выполнена достаточно поверхностно. Она ограничивается:

- извлечением и декомпрессией потока содержимого формы;
- парсингом потока в последовательность операторов PDF;
- предоставлением этой последовательности для дальнейшего анализа или отображения.

Однако функционал работы с формами здесь не полный и не включает поддержку всех аспектов, таких как управление ресурсами, вложенными формами, сложные матричные трансформации или повторное использование форм в разных контекстах. Это сознательное ограничение, обусловленное целью проекта, первичным разбором и анализом структуры PDF, а не полноценным рендерингом.

2.2.3.2 Работа с классом XObjectImage

В отличие от форм, обработка Image реализована гораздо глубже и позволяет полноценно работать с графическими данными. Алгоритм обработки изображений в приложении включает следующие ключевые этапы:

чтение и декомпрессия потока изображения;

После чтения и декомпрессии потока изображения, которые учитывают наличие и порядок фильтров сжатия, например, FlateDecode. Далее приложение анализирует словарь объекта, чтобы определить применённые фильтры. Если в фильтрах содержится /DCTDecode, это означает, что данные закодированы в формате JPEG – наиболее распространённом формате сжатия изображений.

```
var filterVal = dict.FindValue("/Filter");
bool isJpeg = false;
if (filterVal.IsName && filterVal.ToName == "/DCTDecode")
    isJpeg = true;
else if (filterVal.IsArray)
{
```

```

var arr = filterVal.ToArrayItems;
for (int i = 0; i < arr.Length; i++)
{
    if (arr[i].IsName && arr[i].ToName == "/DCTDecode")
    {
        isJpeg = true;
        break;
    }
}
}

```

Если изображение JPEG то приложение создает поток памяти на основе массива байтов и создает объект Bitmap напрямую, что обеспечивает быструю и надежную загрузку изображения без необходимости собственного разбора формата JPEG. Следом изображение рассматривается как «сырые» пиксели, требующие детального разбора. В этом случае из словаря извлекаются ключевые параметры:

- ширина и высота изображения (/Width, /Height);
- биты на компонент BitsPerComponent глубина цвета каждого канала;
- цветовое пространство ColorSpace определяет, сколько компонентов содержит пиксель и как их интерпретировать, например, DeviceGray – 1 компонент, DeviceRGB – 3, DeviceCMYK – 4).

```

string cs = dict.FindValue("/ColorSpace").ToName ??
"/DeviceGray";
int comps = cs switch
{
    "/DeviceGray" => 1,
    "/DeviceRGB" => 3,
    "/DeviceCMYK" => 4,
};

```

В зависимости от цветового пространства выбирается подходящий формат пикселей для объекта Bitmap. Приложение для градаций серого (DeviceGray) создается 8-битный индексированный Bitmap с палитрой оттенков серого, для RGB - 24-битный формат с тремя каналами, а для CMYK - 32-битный формат.

Далее приложение копирует байты пиксельных данных из потока в память Bitmap, учитывая ширину, высоту и количество цветовых компонентов. Для этого используется блокировка данных Bitmap и прямое копирование байтов.

```
var rect = new Rectangle(0, 0, width, height);
BitmapData bd = bmp.LockBits(rect, ImageLockMode.WriteOnly, pf);
int stride = bd.Stride;
IntPtr dst = bd.Scan0;
for (int y = 0; y < height; y++){
    int srcOff = y * width * comps;
    IntPtr rowPtr = IntPtr.Add(dst, y * stride);
    System.Runtime.InteropServices.Marshal.Copy(data,
srcOff, rowPtr, width * comps);
}
```

Этот этап завершает формирование полноценного изображения из потоковых данных PDF.

2.2.4 Форматирование данных и вывод

После того, как данные PDF-документа извлечены и проанализированы, следующей важной задачей является их форматирование и представление в удобной и понятной форме для пользователя. Приложение реализует комплексный механизм, который преобразует внутренние бинарные и структурированные данные в читаемые текстовые или визуальные форматы и передает их клиентскому интерфейсу для отображения.

Многие данные в PDF представлены в виде байтовых потоков – это может быть содержимое текстовых объектов, потоков с инструкциями, шрифтов, или даже сжатые и зашифрованные сегменты. Для визуализации и анализа эти данные необходимо преобразовать в текстовый формат, удобный для восприятия человеком.

Приложение использует методы, которые поэлементно преобразуют байты в символы ASCII, при этом непечатаемые или неразборчивые байты заменяются на точку для сохранения читаемости.

```
public static string ByteArrayToString(byte[] ByteArray)
{
    StringBuilder Str = new StringBuilder();
    byte LastByte = 0;
```

```

foreach (byte OneByte in ByteArray)
{
    if (OneByte == (byte)'\n'){
        if (LastByte != (byte)'\r') Str.Append("\r\n");
    }
    else if (OneByte == (byte)'\r')
        Str.Append("\r\n");
    else if (OneByte < 32 || (OneByte > 126 && OneByte < 160))
        Str.Append('.');
    else
        Str.Append((char)OneByte);
    LastByte = OneByte;
}
if (Str.Length != 0 && Str[Str.Length - 1] != '\n')
    Str.Append("\r\n");
return Str.ToString();
}

```

Для низкоуровневого анализа или отладки приложение предлагает вывод данных в шестнадцатеричном формате. Это особенно важно, когда нужно увидеть точный байтовый состав данных, например, для проверки корректности декомпрессии или дешифровки. При этом данные разбиваются на строки по 16 байт (адрес начала блока данных, шестнадцатеричные значения каждого байта, Отображение этих байтов в ASCII).

```

public static string ByteArrayToHex(byte[] ByteArray){
    byte[] HexLine = new byte[16];
    StringBuilder Str = new StringBuilder();
    int Length = ByteArray.Length & ~15; // длина, кратная 16
    for (int Pos = 0; Pos < Length; Pos += 16)
    {
        Array.Copy(ByteArray, Pos, HexLine, 0, 16);
        FormatHexLine(Pos, HexLine, Str);
    }
    int Extra = ByteArray.Length - Length;
    if (Extra > 0)
    {
        Array.Copy(ByteArray, Length, HexLine, 0, Extra);
        FormatHexLine(Length, HexLine, Str);
    }
    return Str.ToString();
}

```

В конце приложение предоставляет возможность формировать отчёты, отражающие содержимое PDF-документа, его структуру и отдельные объекты. Отчёты включают:

- информацию о трейлере и основных словарях;
- сводки по каждому косвенному объекту, включая тип, положение и содержимое;
- анализ потоков и содержимого страниц;
- метаданные документа;

Для этого реализован вспомогательный класс `OutputCtrl` для управления выводом, который аккумулирует текст, добавляет переносы строк и форматирует информацию для удобного восприятия.

```
OutputCtrl Ctrl = new OutputCtrl();
Ctrl.AppendMessage("PDF file name: example.pdf");
Ctrl.AddEol();
Ctrl.AppendMessage("Trailer Dictionary");
Ctrl.AppendMessage("-----");
Reader.TrailerDict.ToByteArray(Ctrl);
Ctrl.AddEol();
```

2.3 Разработка пользовательского интерфейса

Для реализации графического интерфейса приложения использовалась технология `Windows Forms`, обеспечивающая стабильную и проверенную платформу для создания настольных приложений на платформе `.NET`. В ходе разработки было спроектировано и реализовано несколько ключевых окон, которые обеспечивают удобное взаимодействие пользователя с функционалом анализа PDF-документов: `TestPDFence`, `AnalysisScreen`, `DisplayText`, `Password` и `RecentFilesSelector`.

При проектировании интерфейса особое внимание уделялось созданию интуитивно понятной и доступной среды для пользователей. Были реализованы два основных интерфейсных окна, обеспечивающих удобное взаимодействие пользователя с функционалом анализа PDF-документов, это главное окно

приложения, изображенное на рисунке 12, для открытия PDF-файлов и запуска процесса их анализа и окно истории недавно открытых файлов, представленного на рисунке 13, позволяющее быстро выбирать из списка ранее обработанных документов.

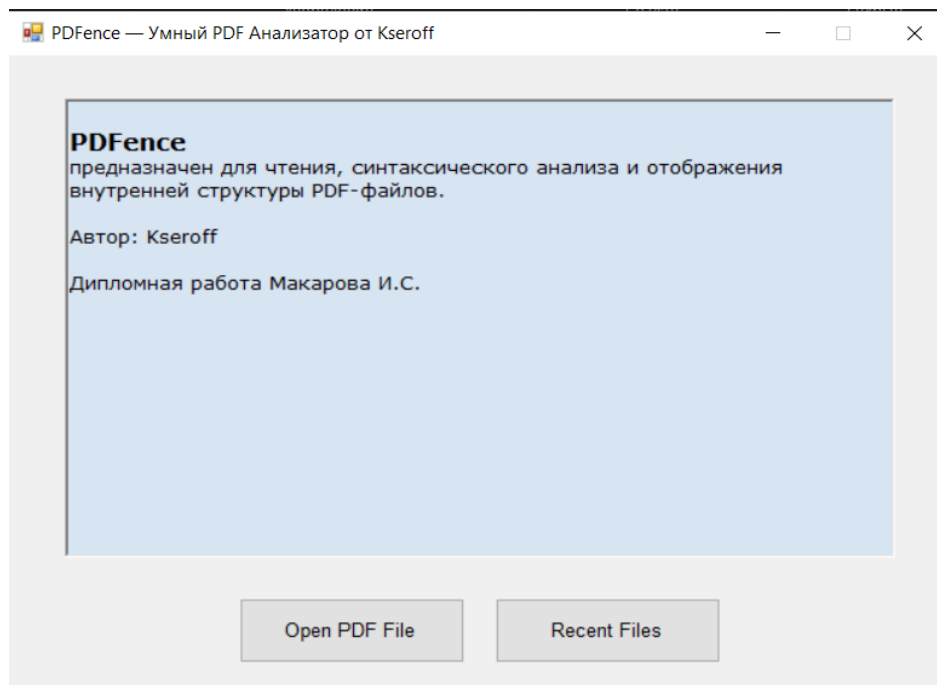


Рисунок 12 – Главное окно программы

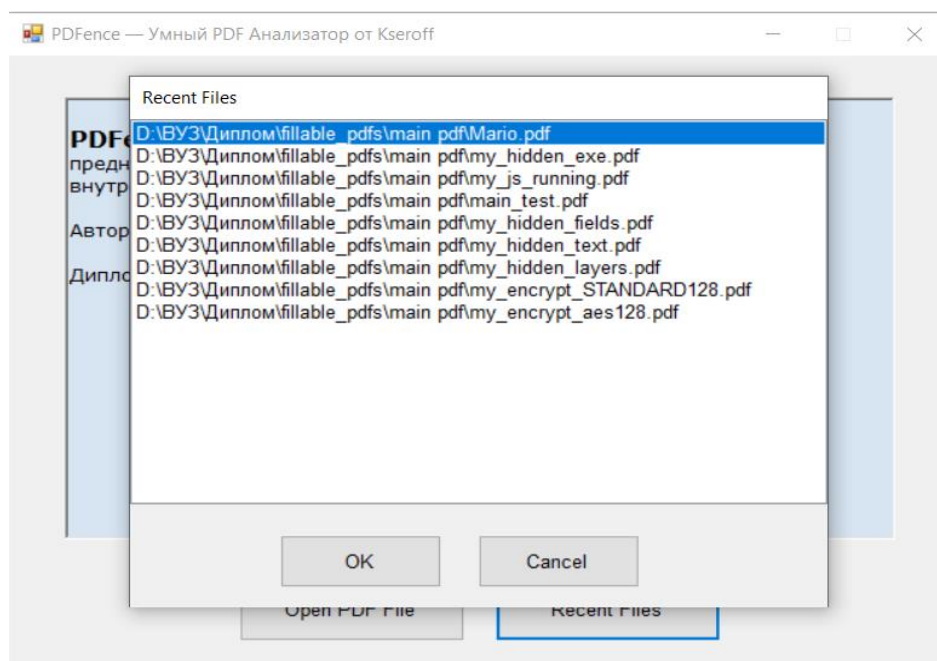


Рисунок 13 – История ранее открытых PDF

После выбора PDF-файла и его успешной загрузки приложение отображает информационное сообщение, содержащее основные характеристики документа: версию PDF, количество страниц и общее число объектов. Это помогает пользователю быстро получить обзор загруженного файла и убедиться в корректности его открытия.

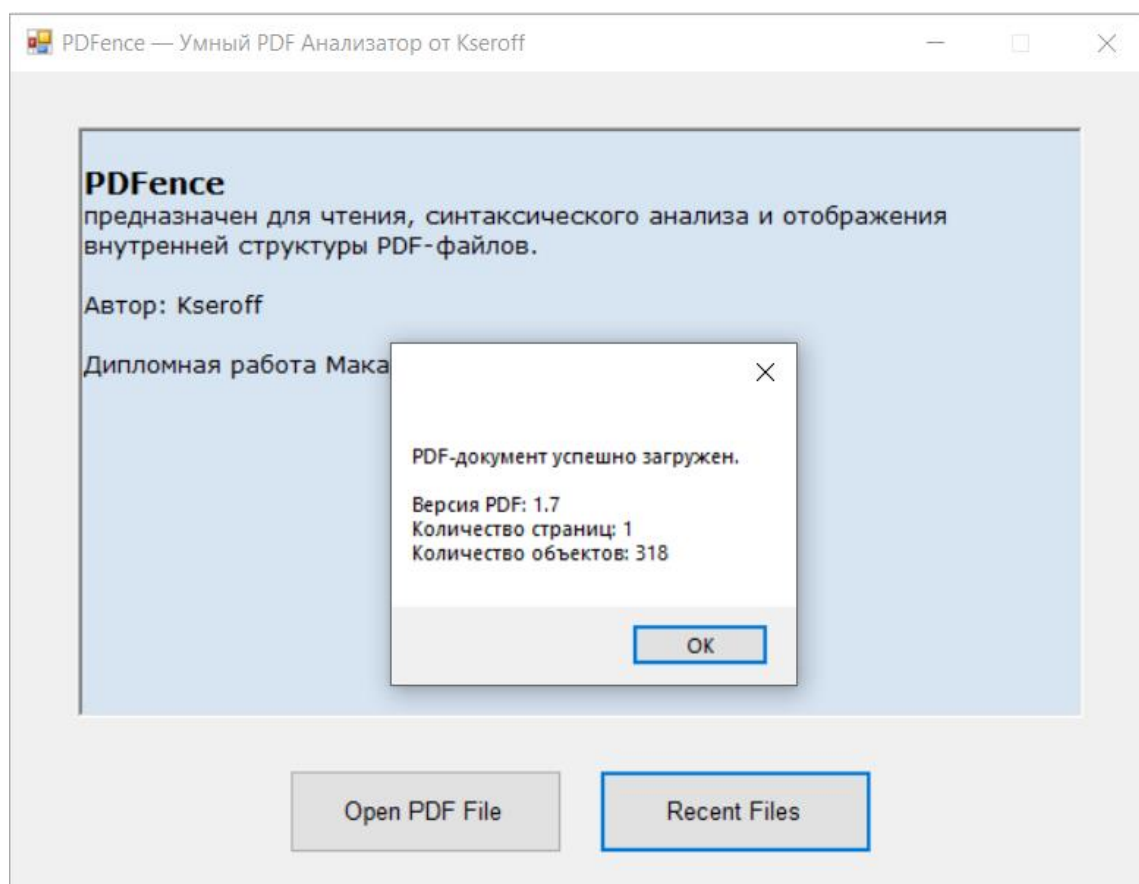


Рисунок 14 – Информационное сообщение с параметрами загруженного PDF-документа

Организация пространства в основных окнах строилась с помощью контейнеров и свойств позиционирования элементов, что позволяло обеспечить адаптивность интерфейса при изменении размеров окна. В частности для окна, изображенного на рисунке 15, использовался компонент DataGridView для отображения структуры документа с динамической подстройкой ширины колонок под содержимое, а строка поиска над таблицей позволяла эффективно

фильтровать объекты по номеру, типу и подтипу, что значительно ускоряет работу с большими файлами.

D:\ВУЗ\Диплом\fillable_pdfs\main pdf\Mario.pdf

Object No.	Object	Type	Subtype	ObjStm ObjNo	ObjStm Index	Object Pos	Stream Pos	Stream Len
245	Dictionary	/Info		373	33	2 016 (0x7E0)		
1	Stream					15 (0xF)	65 (0x41)	336 (0x150)
7	Stream					419 (0x1A3)	525 (0x20D)	312 (0x138)
8	Stream	/Metadata	/XML			855 (0x357)	913 (0x391)	1 080 (0x438)
12	Array			372	0	1 846 (0x736)		
13	Dictionary			372	1	1 878 (0x756)		
14	Dictionary	/Font	/Type1	372	2	1 978 (0x7BA)		
15	Dictionary	/Font	/Type1	372	3	2 062 (0x80E)		
16	Dictionary	/Encoding		372	4	2 132 (0x854)		
17	Dictionary	/StructTreeRoot		372	5	3 442 (0xD72)		
18	Dictionary			372	6	3 563 (0xDEB)		
19	Dictionary			372	7	4 088 (0xFF8)		
20	Dictionary			372	8	5 828 (0x16C4)		
21	Dictionary			372	9	5 995 (0x176B)		
22	Dictionary			372	10	6 067 (0x17B3)		
23	Dictionary			372	11	6 139 (0x17FB)		
24	Dictionary			372	12	6 211 (0x1843)		
25	Dictionary			372	13	6 283 (0x188B)		
26	Dictionary			372	14	6 355 (0x18D3)		
27	Dictionary			372	15	6 427 (0x191B)		
28	Dictionary			372	16	6 499 (0x1963)		
29	Dictionary			372	17	6 571 (0x19AB)		
30	Dictionary			372	18	6 643 (0x19F3)		
31	Dictionary			372	19	6 715 (0x1A3B)		
32	Dictionary			372	20	6 787 (0x1A83)		
33	Dictionary			372	21	6 859 (0x1ACB)		
34	Dictionary	/StructElem		372	22	7 180 (0x1C0C)		
35	Dictionary	/StructElem		372	23	7 322 (0x1C9A)		
36	Dictionary			372	24	7 485 (0x1D3D)		

View Summary Close

Рисунок 15 – Экран анализа структуры PDF с таблицей объектов

Одной из важных особенностей интерфейса стало поддержание нескольких режимов просмотра содержимого объекта, приведенных на рисунках 16–20. Использовались стандартные элементы TextBox и RichTextBox для вывода текстовых данных и отчетов. В зависимости от типа выбранного объекта пользователь может переключаться между текстовым и шестнадцатеричным представлением потоковых данных, а также просматривать встроенные

изображения с помощью элемента PictureBox с возможностью масштабирования и поворота. Для визуализации изображений реализован специальный компонент, использующий класс Bitmap из стандартной библиотеки .NET, что позволяет корректно отображать графику, включая JPEG.

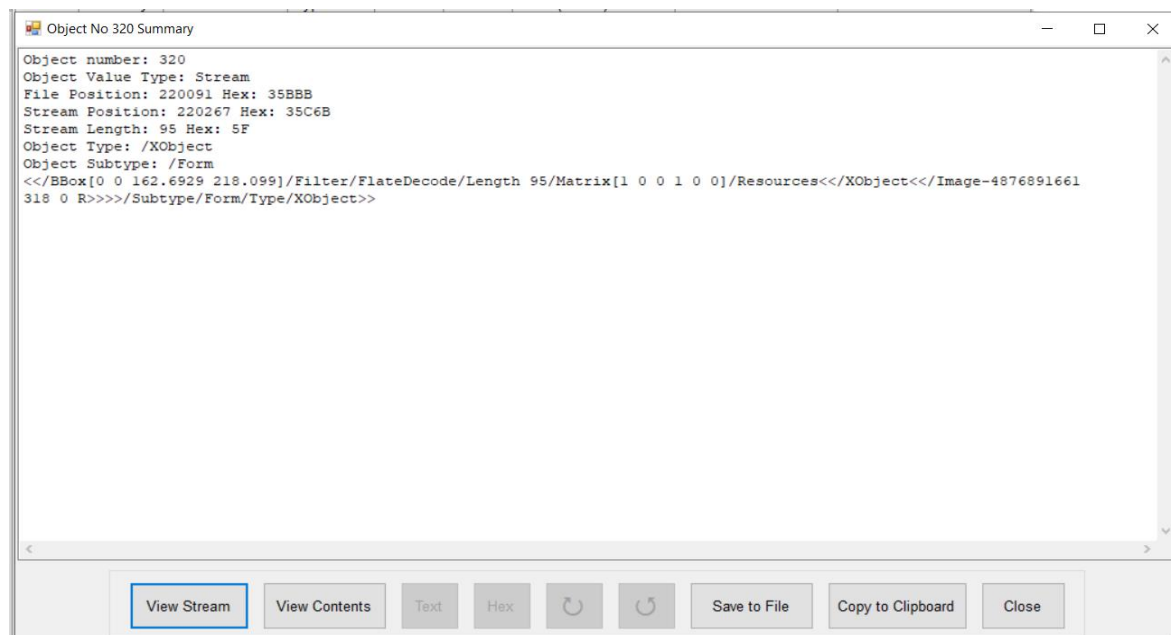


Рисунок 16 – Просмотр содержимого выбранного объекта

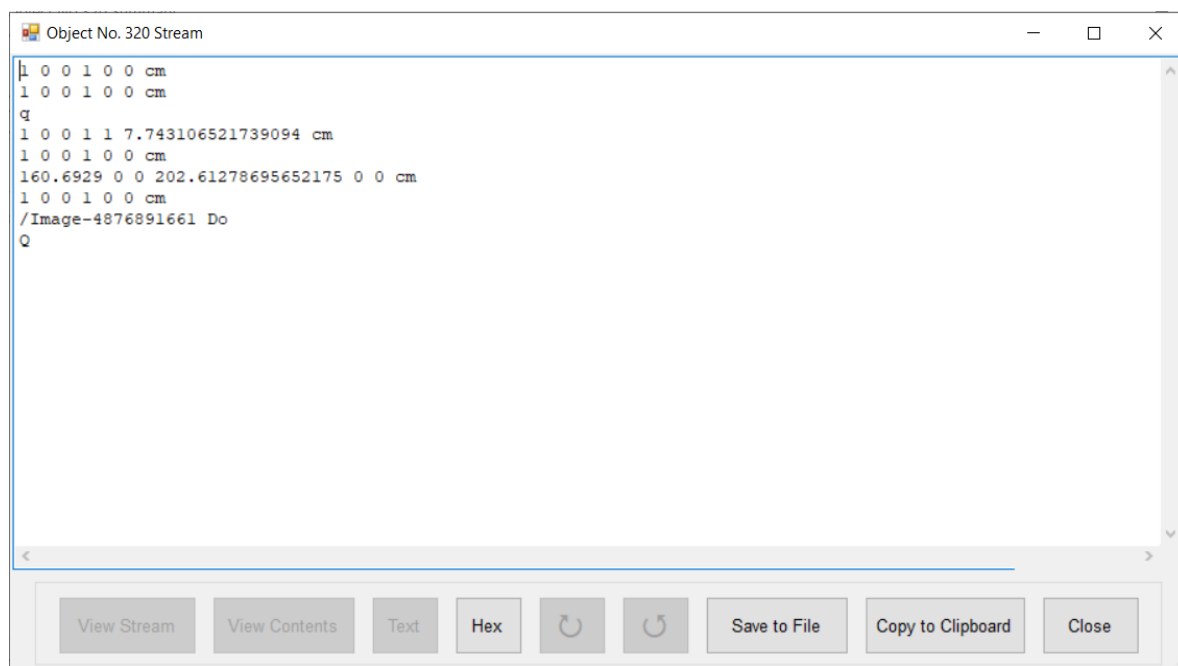


Рисунок 17 – Просмотр потока содержимого объекта

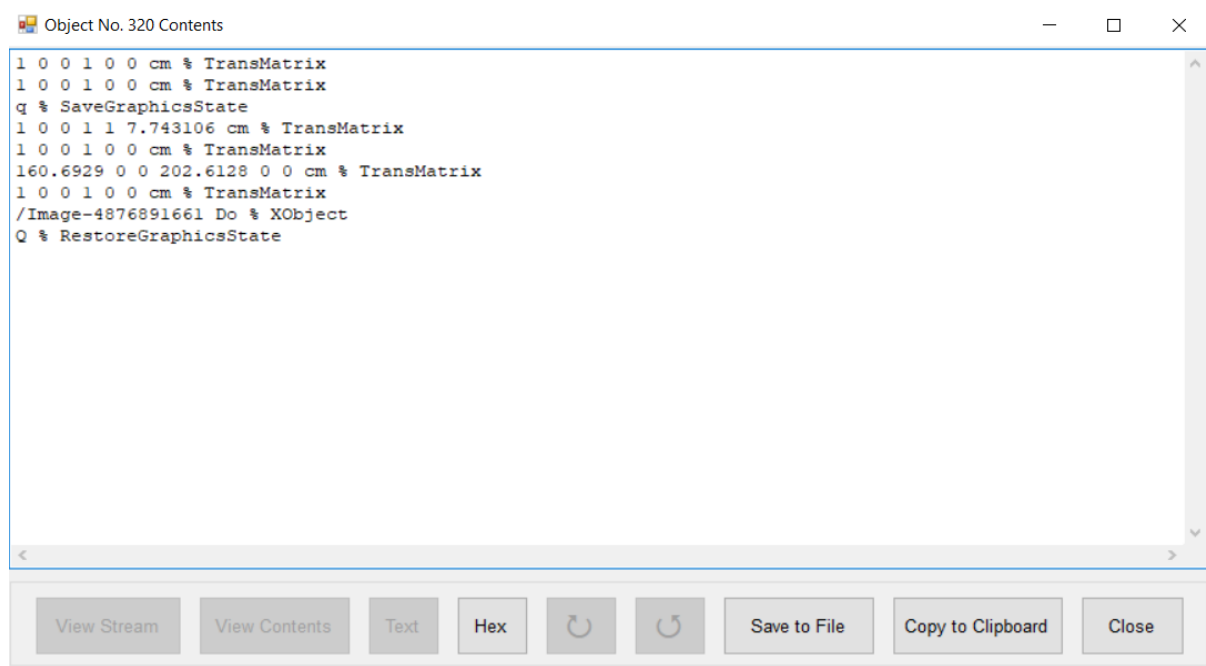


Рисунок 18 – Просмотр контекста объекта в текстовом режиме

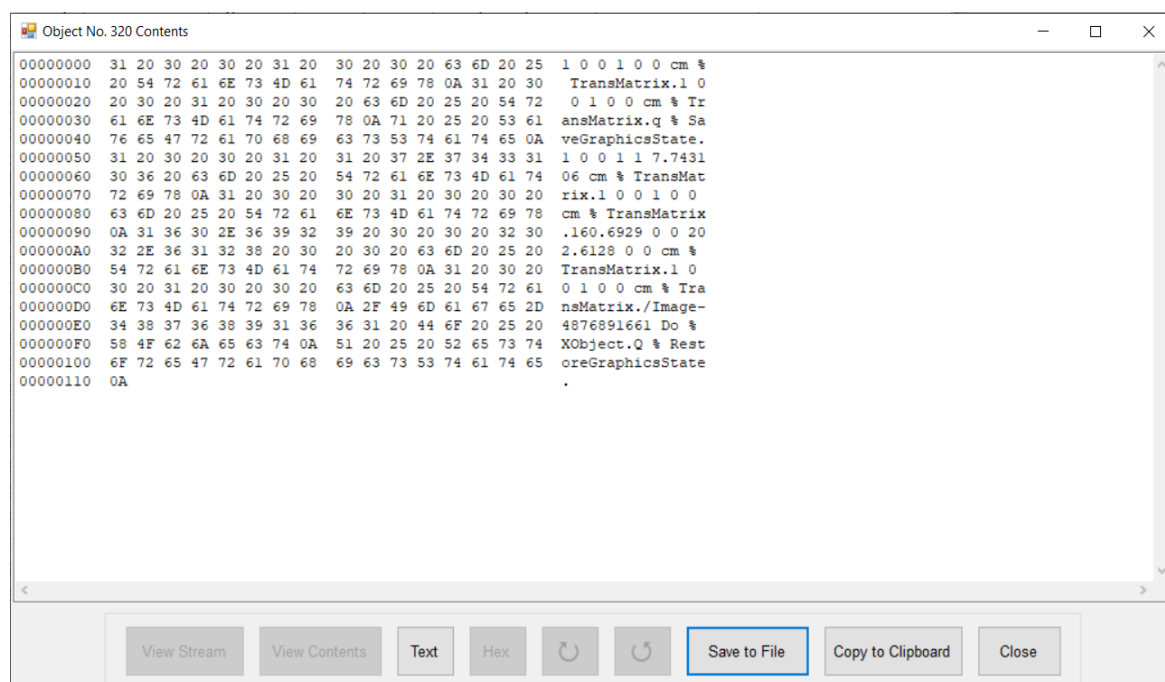


Рисунок 19 – Просмотр контекста объекта в шестнадцатеричном режиме (HEX)

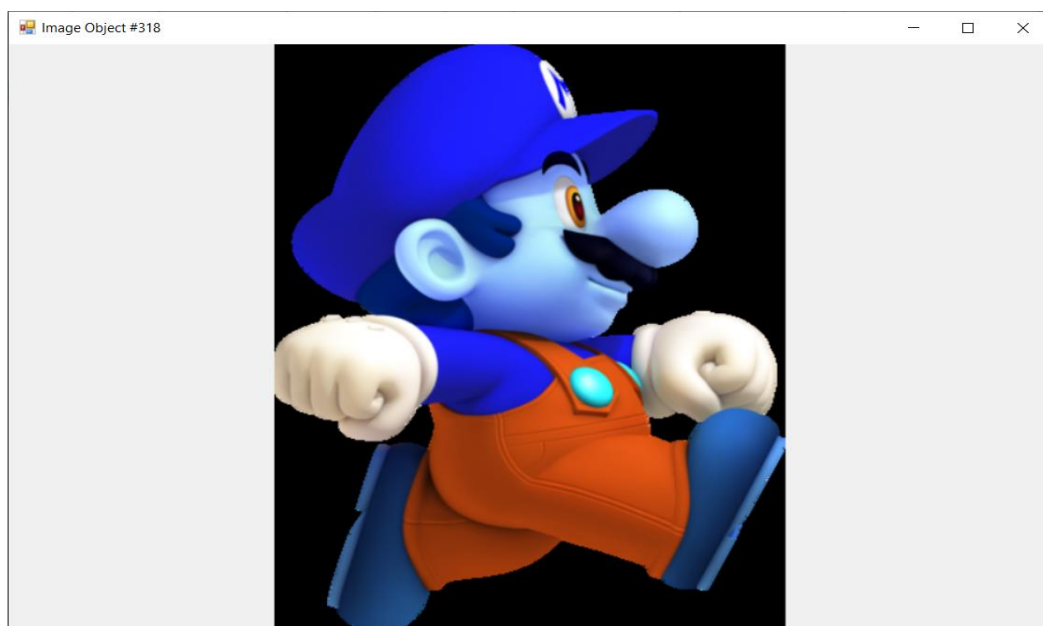


Рисунок 20 – Отображение встроенного изображения из PDF

Работа с зашифрованными PDF-документами организована через отдельный диалог, представленный на рисунке 21, ввода пароля, который появляется автоматически при необходимости. Это обеспечивает безопасное и удобное взаимодействие с защищёнными файлами, позволяя пользователю быстро вводить необходимые данные для расшифровки и продолжать работу.

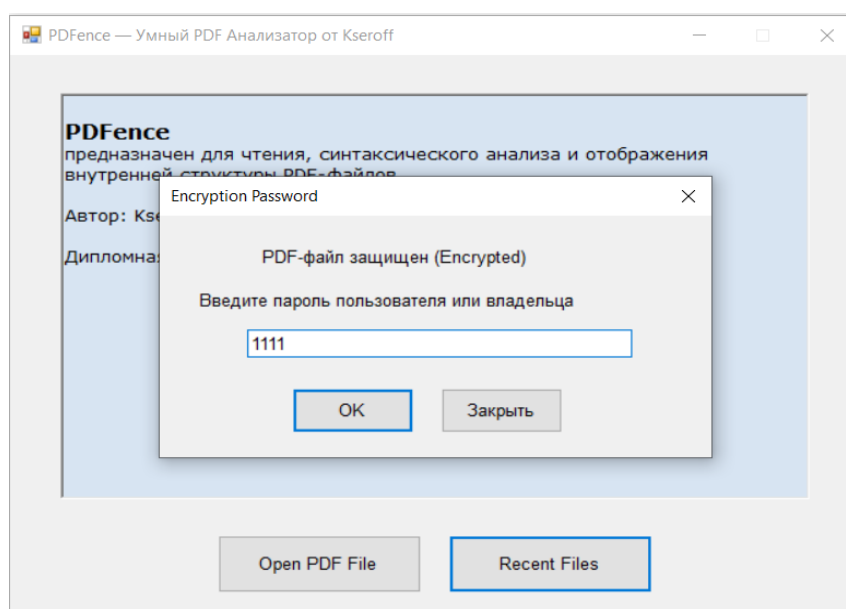


Рисунок 21 – Диалог ввода пароля

Кроме того, предусмотрена функциональность экспорта данных: пользователь может сохранять отчеты, содержимое потоков и изображения в отдельные файлы, а также копировать данные в буфер обмена. Это расширяет возможности приложения и позволяет легко интегрировать результаты анализа в другие системы или процессы.

Визуальное оформление интерфейса выдержано в едином стиле с использованием стандартных шрифтов и цветов, что обеспечивает комфортное восприятие информации и минимизирует усталость пользователя при длительной работе с приложением.

Таким образом, разработанный пользовательский интерфейс обеспечивает удобство, гибкость и полноту функционала, позволяя пользователю эффективно анализировать внутреннюю структуру PDF-документов и работать с их содержимым на низком уровне, не прибегая к сторонним инструментам.

3 Результаты работы приложения

Тестирование работы приложения начинается с проверки корректности и стабильности функционирования на обычных PDF-документах. Это позволило убедиться в правильности разбора структуры файлов и корректном отображении содержимого. Далее тестирование приложения проводилось с учётом реальных угроз, присущих стандарту PDF. В результате было определено, с какими угрозами приложение уже справляется, а с какими оно не может эффективно работать на данном этапе. Важным аспектом является не только выявление враждебных элементов, но и защита от различных атак, включая проверку целостности и манипуляции с цифровыми подписями. Этот анализ позволяет подвести итог текущей разработке и выявить направления для её дальнейшего совершенствования, чтобы учесть все особенности и риски, связанные с безопасностью PDF-документов.

3.1 Тестирование корректности работы программы

Для подтверждения правильности работы разработанного приложения и достоверности анализа PDF-документов был проведён ряд сравнительных тестов с использованием популярного инструмента для разбора PDF – iText RUPS. В качестве основного тестового примера был выбран файл "eXtensible Host Controller Interface for Universal Serial Bus (xHCI)", опубликованный компанией Intel. Это полноценный технический документ в формате PDF, доступный в открытом доступе. Документ содержит десятки страниц, текст, таблицы, закладки и стандартные графические элементы. При этом он создан в версии PDF 1.6, что удовлетворяет техническим требованиям программы (поддерживаются версии PDF ниже 2.0).

После загрузки документа приложение корректно отобразило полную структуру файла, на рисунке 22 представив список всех объектов с указанием их номера, типа, подтипа и позиции в файле. Для подтверждения правильности

разбора был проведён параллельный анализ с помощью инструмента iText RUPS, рисунок 23. Результаты показали полное совпадение номеров и типов объектов в обоих приложениях, что служит надёжным доказательством корректности разбора структуры PDF.

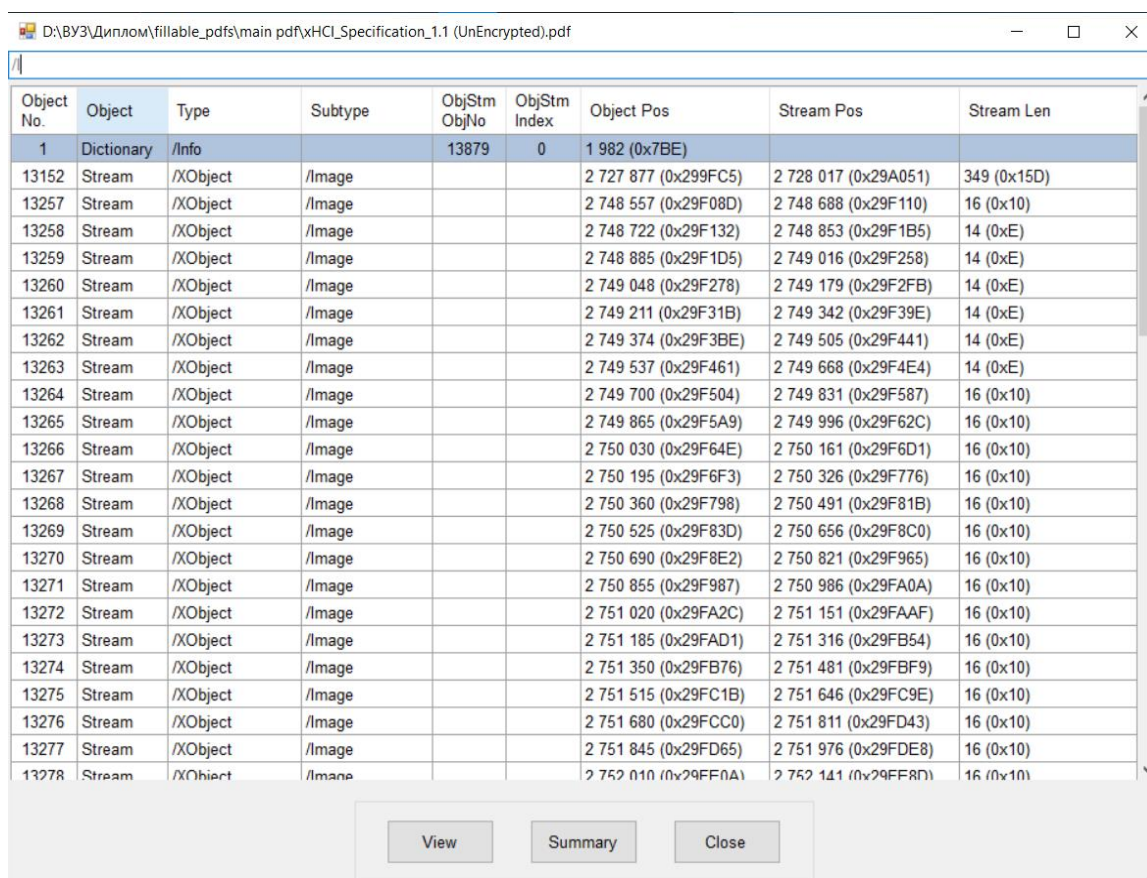
Object No.	Object	Type	Subtype	ObjStm ObjNo	ObjStm Index	Object Pos	Stream Pos	Stream Len
1	Dictionary	/Info				16 (0x10)		
2	Dictionary	/Catalog				390 (0x186)		
3	Stream	/Metadata	/XML			612 (0x264)	669 (0x29D)	4 368 (0x1110)
4	Dictionary					5 056 (0x13C0)		
5	Dictionary	/Page				5 088 (0x13E0)		
6	Dictionary					5 226 (0x146A)		
7	Dictionary					5 279 (0x149F)		
8	Dictionary	/Pages				5 313 (0x14C1)		
9	Dictionary	/StructTreeRoot				5 394 (0x1512)		
10	Array					8 647 (0x21C7)		
11	Dictionary					8 671 (0x21DF)		
12	Dictionary					9 145 (0x23B9)		
13	Stream	/Contents				9 217 (0x2401)	9 267 (0x2433)	477 (0x1DD)
14	Dictionary	/Pages				9 763 (0x2623)		
15	Dictionary					9 894 (0x26A6)		
16	Dictionary					10 069 (0x2755)		
17	Dictionary					10 175 (0x27BF)		
18	Dictionary					10 344 (0x2868)		
19	Dictionary	/Pages				10 368 (0x2880)		
20	Dictionary	/Pages				10 507 (0x290B)		
21	Dictionary	/Pages				10 647 (0x2997)		
22	Dictionary	/Pages				10 787 (0x2A23)		
23	Dictionary	/Pages				10 927 (0x2AAF)		
24	Dictionary					11 067 (0x2B3B)		
25	Dictionary					11 128 (0x2B78)		
26	Dictionary					11 270 (0x2C06)		
27	Dictionary					11 317 (0x2C35)		
28	Dictionary					11 879 (0x2E67)		
29	Dictionary					11 952 (0x2F00)		

Рисунок 22 – Список объектов PDF в приложении

Pages	Outlines	Structure	Form	XFA	XREF	Plain Text
Number	Object					
1	Dictionary					
2	Dictionary of type: /Catalog					
3	Stream of type: /Metadata					
4	Dictionary					
5	Dictionary of type: /Page					
6	Dictionary					
7	Dictionary					
8	Dictionary of type: /Pages					
9	Dictionary of type: /StructTreeRoot					
10	Array					
11	Indirect object					
12	Indirect object					
13	Indirect object					
14	Dictionary of type: /Pages					
15	Indirect object					
16	Dictionary					
17	Dictionary					
18	Indirect object					
19	Dictionary of type: /Pages					
20	Dictionary of type: /Pages					
21	Dictionary of type: /Pages					
22	Dictionary of type: /Pages					
23	Dictionary of type: /Pages					
24	Dictionary					
25	Dictionary					
26	Indirect object					
27	Indirect object					
28	Indirect object					
29	Indirect object					
30	Indirect object					

Рисунок 23 – Список объектов PDF в iText RUPS

Особое внимание было уделено функционалу фильтрации, позволяющему быстро находить объекты по ключевым параметрам, а именно: номеру, типу и подтипу. В ходе тестирования проверялось, что фильтрация корректно обрабатывает частичные и полные запросы, мгновенно отображая только релевантные объекты исключая иные. Например, для проверки был введен запрос "/I", в результате, изображенном на рисунке 24, приложение корректно отобразило все объекты с типом /Info и подтипом /Image. Это значительно облегчает навигацию по большому объёму данных и ускоряет поиск нужных элементов в структуре PDF.



Object No.	Object	Type	Subtype	ObjStm ObjNo	ObjStm Index	Object Pos	Stream Pos	Stream Len
1	Dictionary	/Info		13879	0	1 982 (0x7BE)		
13152	Stream	/XObject	/Image			2 727 877 (0x299FC5)	2 728 017 (0x29A051)	349 (0x15D)
13257	Stream	/XObject	/Image			2 748 557 (0x29F08D)	2 748 688 (0x29F110)	16 (0x10)
13258	Stream	/XObject	/Image			2 748 722 (0x29F132)	2 748 853 (0x29F1B5)	14 (0xE)
13259	Stream	/XObject	/Image			2 748 885 (0x29F1D5)	2 749 016 (0x29F258)	14 (0xE)
13260	Stream	/XObject	/Image			2 749 048 (0x29F278)	2 749 179 (0x29F2FB)	14 (0xE)
13261	Stream	/XObject	/Image			2 749 211 (0x29F31B)	2 749 342 (0x29F39E)	14 (0xE)
13262	Stream	/XObject	/Image			2 749 374 (0x29F3BE)	2 749 505 (0x29F441)	14 (0xE)
13263	Stream	/XObject	/Image			2 749 537 (0x29F461)	2 749 668 (0x29F4E4)	14 (0xE)
13264	Stream	/XObject	/Image			2 749 700 (0x29F504)	2 749 831 (0x29F587)	16 (0x10)
13265	Stream	/XObject	/Image			2 749 865 (0x29F5A9)	2 749 996 (0x29F62C)	16 (0x10)
13266	Stream	/XObject	/Image			2 750 030 (0x29F64E)	2 750 161 (0x29F6D1)	16 (0x10)
13267	Stream	/XObject	/Image			2 750 195 (0x29F6F3)	2 750 326 (0x29F776)	16 (0x10)
13268	Stream	/XObject	/Image			2 750 360 (0x29F798)	2 750 491 (0x29F81B)	16 (0x10)
13269	Stream	/XObject	/Image			2 750 525 (0x29F83D)	2 750 656 (0x29F8C0)	16 (0x10)
13270	Stream	/XObject	/Image			2 750 690 (0x29F8E2)	2 750 821 (0x29F965)	16 (0x10)
13271	Stream	/XObject	/Image			2 750 855 (0x29F987)	2 750 986 (0x29FA0A)	16 (0x10)
13272	Stream	/XObject	/Image			2 751 020 (0x29FA2C)	2 751 151 (0x29FAAF)	16 (0x10)
13273	Stream	/XObject	/Image			2 751 185 (0x29FAD1)	2 751 316 (0x29FB54)	16 (0x10)
13274	Stream	/XObject	/Image			2 751 350 (0x29FB76)	2 751 481 (0x29FBF9)	16 (0x10)
13275	Stream	/XObject	/Image			2 751 515 (0x29FC1B)	2 751 646 (0x29FC9E)	16 (0x10)
13276	Stream	/XObject	/Image			2 751 680 (0x29FCC0)	2 751 811 (0x29FD43)	16 (0x10)
13277	Stream	/XObject	/Image			2 751 845 (0x29FDE5)	2 751 976 (0x29FDE8)	16 (0x10)
13278	Stream	/XObject	/Image			2 752 010 (0x29FF0A)	2 752 141 (0x29FF8D)	16 (0x10)

Рисунок 24 - Результат фильтрации объектов PDF

Далее была проверена точность отображения содержимого объекта, за пример взят блок метаданных /Info. Сравнение информации об авторе, дате создания, ключевых словах и названии документа с аналогичной информацией

из iText RUPS показало полное совпадение, результат можно просмотреть на рисунке 25 и рисунке 26. Это подтвердило, что приложение правильно извлекает и отображает содержимое объекта, что важно для полноценного анализа документа.

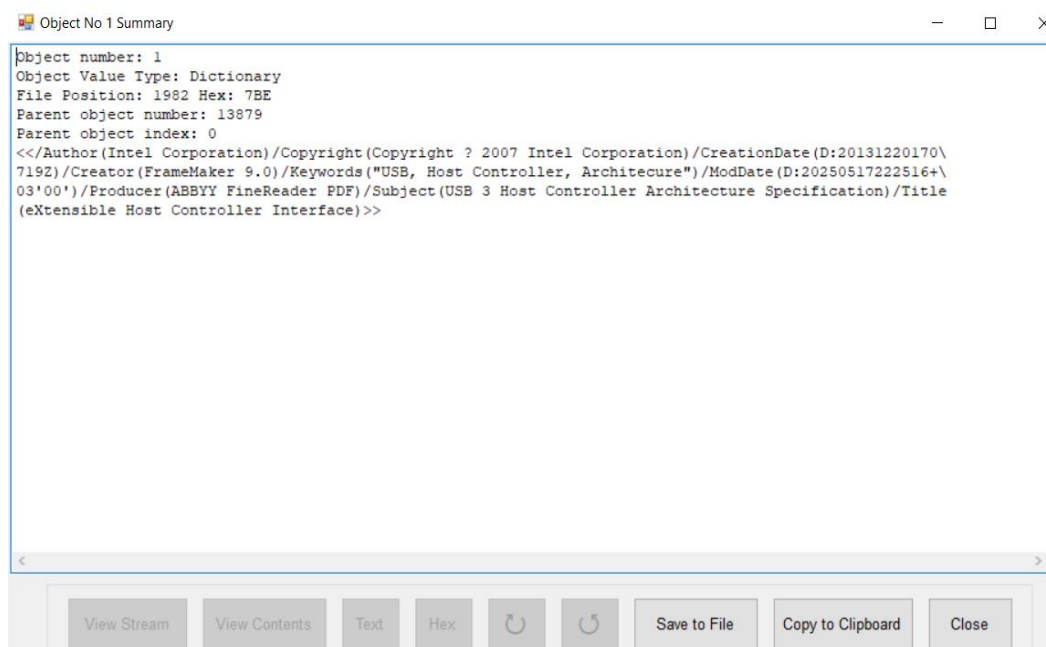


Рисунок 25 – Отображение метаданных PDF в приложении

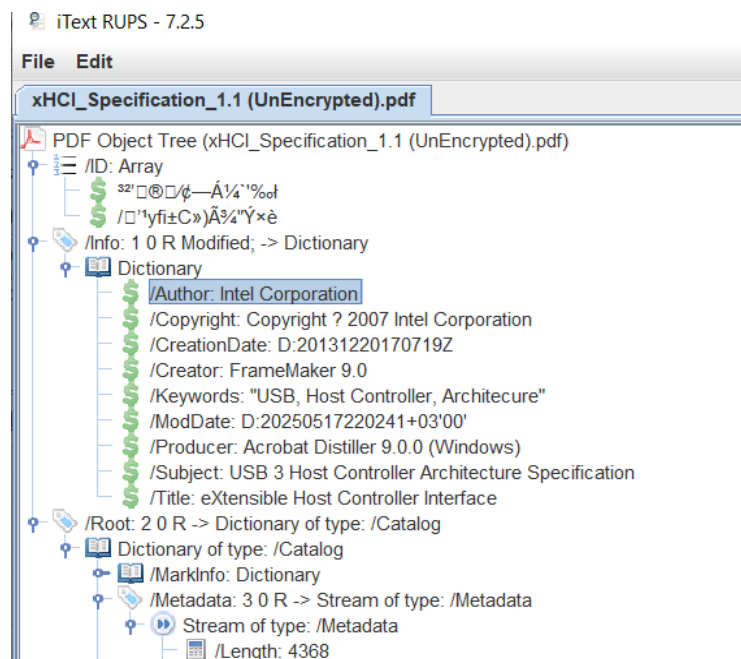


Рисунок 26 – Отображение метаданных PDF в iText RUPS

Для оценки корректности работы с объектами было выполнено сохранение содержимого выбранного объекта в отдельный файл, рисунок 27. Анализ полученного файла показал, что данные экспортированы без искажений и ошибок, сохранив свою структуру.

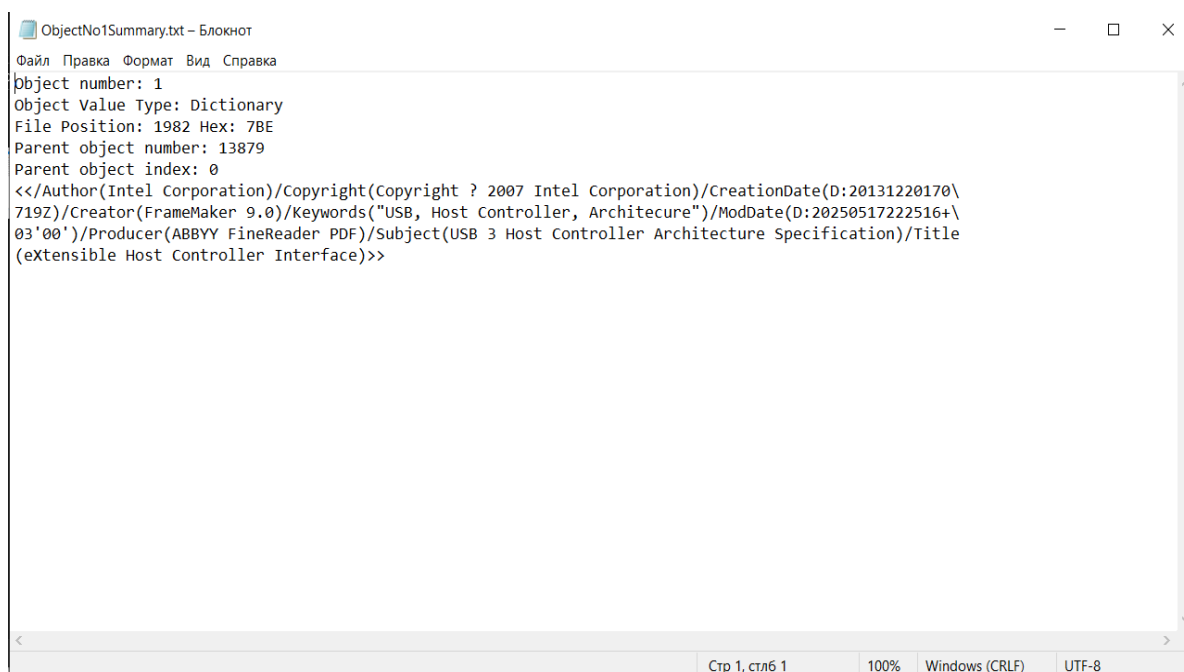


Рисунок 27 - Сохранение объекта PDF в файл

Особенно показательным этапом тестирования стало сравнение содержимого потоков (/Stream). Был выбран случайный поток из документа и проанализирован в текстовом режиме, рисунок 28. Сопоставление, с также извлечённым потоком из iText RUPS, изображенном на рисунке 29, показало абсолютное совпадение всех команд, параметров и данных, что говорит о правильности и точности разбора потокового содержимого.

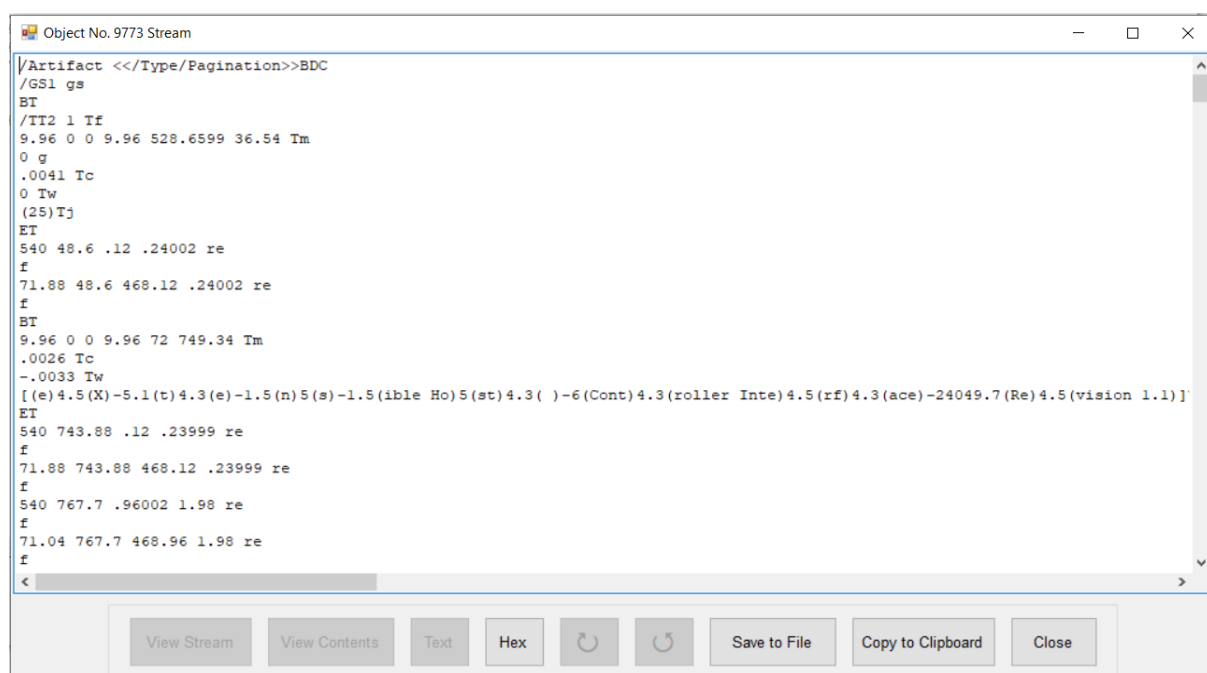


Рисунок 28 – Отображение содержимого потока PDF в приложении

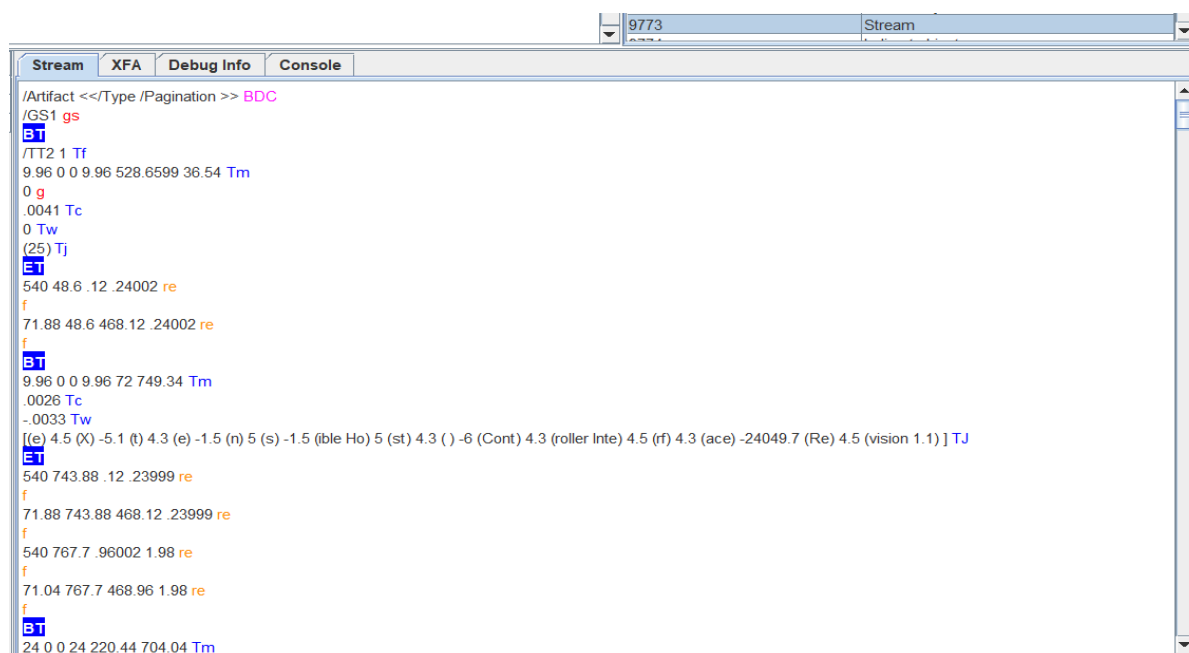


Рисунок 29 – Отображение содержимого потока PDF в iText RUPS

Таким образом, результаты тестирования подтверждают правильность и надёжность работы приложения на реальных и сложных PDF-документах, что является основой для дальнейших этапов анализа и обеспечения безопасности PDF-файлов.

3.2 Тестирование небезопасных PDF документов

Формат PDF предоставляет множество возможностей для инкапсуляции различных типов данных - текстов, изображений, звуков, скриптов и даже исполнимых файлов, что делает его не только удобным, но и опасным инструментом.

Одной из главных проблем PDF является наличие скрытых элементов в документе. Несмотря на то, что сам стандарт PDF достаточно гибкий и поддерживает различные способы структурирования и отображения содержимого, он также предоставляет злоумышленникам возможности для внедрения скрытых данных, которые могут быть не видны на первый взгляд.

3.2.1 Угрозы, выявляемые разработанным приложением

Приложение, разработанное для анализа PDF-документов, направлено на решение проблемы скрытого содержания в PDF. Путем собственных сил были созданы враждебные PDF-документы с использованием библиотеки iText 7.1. На их примере будет продемонстрировано, как приложение обнаруживает и обрабатывает такие враждебные элементы, а также какие меры могут быть предприняты для обеспечения безопасности при работе с PDF-документами. Оно может обнаружить следующие враждебные элементы:

3.2.1.1 Выявление скрытых элементов

Выявление скрытых элементов предполагает определение наличия в PDF-документе слоёв, изображений и текста, которые не отображаются и не участвуют должным образом при работе с документом. Для эффективного выявления таких элементов необходимо учитывать их взаимодействие с внутренними механизмами PDF.

Основной и самый яркий пример — это предоставление возможности использовать скрытые слои, которые могут быть включены или исключены в зависимости от настроек документа. Этот механизм используется для создания многостраничных документов, но также может быть использован для скрытия

важной информации. С помощью слоёв можно подменить содержимое документа без изменения его основной структуры. На рисунке 30 можно увидеть отображения такой pdf в на странице просмотра в Adobe. Аналогичный результат наблюдается в просмотрщике от chromium браузерх.

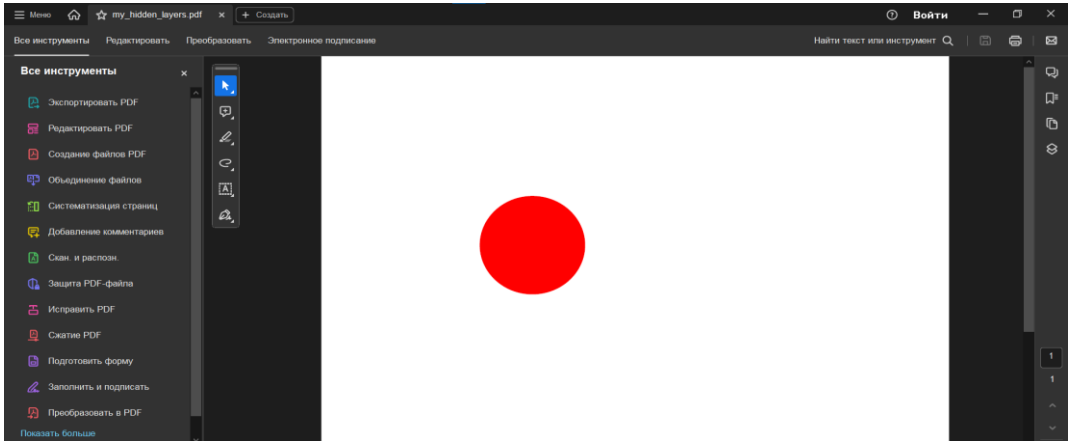


Рисунок 30 – PDF со скрытым слоем

На рисунке 31, в свою очередь изображена эта же PDF, но с убранном параметром отвечающим за скрытие слоя.

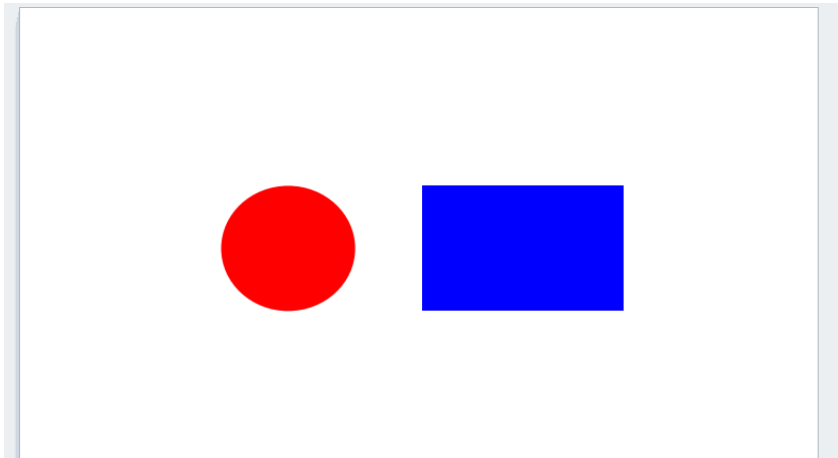


Рисунок 31 – Безопасный PDF

Помимо внедрения объектов в скрытый слой, некоторые элементы, такие как текст или изображения, могут быть внедрены в документ с использованием различных методов, включая низкую прозрачность или наложение элементов. На рисунке 32 продемонстрирована одна из таких PDF в chromium браузере.

Выделенный скрытый дает говорит о его наличии. Эти элементы могут быть замаскированы и не видны пользователю при стандартном просмотре, но они остаются в документе и могут быть использованы для фальсификации данных.

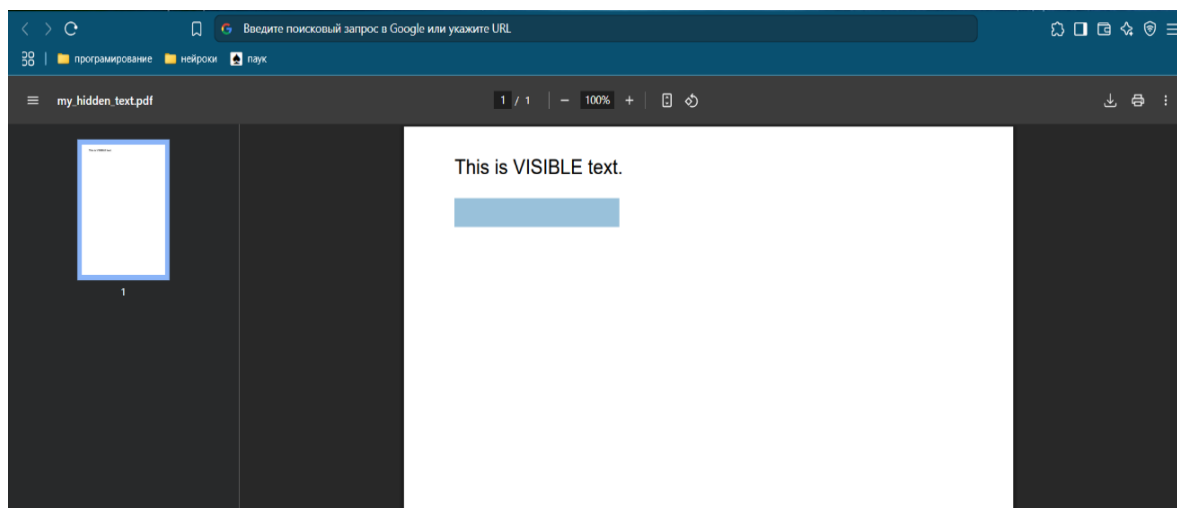


Рисунок 32 – PDF со скрытым текстом

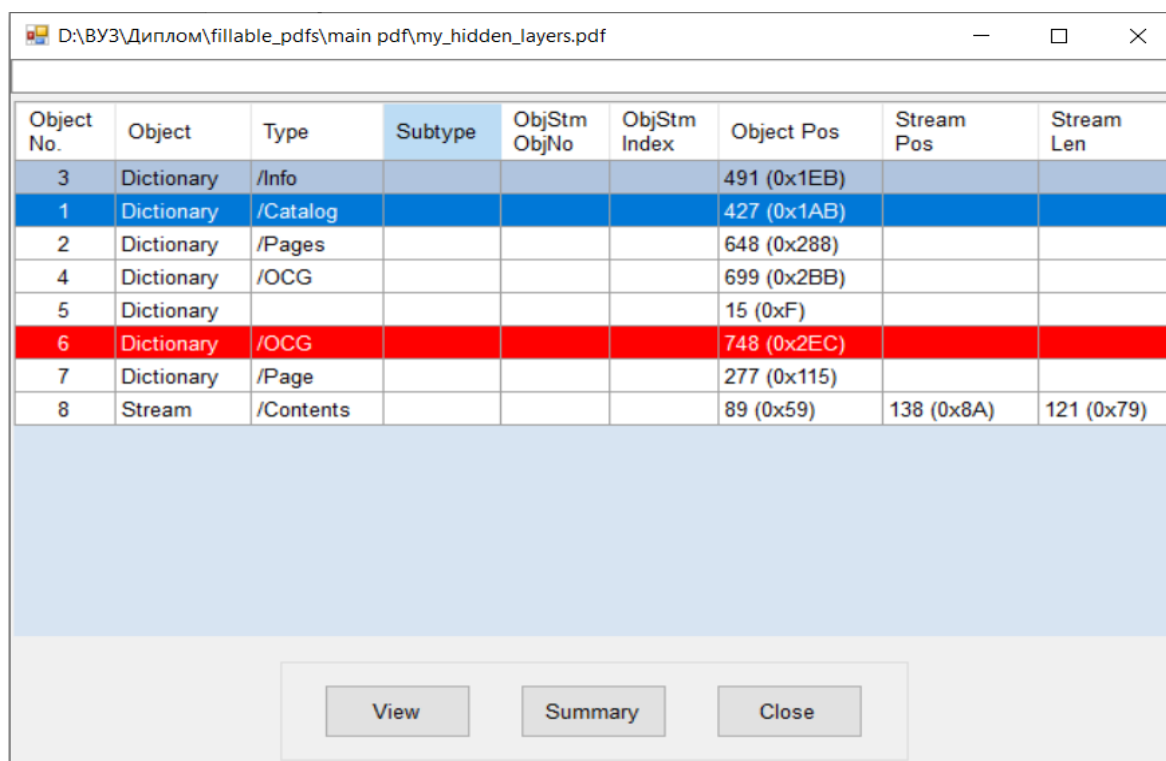
Злоумышленники могут использовать эти возможности для вставки скрытого текста или графики, которые становятся видимыми только при определённых действиях с документом. Например, при распечатывании документа или открытии его в другом приложении.

Для обнаружения таких скрытых объектов приложение ищет OCG (Optional Content Groups), которые представляют собой слои документа. В процессе анализа следует обратить внимание на поле /OCProperties, которое указывает на настройки этих слоёв. Из этого поля извлекается информация о Default-конфигурации (параметр /D), в которой хранятся данные о том, какие слои должны быть отображены по умолчанию, а какие — скрыты.

Затем приложение извлекает массив ссылок на выключенные OCG. Если в этом массиве есть ссылки на объекты, которые являются скрытыми, это сигнализирует о наличии скрытых элементов. Эти элементы могут быть использованы для манипуляций с документом, например, для замены текста или добавления нежелательных графических объектов, которые не видны

пользователю, но могут быть активированы при печати или открытии в другом приложении.

На рисунке 33, графического интерфейса приложения, показан разбор вредоносного PDF файла, где красным цветом подсвечен опасный элемент — скрытый слой, который представляет собой OCG объект.



Object No.	Object	Type	Subtype	ObjStm ObjNo	ObjStm Index	Object Pos	Stream Pos	Stream Len
3	Dictionary	/Info				491 (0x1EB)		
1	Dictionary	/Catalog				427 (0x1AB)		
2	Dictionary	/Pages				648 (0x288)		
4	Dictionary	/OCG				699 (0x2BB)		
5	Dictionary					15 (0xF)		
6	Dictionary	/OCG				748 (0x2EC)		
7	Dictionary	/Page				277 (0x115)		
8	Stream	/Contents				89 (0x59)	138 (0x8A)	121 (0x79)

Рисунок 33 – Отображение PDF со скрытым слоем в приложении

На рисунке 34 отображается содержимое вредоносного объекта. В этом случае объект типа /OCG скрывает слой с названием "Hidden Layer", что может использоваться для скрытых операций с документом.

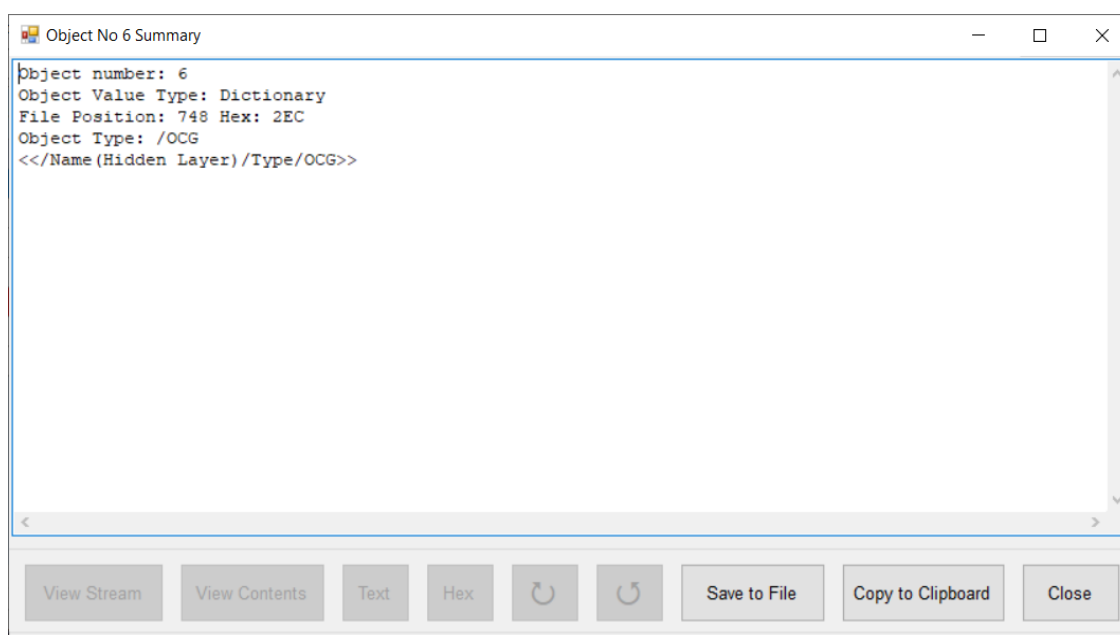


Рисунок 34 – Содержимое скрытого слоя

3.2.1.2 Выявление скрытых полей

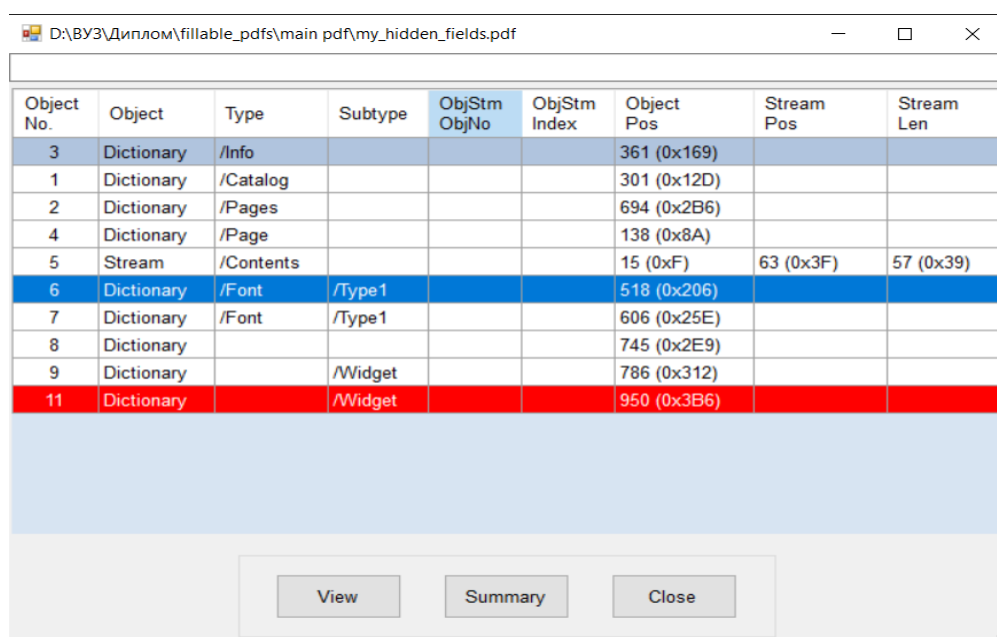
Скрытые поля, такие как текстовые поля, метки времени и другие в PDF-документах, представляют собой еще одну форму враждебного или манипулированного содержимого. Проблема с такими полями схожа с угрозами, связанными с скрытыми элементами и слоями, однако, из-за того, что поля в PDF-документах относятся к категории AcroForm, а именно к типу /Annot, они требуют совершенно другого подхода и механизмов обнаружения.

Эти поля могут нести дополнительные угрозы, так как главное отличие от скрытых слоев или элементов, которые могут быть маскируются с помощью слоев и объектов, скрытые поля являются частью формы и могут взаимодействовать с пользователем при отправке формы или при ее изменении.

Приложение для анализа PDF-документов использует проверку флага видимости, такой параметр как /F, который может указывать на скрытость поля. Также проводится анализ связи с OCG, чтобы удостовериться, не скрыто ли поле

в результате настройки слоёв. Это позволяет приложению выявлять скрытые /Widget, даже если они не видны пользователю, и предупреждать о возможных манипуляциях с содержимым документа.

Аналогично скрытым слоям на графически составляющей части приложение, рисунке 35, можно видеть что красным цветом подсвечено скрытое поле типа /Widget. В свою очередь содержимое объекта, изображенное на рисунке 36, указывает что поле скрыто, благодаря параметру “/F 6” в описании.



Object No.	Object	Type	Subtype	ObjStm ObjNo	ObjStm Index	Object Pos	Stream Pos	Stream Len
3	Dictionary	/Info				361 (0x169)		
1	Dictionary	/Catalog				301 (0x12D)		
2	Dictionary	/Pages				694 (0x2B6)		
4	Dictionary	/Page				138 (0x8A)		
5	Stream	/Contents				15 (0xF)	63 (0x3F)	57 (0x39)
6	Dictionary	/Font	/Type1			518 (0x206)		
7	Dictionary	/Font	/Type1			606 (0x25E)		
8	Dictionary					745 (0x2E9)		
9	Dictionary		/Widget			786 (0x312)		
11	Dictionary		/Widget			950 (0x3B6)		

Рисунок 35 – Отображение PDF со скрытым полем в приложении

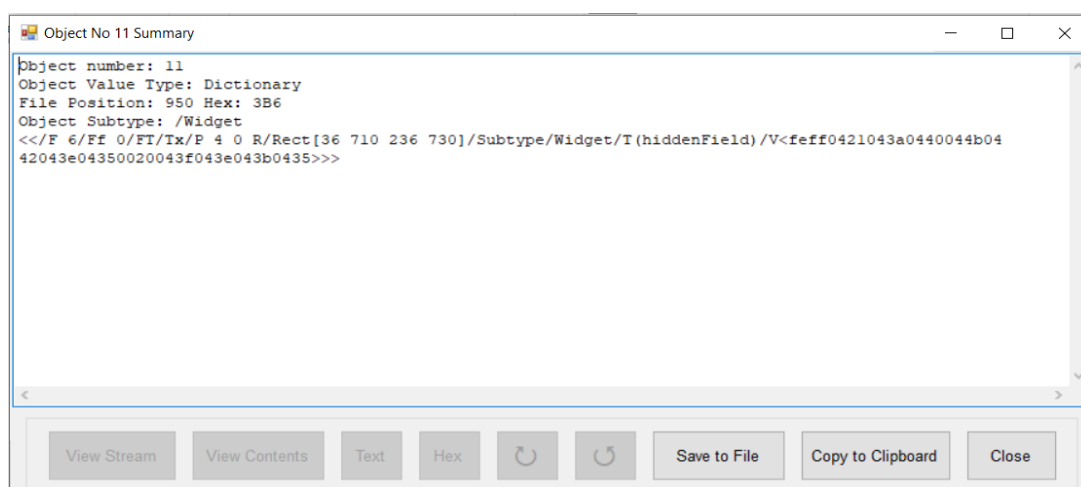


Рисунок 36 – Содержимое скрытого поля

3.2.1.3 Вредоносный JavaScript

Встроенные JavaScript-скрипты в PDF-документ могут быть использованы для выполнения вредоносных действий на компьютере пользователя, таких как удаление файлов, кража данных или загрузка дополнительного вредоносного ПО. PDF поддерживает исполнение JavaScript, что даёт возможность злоумышленникам выполнять вредоносный код, например, кража куки-файлов (Cookies), перехват ввода данных (Keylogging) или перенаправление на фишинговые сайты. Для демонстрации, одна из таких PDF вызывает alert окно браузера, на рисунке 37 результат вызова при открытии файла, а на рисунке 38, при нажатии на кнопку внутри файла.

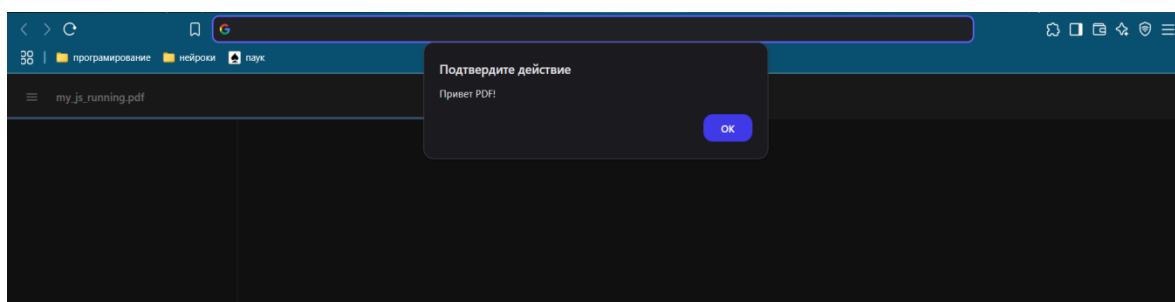


Рисунок 37 – Выполнение JavaScript кода при открытии PDF файла

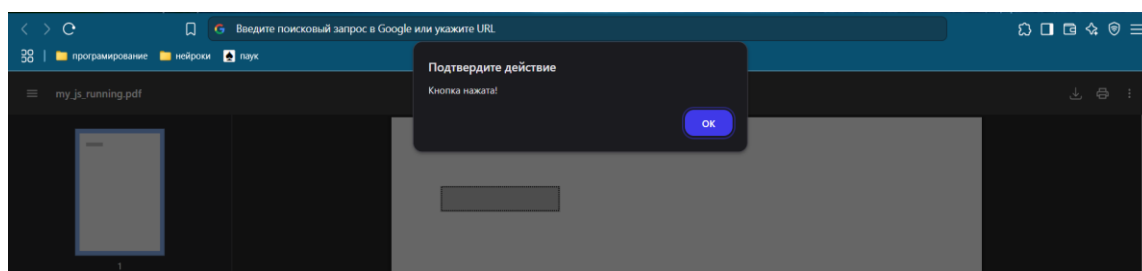
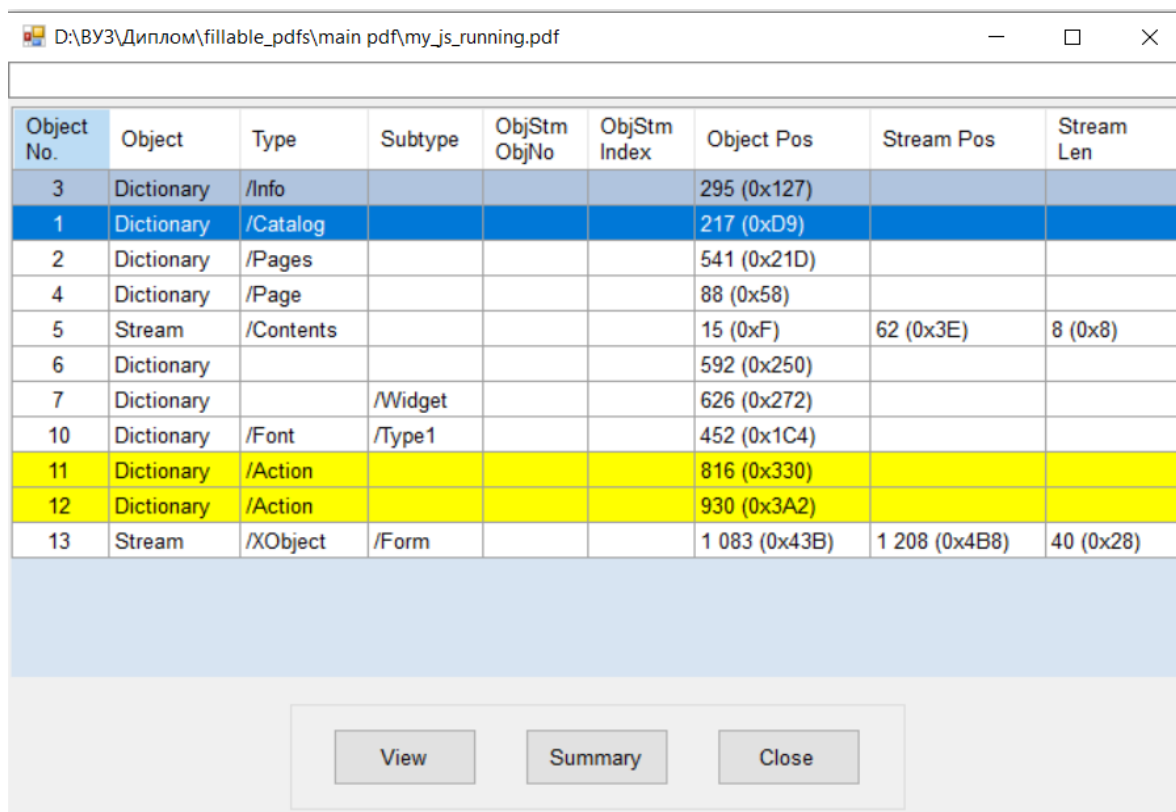


Рисунок 38 – Выполнение JavaScript кода при нажатии на кнопку

Приложение проверяет документ на наличие встроенных скриптов. Для этого оно ищет поля, связанные с JavaScript-эксплуатацией, такие как /JS (JavaScript), и помечает элемент как возможный источник проблемы. Для этого пользователю предоставляется возможность анализа содержимого и проверки исполняемого кода вместе с событием его срабатывания. Наблюдать пример

можно на рисунке 39, где желтым цветом выделены объекты с типом /Action, что указывает на наличие действия, связанного с JavaScript.



Object No.	Object	Type	Subtype	ObjStm ObjNo	ObjStm Index	Object Pos	Stream Pos	Stream Len
3	Dictionary	/Info				295 (0x127)		
1	Dictionary	/Catalog				217 (0xD9)		
2	Dictionary	/Pages				541 (0x21D)		
4	Dictionary	/Page				88 (0x58)		
5	Stream	/Contents				15 (0xF)	62 (0x3E)	8 (0x8)
6	Dictionary					592 (0x250)		
7	Dictionary		/Widget			626 (0x272)		
10	Dictionary	/Font	/Type1			452 (0x1C4)		
11	Dictionary	/Action				816 (0x330)		
12	Dictionary	/Action				930 (0x3A2)		
13	Stream	/XObject	/Form			1 083 (0x43B)	1 208 (0x4B8)	40 (0x28)

View Summary Close

Рисунок 39 – Наличие выполняемого JavaScript в приложении

На рисунке 40 показан исполнимый JavaScript код, который активируется при определенных условиях. В коде содержится строка, которая вызывает всплывающее сообщение с текстом на русском языке.

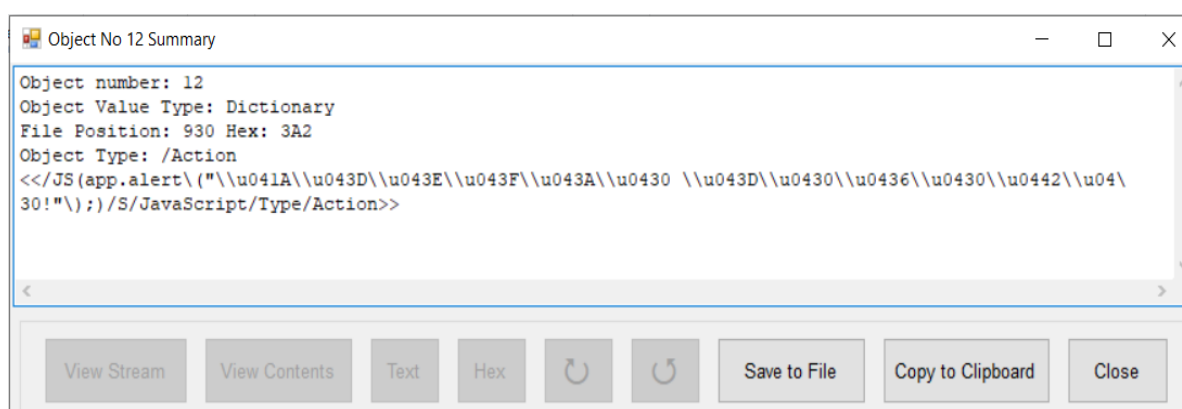


Рисунок 40 – Содержимое объекта с JavaScript кодом

Некоторые элементы контейнера AcroForm, субтипа /Widget, например, текстовые поля для ввода даты и времени, могут содержать встроенный JavaScript код. В таких полях JavaScript используется для форматирования ввода или выполнения других действий, таких как автоматическое обновление значений или проверка на корректность. Встроенный JavaScript может также управлять поведением поля, например, изменять формат отображения даты или времени в зависимости от ввода пользователя. На рисунке 41 как раз изображен пример такого элемента.

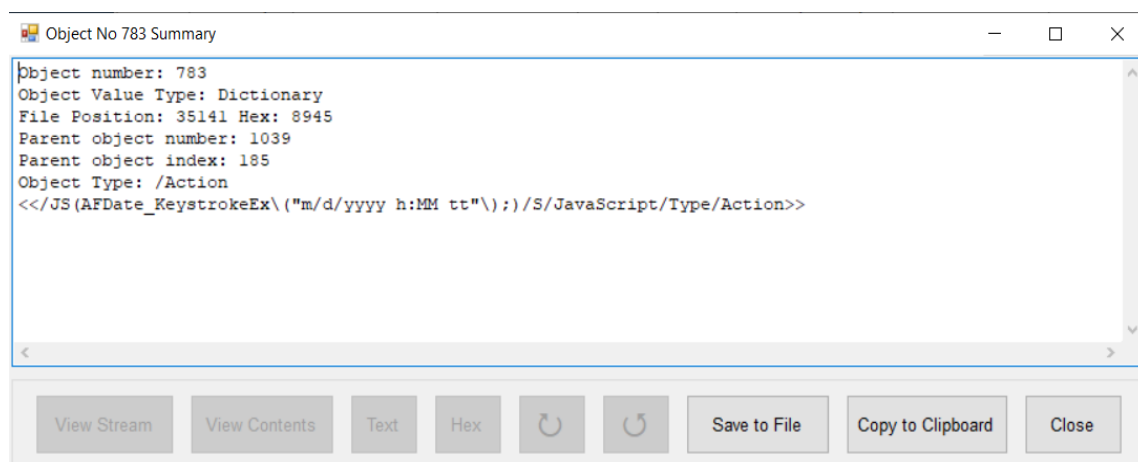


Рисунок 41 – JavaScript поля для ввода даты и времени

В результате, использование JavaScript в формах PDF для форматирования и обработки данных может быть полезным, но также требует осторожности, чтобы предотвратить возможность внедрения вредоносных скриптов.

3.2.1.4 Встраивание исполнимых файлов

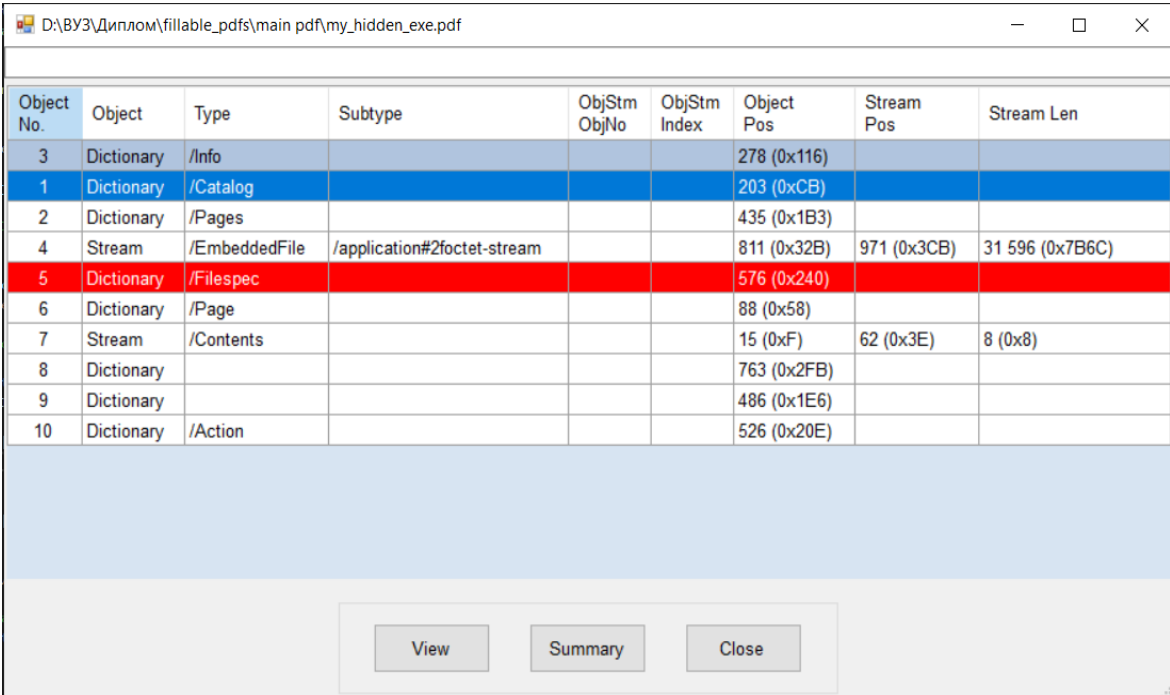
Одной из значительных угроз безопасности является встраивание исполнимых файлов в PDF-документ. Вредоносные файлы могут быть скрыты в виде вложенных объектов, таких как /EmbeddedFiles или /FileSpec, и автоматически выполняться при открытии документа или при взаимодействии с ним, например, при нажатии на скрытую ссылку или открытии вложения. Это может включать в себя такие типы файлов, как .exe, .bat, .vbs или даже скрипты,

которые при запуске могут привести к выполнению вредоносного кода на устройстве пользователя.

Опасность заключается в том, что такие исполнимые файлы могут содержать вирусы, трояны, программы для удалённого доступа, скрытые службы слежки или другие виды вредоносного ПО. Эти программы могут повредить операционную систему, вызвать утечку конфиденциальных данных, получить несанкционированный доступ к устройству или даже использовать его в атакующих целях.

Приложение проводит анализ всех вложенных файлов в PDF-документе, проверяя их на наличие исполнимых файлов. Каждый вложенный объект подвергается анализу на соответствие типам файлов, которые могут быть выполнены, и их потенциальной опасности. Если такие файлы обнаруживаются, приложение немедленно уведомляет пользователя о наличии угрозы, обеспечивая дополнительный уровень защиты от вредоносных действий.

На рисунке 42 и рисунке 43 продемонстрирована работа приложения с встраиваемыми файлами в документе.



Object No.	Object	Type	Subtype	ObjStm ObjNo	ObjStm Index	Object Pos	Stream Pos	Stream Len
3	Dictionary	/Info				278 (0x116)		
1	Dictionary	/Catalog				203 (0xCB)		
2	Dictionary	/Pages				435 (0x1B3)		
4	Stream	/EmbeddedFile	/application#2foctet-stream			811 (0x32B)	971 (0x3CB)	31 596 (0x7B6C)
5	Dictionary	/Filespec				576 (0x240)		
6	Dictionary	/Page				88 (0x58)		
7	Stream	/Contents				15 (0xF)	62 (0x3E)	8 (0x8)
8	Dictionary					763 (0x2FB)		
9	Dictionary					486 (0x1E6)		
10	Dictionary	/Action				526 (0x20E)		

Рисунок 42 – Отображение PDF с вложенным exe приложением

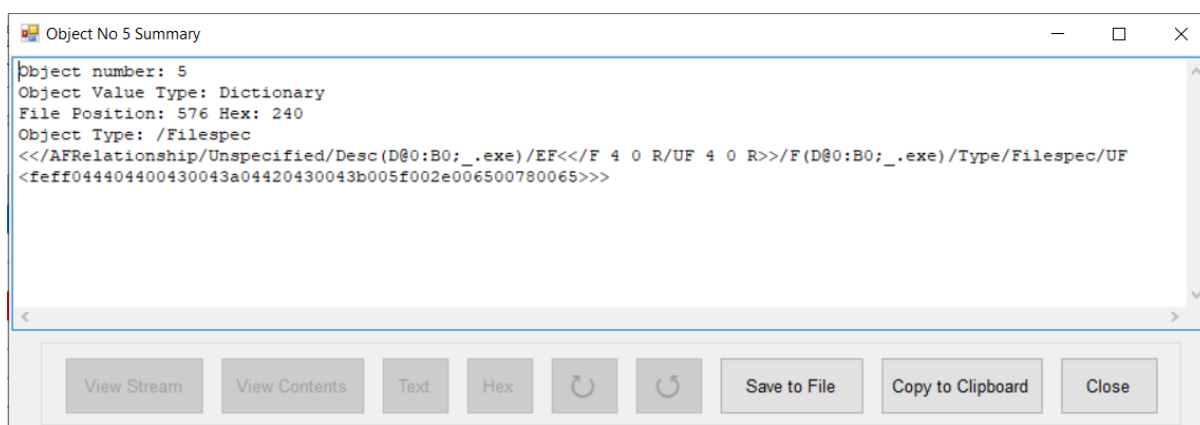


Рисунок 43 – Содержимое объекта с вложенным файлом

3.2.2 Иные угрозы

3.2.2.1 Инкрементальные обновления PDF (Append Updates)

PDF-документы имеют механизм инкрементальных обновлений, который позволяет добавлять новые данные к уже существующему документу, не изменяя его. Этот метод широко используется, например, для добавления аннотаций или обновлений к подписанным документам. Однако данный механизм представляет собой серьезную угрозу для безопасности, так как злоумышленник может воспользоваться им для скрытого изменения документа, оставляя при этом оригинальный файл на первый взгляд неизменным.

Приложение не в состоянии эффективно анализировать изменения в файлах, где использованы инкрементальные обновления. В такой ситуации злоумышленник может скрыть изменения текста или добавить новые элементы после того, как документ был подписан или зашифрован, при этом эти изменения не будут отражены в процессе стандартной верификации, так как проверка целостности будет ограничена лишь доступной версией документа, без учета обновлений. Таким образом, можно скрыть вредоносный код или изменить данные в документе, не затронув его внешнюю целостность. Примером может быть ситуация, когда изначально подписанный документ, например, договор, позднее был изменен без уведомления сторон, что может привести к юридическим последствиям или манипуляциям с информацией.

3.2.2.2 Изменение структуры документа с использованием техники "Object Streams"

PDF-документы могут использовать технику сжатия данных в виде «object streams», что позволяет эффективно хранить и передавать информацию в сжатой и упорядоченной форме. Несмотря на удобство и экономию ресурсов, данная техника может быть использована для скрытия данных внутри документа, что создает дополнительные сложности при анализе структуры PDF.

Приложение не поддерживает полный анализ так называемых «object streams». Эти потоки данных могут скрывать вредоносные объекты или даже целые разделы документа, которые становятся доступными только при определенных условиях, например, при рендеринге или при выполнении специфичных операций с документом. Вредоносный код может быть встроен в такие потоки с помощью специальной манипуляции, которая делает его доступным только в момент исполнения. Это представляет собой серьезную угрозу безопасности, так как позволяет злоумышленнику скрыть элементы документа от стандартных методов анализа, что затрудняет проверку целостности и безопасности документа.

3.2.2.3 Неизвестные и пользовательские элементы

PDF-документы обладают большой гибкостью, поддерживая не только стандартные элементы, такие как текст и изображения, но и различные пользовательские элементы, которые могут быть добавлены для расширения функционала. Эти элементы могут включать в себя скрытые формы, динамические объекты или произвольные объекты, которые используются для специфичных целей (например, для адаптации документа под конкретную задачу).

Проблема заключается в том, что пользовательские элементы не всегда могут быть распознаны стандартными инструментами для анализа. Злоумышленники могут использовать эти элементы для скрытия вредоносного кода или для манипуляций с содержимым документа без явных следов.

Например, скрытые формы могут быть использованы для сбора личных данных или выполнения непредусмотренных операций при взаимодействии с документом.

Эти элементы могут быть добавлены через различные механизмы, такие как пользовательские шрифты, скрытые текстовые поля или встроенные пользовательские объекты. Приложение для анализа PDF может не всегда корректно обрабатывать эти элементы, так как они могут не быть явно указаны в структуре документа и не всегда поддаются стандартному анализу.

3.2.2.4 Подписи с фальсификацией

Цифровые подписи в PDF являются важным элементом для обеспечения подлинности и целостности документа. Однако существующие механизмы проверки подписей могут быть уязвимы к атаке с использованием поддельных сертификатов, если злоумышленник использует фальшивые или ненадежные центры сертификации.

Приложение не в состоянии предотвратить или обнаружить такие атаки, как фальсификация подписей, если используются поддельные или ненадежные цифровые сертификаты. Это может привести к тому, что документ будет считаться действительным и неподдельным, несмотря на то что он может быть изменен или создан злоумышленниками. В таких случаях подпись, основанная на поддельном сертификате, пройдет проверку, даже если в содержимом документа были произведены изменения. Примером может служить ситуация, когда злоумышленник использует фальшивый сертификат для подделки подписи, чтобы документ выглядел как законный и подписанный официальным лицом или организацией.

3.3 Дальнейшие развитие приложения

Для улучшения функциональности и безопасности приложения возможно внесение ряда изменений, направленных на более эффективную обработку различных угроз и расширение поддерживаемых возможностей. Приложение

					ПГУ1.090301.05.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

должно поддерживать не только базовый анализ PDF, но и более глубокое взаимодействие с документами, включая их просмотр, редактирование и удаление вредоносных данных. Следуя современным тенденциям разработки PDF-риадеров рассмотрим ключевые улучшения, которые могут быть реализованы в будущем:

– для более эффективного выявления и нейтрализации угроз, описанных в пункте 3.2.2. Приложение должно быть доработано и усилено соответствующими механизмами анализа. Будет разработан механизм для более тщательной проверки всех элементов документа, включая скрытые слои, элементы и потенциально вредоносный код;

– поддержка версий PDF 2.0 и выше. Для повышения совместимости и обеспечения актуальности приложения, важно добавить поддержку форматов PDF версии 2.0 и выше. Это включает в себя такие особенности, как улучшенные возможности для защиты документов, добавление дополнительных объектов и слоев в документ, а также улучшенные механизмы для работы с шифрованием и цифровыми подписями. Внедрение этой поддержки сделает приложение более гибким и готовым к работе с современными требованиями и инновациями в области PDF-документов;

– расширение пользовательских функций приложения. В будущих обновлениях пользователям будет предоставлен удобный и интуитивно понятный интерфейс для работы с PDF-документами, который обеспечит простоту в просмотре и функциональные возможности для удобного управления страницами документа. Приложение должно включать функционал не только для анализа, но и быстрого добавления или удаления текста, аннотаций, изображений, а также автоматической оптимизации размера файла. Пользовательский интерфейс будет максимально дружелюбным, с поддержкой функции перетаскивания элементов, просмотра нескольких документов одновременно, а также гибкими настройками отображения. Это позволит

значительно упростить работу с документами, повысить производительность и сделать работу с PDF более приятной и эффективной;

– для повышения универсальности и поддержки различных отраслевых стандартов приложение должно быть адаптировано для работы с подстандартами PDF: PDF/A, PDF/X, PDF/UA, PDF/E. Их поддержка расширит возможности приложения, позволяя работать с различными типами документов, соответствующими международным стандартам;

– поддержка новых методов шифрования и возможность смены алгоритма. В следующих обновлениях приложения следует добавить расширенную поддержку современных методов шифрования, таких как AES-256, что обеспечит более высокий уровень безопасности данных. Помимо этого, необходимо реализовать функционал для работы с пользовательскими методами шифрования, позволяя менять алгоритмы шифрования, если это требуется для специфических задач;

– добавление функций для автоматизированного тестирования PDF-документов на наличие уязвимостей и угроз. Приложение может автоматически проверять документы на использование устаревших алгоритмов шифрования, уязвимости в скриптах и макросах, а также скрытые элементы, что позволит пользователям быстро обнаруживать потенциальные угрозы и принимать меры;

– внедрение функционала для автоматической оптимизации размера PDF-документа, чтобы уменьшить его объем без потери качества. Это особенно полезно для документов с большим количеством изображений или мультимедийных элементов, когда важна экономия пространства и времени на загрузку;

– блокчейн-технология быть интегрирована в приложение для обеспечения подлинности и неизменности метаданных PDF-документов. Использование блокчейна позволит проверять и верифицировать такие данные, как автор документа, дата создания или изменения, гарантируя, что они не были

подделаны после создания. Это особенно важно для юридически значимых документов и обеспечит дополнительный уровень безопасности, который невозможно подделать без изменения всей истории документа;

– использование искусственного интеллекта в приложении является одной из основных целей в будущих обновлениях. ИИ можно использовать для нескольких ключевых задач, направленных на повышение безопасности, улучшение взаимодействия с интерфейсом и автоматизацию ряда операций:

а) ИИ будет использоваться для детального анализа PDF-документов на предмет скрытых угроз, таких как вредоносные скрипты, скрытые текстовые или графические элементы, а также инкрементальные обновления. Алгоритмы машинного обучения смогут выявлять подозрительные изменения в документе и предупреждать пользователя о возможных рисках. ИИ обеспечит быстрое распознавание вредоносных кодов и объектов, что повысит безопасность работы с документами;

б) для улучшенного поиска и анализа документов ИИ будет использовать оптическое распознавание символов (OCR), чтобы эффективно работать с отсканированными PDF-документами и изображениями. Это позволит не только искать текст в традиционных документах, но и по изображениям или отсканированным страницам, расширяя возможности поиска и улучшая точность;

в) ИИ также будет выполнять роль интеллектуального помощника, помогая пользователям вносить изменения в PDF-документы. На основе контекста и задач, стоящих перед пользователем, ИИ сможет предложить варианты для удаления вредоносных элементов, таких как скрытые слои или потенциально опасные вложенные объекты. Более того, ИИ может автоматически вносить исправления или очистку документа от угроз, не требуя участия пользователя на каждом шаге.

4 Безопасность жизнедеятельности

4.1 Требования к помещениям для работы с ПК

Для комфортной и безопасной работы за компьютером необходимо правильно организовать рабочее место, которое обеспечивает не только удобство, но и способствует высокой производительности и сохранению здоровья пользователя. Правильное обустройство рабочего пространства снижает риск усталости, напряжения глаз и проблем с опорно-двигательным аппаратом.

Рабочее пространство должно быть достаточно просторным, чтобы разместить компьютер, стол, стул и другие необходимые предметы. Рекомендуется, чтобы площадь помещения составляла не менее 6-8 квадратных метров, что позволяет избежать чрезмерного скопления оборудования и обеспечить свободное движение. Помимо этого, важно правильно располагать мебель и технику, чтобы минимизировать риск травматизма и обеспечить эргономичное положение тела при работе.

Обязательным условием является наличие эффективной вентиляционной системы, которая обеспечивает постоянный приток свежего воздуха и поддерживает комфортную температуру в помещении. Хорошая вентиляция предотвращает застой воздуха, что особенно важно при длительной работе с компьютером, чтобы снизить риск возникновения головных болей и утомляемости. Кроме того, необходимо контролировать уровень влажности, так как слишком сухой или влажный воздух может негативно влиять на здоровье и работоспособность.

Для подключения компьютера и периферийных устройств должно быть достаточно электрических розеток. Электропроводка должна находиться в исправном состоянии, без повреждений или перегрузок, чтобы исключить вероятность короткого замыкания и пожара. Рекомендуется использовать

стабилизаторы напряжения и источники бесперебойного питания (ИБП), которые защищают оборудование от скачков напряжения и обеспечивают его безопасное выключение в случае перебоев в электроснабжении.

Важным аспектом является снижение уровня внешних шумов, которые способны отвлекать и снижать концентрацию пользователя. Если помещение расположено рядом с шумными объектами, например, дорогами или производственными цехами, следует применять звукоизоляционные материалы, шумоизоляционные панели или использовать наушники с шумоподавлением. Это поможет создать более комфортную рабочую обстановку и повысить эффективность работы.

Искусственное освещение должно быть организовано равномерно по всему рабочему пространству. Предпочтение следует отдавать люминесцентным лампам с рассеивателями и экранирующими решётками, которые обеспечивают мягкий и равномерный свет, минимизируя блики и тени на экране и рабочей поверхности. Использование светильников без защитных элементов может приводить к дискомфорту и зрительному напряжению. Для локального освещения допустимо применять лампы накаливания, включая галогенные, которые создают более направленный и мягкий свет. Регулярное обслуживание светильников и окон, включая очистку от пыли и замену ламп, помогает поддерживать необходимый уровень освещённости и комфортные условия для работы.

Помещения, где расположены рабочие места с компьютерами, должны иметь систему защитного заземления, которая предотвращает поражение электрическим током при неисправностях оборудования. Важно регулярно проводить влажную уборку, чтобы поддерживать чистоту и снизить распространение пыли и аллергенов, а также проветривать помещение после каждого часа работы для обновления воздуха и предотвращения концентрации углекислого газа.

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

Для поддержания комфортных условий работы необходимо обеспечить эффективные системы отопления и кондиционирования воздуха, которые позволяют поддерживать оптимальные показатели температуры (обычно от 20 до 24 градусов Цельсия) и влажности (около 40-60%). Это способствует снижению усталости и повышению общего самочувствия сотрудников.

Кроме технических и организационных мер, для безопасности жизнедеятельности важно наличие аптечки первой помощи в помещении, что позволяет оперативно оказать помощь при несчастных случаях или мелких травмах. Также следует обеспечить наличие углекислотного огнетушителя, предназначенного для тушения электрооборудования, а также установить пожарные извещатели, которые своевременно оповестят о возгорании и позволят предотвратить распространение пожара.

Все перечисленные меры не только обеспечивают безопасность и комфорт, но и способствуют повышению эффективности работы, снижению профессиональных заболеваний и улучшению общего состояния здоровья пользователей ПК.

4.2 Требования к организации рабочих мест

При организации рабочих мест с персональными компьютерами в помещениях культурно-развлекательных учреждений, где используются современные жидкокристаллические или плазменные экраны, важно соблюдать требования по площади. На каждое рабочее место следует выделять не менее 4,5 квадратных метров, что обеспечивает достаточное пространство для комфортной и безопасной работы пользователя.

Для поддержания удобного и эргономичного расположения оборудования расстояние между столами, на которых установлены мониторы, должно составлять не менее двух метров. Кроме того, боковые поверхности экранов должны находиться на расстоянии не менее 1,2 метра друг от друга, чтобы

минимизировать визуальное и физическое вмешательство соседних рабочих мест, что способствует снижению усталости глаз и повышению концентрации.

Рабочие столы рекомендуется располагать таким образом, чтобы мониторы были повернуты боковой стороной к источникам естественного освещения — окнам. Это позволяет естественному свету падать преимущественно слева, что считается оптимальным для уменьшения бликов на экране и создания комфортных условий для зрения.

В случаях, когда деятельность пользователя связана с высокой умственной нагрузкой и требует творческого подхода, целесообразно применять перегородки между рабочими местами. Оптимальная высота таких разделительных конструкций составляет от 1,5 до 2,0 метров, что обеспечивает не только визуальную, но и акустическую изоляцию, помогая сохранить концентрацию и снизить уровень внешних раздражителей.

Экран монитора должен находиться на расстоянии примерно 600–700 мм от глаз пользователя. Это расстояние обусловлено размером алфавитно-цифровых символов, которые должны быть легко различимы без излишнего напряжения глаз, при этом не рекомендуется приближать монитор ближе 500 мм. Клавиатуру целесообразно располагать на столе на расстоянии от 100 до 300 мм от его переднего края, что позволяет избежать чрезмерного напряжения рук и обеспечивает удобство набора текста.

Современные рабочие столы могут иметь различные конструкции, соответствующие требованиям эргономики и удобства. Предпочтительно использовать модели со специальной выдвижной или отдельной площадкой для клавиатуры, что помогает оптимизировать рабочее пространство. Столы могут иметь регулируемую высоту, что позволяет адаптировать рабочее место под индивидуальные особенности пользователя. Если стол фиксирован по высоте, оптимальный диапазон составляет от 680 до 800 мм.

Для обеспечения физиологически правильной позы в течение всего рабочего дня рекомендуется использовать подъемно-поворотные офисные

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

кресла, оснащённые механизмами регулировки высоты сиденья и углов наклона спинки и сиденья. Это позволяет пользователю настроить рабочее место под свои потребности, что значительно снижает риск развития мышечного и скелетного дискомфорта.

Рабочее место должно быть достаточно просторным, чтобы разместить все необходимые периферийные устройства – клавиатуру, мышь, документы и иное оборудование – без создания ощущения тесноты. Кресло при этом обязательно должно поддерживать правильную осанку, обеспечивая опору для спины, а также предотвращать появление напряжения в мышцах и суставах.

Комфорт в положении сидя достигается путем регулировки взаимного положения между рабочей поверхностью и сиденьем, включая высоту и размер стола, а также настройку высоты и угла наклона подставки для ног, если она используется. Такая регулировка способствует равномерному распределению нагрузки на тело и предотвращает переутомление.

При оборудовании рабочих мест необходимо предусмотреть возможность смены позы в течение рабочего времени. Регулярная смена позы помогает снизить усталость, улучшает кровообращение и поддерживает общее здоровье работников, что в итоге повышает их производительность и качество выполняемой работы.

Заключение

В ходе выполнения данной дипломной работы было разработано приложение для глубокого анализа и проверки безопасности PDF-документов. Приложение разработано на языке C# в среде MS Visual Studio. Используя подходы, направленные на детальное исследование структуры PDF, приложение выявляет такие угрозы, как скрытые слои, текстовые поля, встраиваемые JavaScript-скрипты и исполнимые файлы. Приложение позволяет читать и анализировать PDF-документ, разбивая его на отдельные объекты, включая страницы, шрифты, изображения и другие элементы. Это даёт пользователю возможность подробно исследовать содержимое документа.

Визуализация данных в программе осуществляется через графический интерфейс, который взаимодействует с основным функционалом через динамически подключаемую библиотеку (DLL). Графический интерфейс обеспечивает отображение структуры документа в виде таблицы объектов, показывая типы объектов, их позиции и другие метаданные.

Кроме того, разработанное приложение поддерживает работу с зашифрованными PDF-файлами, что позволяет анализировать защищённые документы. Это является одним из ключевых моментов в повышении уровня безопасности при работе с PDF-документами. Также приложение реализует проверку контрольных сумм для подтверждения целостности документа, что позволяет точно удостовериться в отсутствии изменений в содержимом файла.

В процессе разработки были освоены различные аспекты работы с форматом PDF: декодирование потоков данных; орнаментальность PDF-файлов; анализ объектов и их взаимосвязей; лексическое сканирование документов, направленное на поиск скрытых элементов. Разработка графического пользовательского интерфейса позволила улучшить навыки в визуализации всех объектов документа.

Достоинства разработанного приложения заключаются в наличии специфичных функций, таких как, сводка по документу, в которой отображаются основные характеристики файла, что делает работу с документами более удобной. Функций, обеспечивающих эффективное и наглядное разделение PDF-документа на структурированные элементы, а также функций, поддерживающих работу с зашифрованными файлами и отображения всех ключевых объектов в документе. Это делает разработанное приложение удобным инструментом для специалистов по информационной безопасности, цифровой криминалистике и разработчиков систем документооборота.

В будущем возможно улучшить приложение, добавив использование технологий машинного обучения и искусственного интеллекта для анализа документов, это может значительно улучшить процесс выявления угроз и повысить эффективность работы с документами. Еще одной важной возможностью является интеграция с другими библиотеками и фреймворками для работы с PDF, что дополнительно улучшит функциональность программы.

Таким образом, выполненная работа не только предоставила решение для анализа PDF-документов, но и открыла возможности для дальнейшего развития и улучшения инструментов безопасности, в сфере защиты данных.

Список использованных источников

1. Kaspersky Lab. PDF Exploits: Evolution and Mitigation 2020. [электронный ресурс] URL: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://content.kaspersky-labs.com/se/media/en/business-security/enterprise/KES_AEP_Corking_the_Bottleneck_How_To_Block_an_Exploit.pdf (дата обращения: 30.04.2025).

2. alexeykalina.github.io [электронный ресурс] URL: <https://alexeykalina.github.io/technologies/pdf.html> (дата обращения: 20.02.2025).

3. blog.idrsolutions.com [электронный ресурс] URL: <https://blog.idrsolutions.com/what-is-the-pdf-file-format/> (дата обращения: 20.02.2025).

4. Документация по C#: [Электронный ресурс] URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/>. (дата обращения: 05.04.2025).

5. Narayan S. Adhikari, A Comparative Study of PDF Parsing Tools Across Diverse Document Categories (Сравнительное исследование инструментов анализа PDF-файлов в различных категориях документов): JadooAI, Университет науки и технологий Миссури

6. Adobe Developer [электронный ресурс] URL: <https://developer.adobe.com/document-services/docs/overview/pdf-extract-api/howtos/extract-api/> (дата обращения: 28.04.2025).

7. «Practical PDF Analysis and Forensics» - SANS Institute; [электронный ресурс] URL: <https://digital-forensics.sans.org/media/sans-pdf-analysis-forensics-article.pdf> (дата обращения: 20.05.2025).

8. «Mastering cryptography standards for secure PDFs » - Nutrient; [электронный ресурс] URL: <https://www.nutrient.io/blog/understanding-cryptography-standards/> (дата обращения: 20.05.2025).

9. ГОСТ 12.2.007.12-88 - «ССБТ. Помещения жилые и общественные. Общие санитарно-технические требования»

10. Руководство по охране труда при работе с персональными компьютерами (Минздрав России); [электронный ресурс] URL: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://tula.rpa-mu.ru/Media/tula/2021/2021_06_05/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F%20%D0%BF%D0%BE%20%D0%BE%D1%85%D1%80%D0%B0%D0%BD%D0%B5%20%D1%82%D1%80%D1%83%D0%B4%D0%B0%20%D0%BF%D1%80%D0%B8%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B5%20%D0%BD%D0%B0%20%D0%BF%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC%20%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%B5.pdf (дата обращения: 23.05.2025).

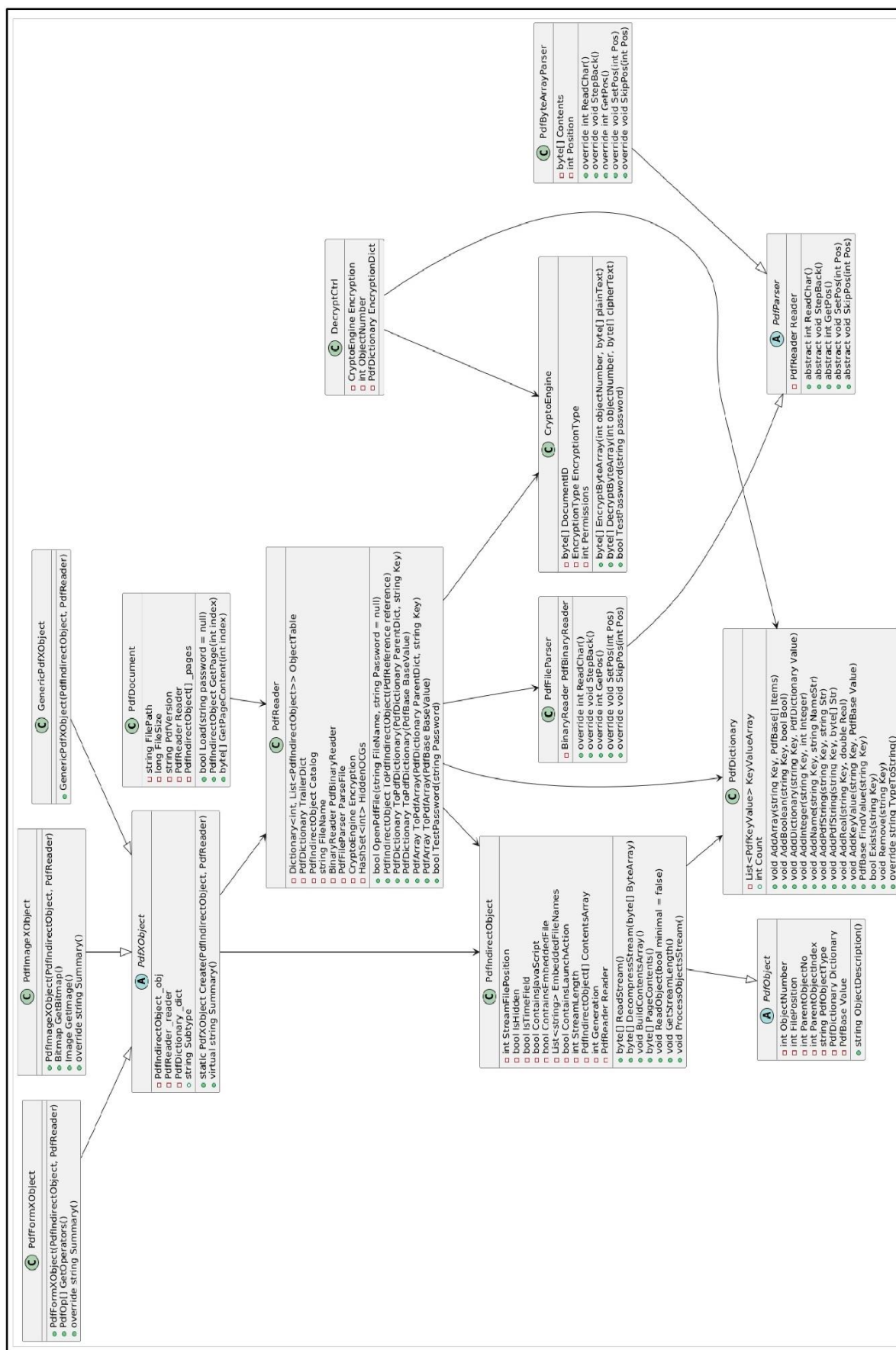
					ПГУ1.090301.05.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А
(Обязательное)

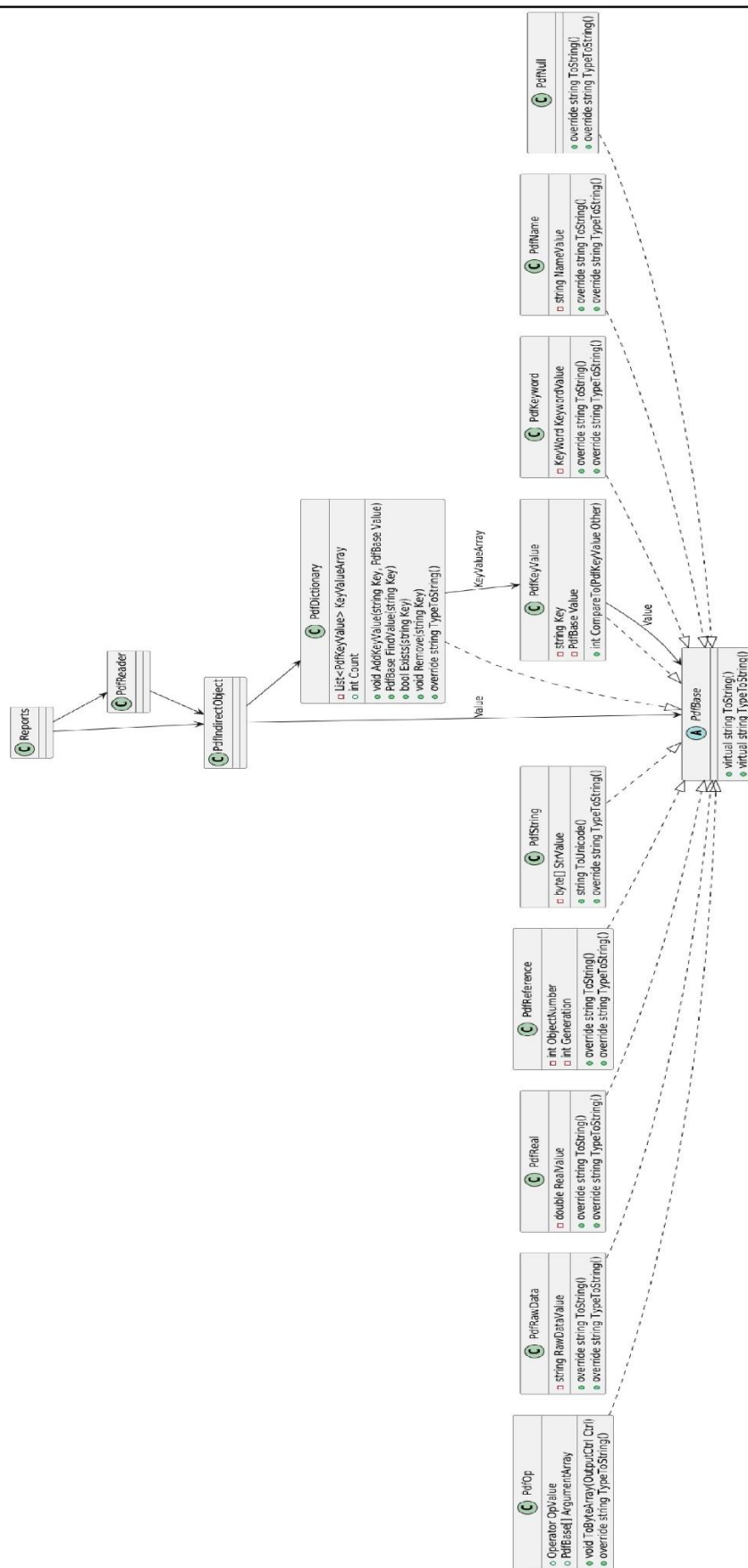
Приложение анализа безопасности pdf-файлов
UML-диаграммы

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						98
Изм.	Лист	№ докум.	Подпись	Дата		

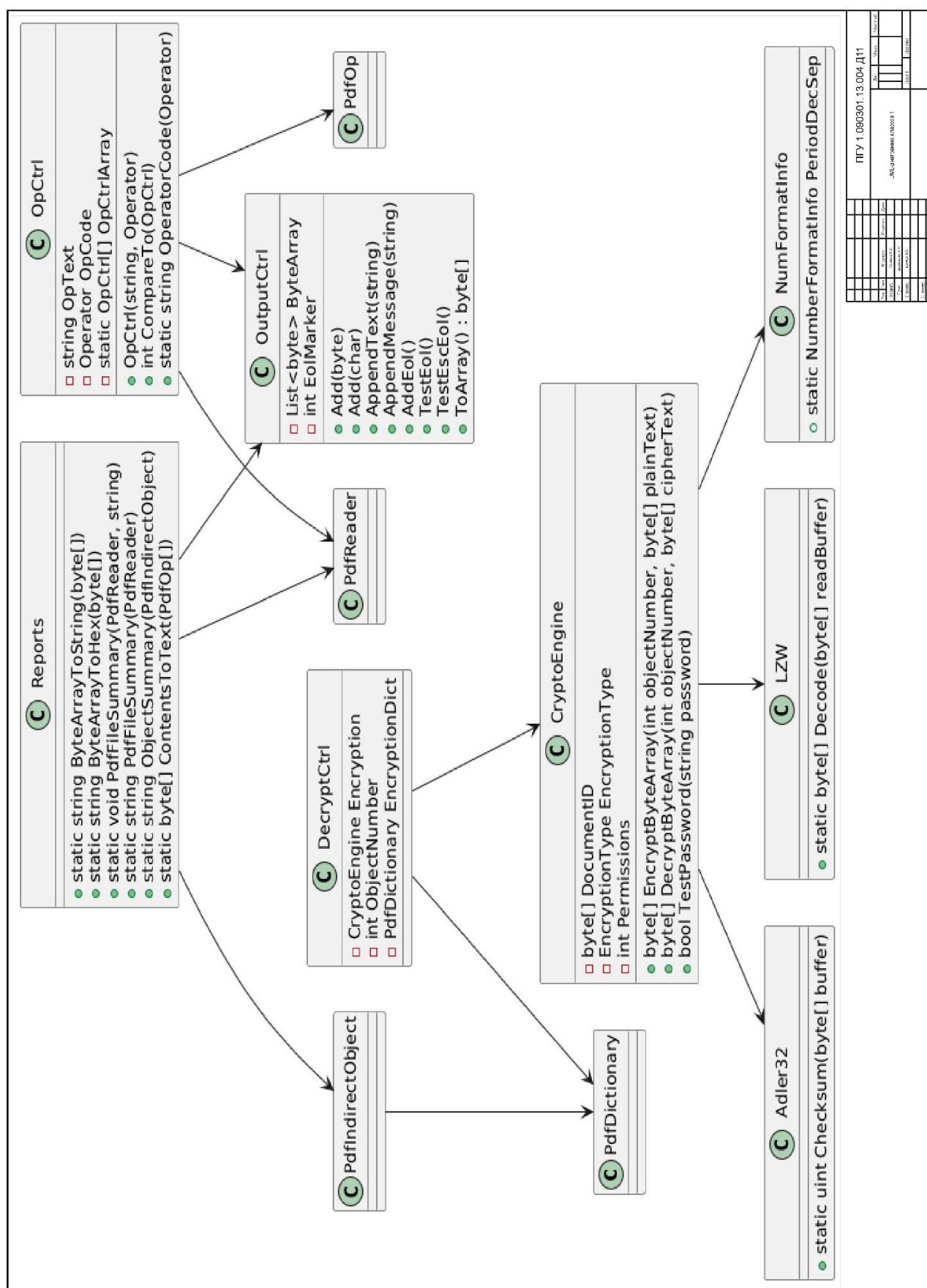
Приложение А.1 – UML – диаграмма классов 1

[illegible]

Приложение А.2 – UML – диаграмма классов 2

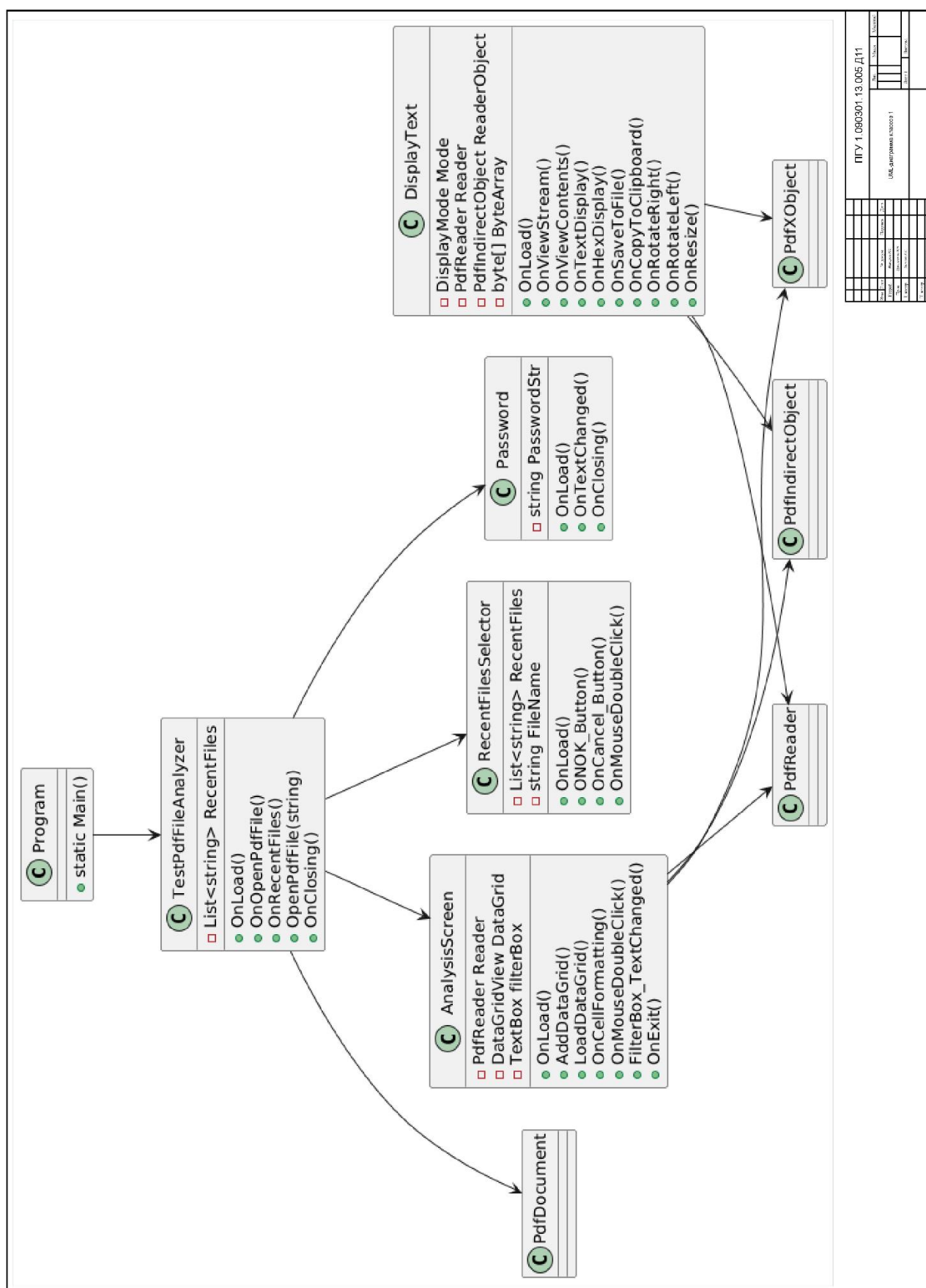
[illegible]

Приложение А.3 – UML – диаграмма классов 3



																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					</
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

Приложение А.4 – UML – диаграмма классов 4



Приложение Б
(Обязательное)

Приложение анализа безопасности pdf-файлов

Листинг программы

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						103
Изм.	Лист	№ докум.	Подпись	Дата		

Файл « PdfReader.cs»

```
using PdfFileAnalyzer.Crypto;
using PdfFileAnalyzer.Element;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;

namespace PdfFileAnalyzer
{
    public enum DecryptionStatus
    {
        FileNotProtected, // File is not protected
        OwnerPassword, // File was decrypted with owner password
        UserPassword, // File was decrypted with user password
        InvalidPassword, // Decryption failed
        Unsupported, // No support for encryption method used
    }

    public class PdfReader : IDisposable
    {
        public bool Active { get; internal set; }
        public bool InvalidPdfFile { get; internal set; }
        public DecryptionStatus DecryptionStatus { get; internal
set; }

        // Таблица всех версий объектов: ключ – номер объекта,
        значение – список всех поколений.
        public Dictionary<int, List<PdfIndirectObject>>
ObjectTable { get; private set; }
        public PdfDictionary TrailerDict { get; internal set; }
        public PdfIndirectObject Catalog { get; internal set; }
        public string FileName { get; private set; }
        public string SafeFileName { get; private set; }

        internal BinaryReader PdfBinaryReader;
        internal PdfFileParser ParseFile;
        internal int StartPosition;
        internal bool TableCrossReference;
        internal int[] ObjStmArray;

        internal PdfDictionary EncryptionDict;
        internal byte[] DocumentID;
        internal EncryptionType EncryptionType;
        internal int Permissions;
        internal byte[] OwnerKey;
        internal byte[] UserKey;
        internal CryptoEngine Encryption;
    }
}
```

```

        public HashSet<int> HiddenOCGs { get; private set; }

        public bool OpenPdfFile( string FileName, string
Password = null )
        {
            if (!FileName.EndsWith(".pdf",
StringComparison.OrdinalIgnoreCase))
                throw new ArgumentException("PDF file must have
.pdf extension");

            if (!File.Exists(FileName)) throw new
ArgumentException("PDF file does not exist");

            this.FileName = FileName;

            // safe file name is a name with no path
            SafeFileName =
FileName.Substring(FileName.LastIndexOf('\\') + 1);

            // open pdf file for reading
            PdfBinaryReader = new BinaryReader(new
FileStream(FileName, FileMode.Open, FileAccess.Read,
FileShare.Read), Encoding.UTF8);

            // create parse file object
            ParseFile = new PdfFileParser(this);

            ValidateFile();

            // search for document ID
            PdfBase TempIDArray = TrailerDict.FindValue("/ID");
            if (TempIDArray.IsArray)
            {
                // document ID is an array of two ids. Normally
the two are the same
                PdfBase[] IDArray =
((PdfArray)TempIDArray).ArrayItems;

                // take the first as the id for encryption
                if (IDArray.Length > 0 &&
IDArray[0].IsPdfString) DocumentID =
((PdfString)IDArray[0]).StringValue;
            }

            // search for /Encrypt
            PdfBase TempEncryptionDict =
TrailerDict.FindValue("/Encrypt");

            // document is not encrypted
            if (TempEncryptionDict.IsEmpty)

```

```

        {
            // document is not encrypted
            DecryptionStatus =
DecryptionStatus.FileNotProtected;

            // set reader active
            SetReaderActive();
            return true;
        }

        // value is a reference
        if (TempEncryptionDict.IsReference)
        {
            PdfIndirectObject ReaderObject =
ToPdfIndirectObject((PdfReference)TempEncryptionDict);
            if (ReaderObject != null)
            {
                ReaderObject.ReadObject();
                if (ReaderObject.ObjectType ==
ObjectType.Dictionary)
                {
                    ReaderObject.PdfObjectType =

"/Encryption";
                    TempEncryptionDict =
ReaderObject.Dictionary;
                }
            }

            if (!TempEncryptionDict.IsDictionary)
                throw new ApplicationException("Encryption
dictionary is missing");

            // save encryption dictionary
            EncryptionDict = (PdfDictionary)TempEncryptionDict;

            // decryption is not possible without document ID
            if (DocumentID == null)
                throw new ApplicationException("Encrypted
document document ID is missing.");

            // encryption method is not supported by this
library
            if (!TestEncryptionSupport())
                return false;

            // try given password or default password
            if (!TestPassword(Password))
                return false;

            return true;

```

```

    }

    internal bool TestEncryptionSupport()
    {
        // assume unsupported
        DecryptionStatus = DecryptionStatus.Unsupported;
        EncryptionType = EncryptionType.Unsupported;

        PdfBase Temp = EncryptionDict.FindValue("/Filter");
        if (!Temp.IsName || Temp.ToName != "/Standard")
            return false;

        if
        (!EncryptionDict.FindValue("/Length").GetInteger(out int KeyLen)
        || KeyLen != 128)
            return false;

        if
        (!TryResolveInteger(EncryptionDict.FindValue("/P"), out
        Permissions))
            return false;

        Temp = EncryptionDict.FindValue("/O");
        if (!Temp.IsPdfString)
            return false;
        OwnerKey = ((PdfString)Temp).StrValue;

        Temp = EncryptionDict.FindValue("/U");
        if (!Temp.IsPdfString)
            return false;
        UserKey = ((PdfString)Temp).StrValue;

        if (!EncryptionDict.FindValue("/R").GetInteger(out
        int R))
            return false;

        if (!EncryptionDict.FindValue("/V").GetInteger(out
        int V))
            return false;

        // test for AES128
        if (R == 4 && V == 4)
        {
            Temp = EncryptionDict.FindValue("/StrF");
            if (!Temp.IsName || Temp.ToName != "/StdCF")
                return false;

            Temp = EncryptionDict.FindValue("/StmF");
            if (!Temp.IsName || Temp.ToName != "/StdCF")
                return false;
        }
    }

```

```

Temp = EncryptionDict.FindValue("/CF");
if (!Temp.IsDictionary)
    return false;
PdfDictionary CFDict = Temp.ToDictionary;

Temp = CFDict.FindValue("/StdCF");
if (!Temp.IsDictionary)
    return false;
PdfDictionary StdCFDict = Temp.ToDictionary;

if
(!StdCFDict.FindValue("/Length").GetInteger(out int Len) || Len
!= 16)

    return false;

Temp = StdCFDict.FindValue("/AuthEvent");
if (!Temp.IsName || Temp.ToName != "/DocOpen")
    return false;

Temp = StdCFDict.FindValue("/CFM");
if (!Temp.IsName || Temp.ToName != "/AESV2")
    return false;

// save result
EncryptionType = EncryptionType.Aes128;
return true;
}

// test for Standard 128
if (R == 3 && V == 2)
{
    // save result
    EncryptionType = EncryptionType.Standard128;
    return true;
}

//test AES256
/*if (R == 6 && V == 5)
{
    EncryptionType = EncryptionType.Aes256;

    // Извлекаем ключи
    var o = EncryptionDict.FindValue("/O");
    var u = EncryptionDict.FindValue("/U");
    var oe = EncryptionDict.FindValue("/OE");
    var ue = EncryptionDict.FindValue("/UE");
    var perms = EncryptionDict.FindValue("/Perms");

    if (!o.IsPdfString || !u.IsPdfString ||
!oe.IsPdfString || !ue.IsPdfString || !perms.IsPdfString)

```

```

        return false;

        OwnerKey = ((PdfString)o).StrValue;
        UserKey = ((PdfString)u).StrValue;
        OwnerKeyEncrypted = ((PdfString)oe).StrValue;
        UserKeyEncrypted = ((PdfString)ue).StrValue;
        PermsEncrypted = ((PdfString)perms).StrValue;

        // salt из /O и /U
        OValidationSalt = new byte[8];
        OKeySalt = new byte[8];
        UValidationSalt = new byte[8];
        UKeySalt = new byte[8];
        Array.Copy(OwnerKey, 32, OValidationSalt, 0, 8);
        Array.Copy(OwnerKey, 40, OKeySalt, 0, 8);
        Array.Copy(UserKey, 32, UValidationSalt, 0, 8);
        Array.Copy(UserKey, 40, UKeySalt, 0, 8);

        return true;
    }*/

    return false;
}

internal PdfIndirectObject[] GetKidsArray(PdfDictionary
Dict)
{
    // get dictionary pair
    PdfBase KidsValue = Dict.FindValue("/Kids");

    // Kids value is a reference
    if (KidsValue.IsReference)
    {
        // get the object pointed by the reference
        PdfIndirectObject KidsObj =
ToPdfIndirectObject((PdfReference)KidsValue);

        // the indirect object must be an array
        if (KidsObj == null || KidsObj.ObjectType !=
ObjectType.Other || !KidsObj.Value.IsArray)
            return null;

        // replace KidsValue with array object
        KidsValue = KidsObj.Value;

        // replace /Kids with direct array object
        Dict.AddKeyValue("/Kids", KidsValue);
    }
    else if (!KidsValue.IsArray) // Kids value must be
an array or a reference

```

```

        return null;

        // array items
        PdfBase[] ReferenceArray =
((PdfArray)KidsValue).ArrayItems;

        // create result array
        PdfIndirectObject[] ResultArray = new
PdfIndirectObject[ReferenceArray.Length];

        // loop for all entries
        for (int Index = 0; Index < ReferenceArray.Length;
Index++)
        {
            // make sure we have a reference
            if (!ReferenceArray[Index].IsReference)
                return null;

            // find page or pages object
            PdfIndirectObject PageObj =
ToPdfIndirectObject((PdfReference)ReferenceArray[Index]);

            // all values in reference array must be page or
pages
            if (PageObj == null || PageObj.ObjectType !=
ObjectType.Dictionary ||
                PageObj.PdfObjectType != "/Page" &&
PageObj.PdfObjectType != "/Pages")
                return null;

            // save page object
            ResultArray[Index] = PageObj;
        }

        return ResultArray;
    }

    internal int ReadXrefTable()
    {
        // 1) Читаем блоки xref до Trailer
        while (true)
        {
            PdfBase token = ParseFile.ParseNextItem();

            if (token.ToKeyword == Keyword.Trailer)
                break;

            if (!token.GetInteger(out int firstObjNo))
                throw new ApplicationException("Cross
reference Table error");

```

```

        if (!ParseFile.ParseNextItem().GetInteger(out
int objCount))
            throw new ApplicationException("Cross
reference Table error");

        for (int i = 0; i < objCount; i++)
        {
            var posToken = ParseFile.ParseNextItem();
            var genToken = ParseFile.ParseNextItem();
            var statusToken = ParseFile.ParseNextItem();

            if (!posToken.GetInteger(out int pos))
                throw new ApplicationException($"Invalid
xref position: {posToken.ToString()}");

            if (!genToken.GetInteger(out int gen))
                throw new ApplicationException($"Invalid
generation number: {genToken.ToString()} ({genToken})");

            KeyWord status = statusToken.ToKeyWord;
            if (status != KeyWord.N && status !=
KeyWord.F)
                throw new ApplicationException($"Invalid
xref status: {statusToken}");

            if (status == KeyWord.F) continue;

            int objNo = firstObjNo + i;

            // создаём объект с учётом поколения
            var obj = new PdfIndirectObject(this, objNo,
pos)
            {
                Generation = gen,
                FilePosition = pos
            };

            // сохраняем в ObjectTable
            if (!ObjectTable.TryGetValue(objNo, out var
list))
            {
                list = new List<PdfIndirectObject>();
                ObjectTable[objNo] = list;
            }
            list.Add(obj);
        }
    }

    // 2) Trailer dictionary
    var trDict = ParseFile.ParseNextItem().ToDictionary;
    if (trDict == null)

```

```

        throw new ApplicationException("Cross reference
table missing trailer dictionary");

        // 3) Обработка /XRefStm
        if (trDict.FindValue("/XRefStm").GetInteger(out int
xRefStmPos))
        {
            SetFilePosition(xRefStmPos);
            ParseFile.ReadFirstChar();
            int r = ReadXRefStream(xRefStmPos);
            if (r != 0)
                throw new ApplicationException("/XRefStm
logic error");
        }

        // 4) Ссылка на предыдущий xref
        if (trDict.FindValue("/Prev").GetInteger(out int
prev))
            return prev;

        return 0;
    }

    internal int ReadXRefStream(int XRefPos)
    {
        // 1) Устанавливаем позицию и читаем заголовок
        объекта xref-stream
        SetFilePosition(XRefPos);
        ParseFile.ReadFirstChar();
        int XRefObjNo = ParseFile.ParseObjectRefNo();
        if (XRefObjNo <= 0)
            throw new ApplicationException("Cross reference
stream error");

        // 2) Считываем сам объект xref-stream
        var XRefObj = new PdfIndirectObject(this, XRefObjNo,
XRefPos);
        XRefObj.ReadObject();
        if (XRefObj.ObjectType != ObjectType.Stream)
            throw new ApplicationException("Cross reference
stream error");

        // 3) Если это первый xref, инициализируем
        TrailerDict и ObjectTable
        if (TrailerDict == null)
        {
            TrailerDict = XRefObj.Dictionary;
            if
(!TrailerDict.FindValue("/Size").GetInteger(out int size) ||
size == 0)

```

```

        throw new ApplicationException("Cross
reference stream error");
        InitObjectTable(size);
    }

    // 4) Получаем длину потока и параметры
    XRefObj.GetStreamLength();
    var indexArray =
XRefObj.Dictionary.FindValue("/Index").ToArrayItems;
    if (indexArray == null)
    {
        indexArray = new PdfBase[] {
new PdfInteger(0),
XRefObj.Dictionary.FindValue("/Size")
    };
    }
    var w =
XRefObj.Dictionary.FindValue("/W").ToArrayItems;
    if (w == null || w.Length != 3)
        throw new ApplicationException("XRef object
missing W array");
    int w1 = ((PdfInteger)w[0]).IntValue,
        w2 = ((PdfInteger)w[1]).IntValue,
        w3 = ((PdfInteger)w[2]).IntValue;

    // 5) Читаем и раскомпрессируем данные
    byte[] data = XRefObj.ReadStream();
    data = XRefObj.DecompressStream(data);

    // 6) Разбираем записи
    var objStmParents = new List<int>();
    int ptr = 0;
    for (int block = 0; block < indexArray.Length; block
+= 2)
    {
        int objNo =
((PdfInteger)indexArray[block]).IntValue;
        int count = ((PdfInteger)indexArray[block +
1]).IntValue;

        for (int i = 0; i < count; i++, objNo++)
        {
            int type = GetField(data, ptr, w1); ptr +=
w1;

            int field2 = GetField(data, ptr, w2); ptr +=
w2;

            int field3 = GetField(data, ptr, w3); ptr +=
w3;

            switch (type)
            {

```

```

        case 0:
            // deleted – игнорируем
            break;

        case 1:
            // нормальный объект n g R
            var obj = new
PdfIndirectObject(this, objNo, field2)
            {
                Generation = field3,
                FilePosition = field2
            };
            if (!ObjectTable.TryGetValue(objNo,
out var list1))
            {
                list1 = new
List<PdfIndirectObject>();
                ObjectTable[objNo] = list1;
            }
            list1.Add(obj);
            break;

        case 2:
            // объект из object stream
            var objStream = new
PdfIndirectObject(this, objNo, field2, field3)
            {
                Generation = 0,
                FilePosition =
XRefObj.StreamFilePosition
            };
            if (!ObjectTable.TryGetValue(objNo,
out var list2))
            {
                list2 = new
List<PdfIndirectObject>();
                ObjectTable[objNo] = list2;
            }
            list2.Add(objStream);

            // запомним parent-stream, чтобы
ProcessObjectsStream знал, что его обрабатывать
            if (!objStmParents.Contains(field2))
                objStmParents.Add(field2);
            break;

        default:
            throw new
ApplicationException("Cross reference stream error");
    }
}

```

```

    }

    // 7) Обработка object streams
    if (objStmParents.Count > 0)
    {
        ObjStmArray = objStmParents.ToArray();
    }

    // 8) Ищем ссылку на предыдущий xref
    if
(XRefObj.Dictionary.FindValue("/Prev").GetInteger(out int prev))
        return prev;
    return 0;
}
internal static int GetField( byte[] Contents, int Pos,
int Len)
{
    int Val = 0;
    for (; Len > 0; Pos++, Len--)
        Val = 256 * Val + Contents[Pos];
    return Val;
}

public void Dispose()
{
    if (PdfBinaryReader != null)
    {
        PdfBinaryReader.Close();
        PdfBinaryReader = null;
    }
    if (Encryption != null)
    {
        Encryption.Dispose();
        Encryption = null;
    }
    GC.SuppressFinalize(this);
}
}
}

```

Файл «PdfIndirectObject.cs»

```

using PdfFileAnalyzer.Crypto;
using PdfFileAnalyzer.Element;
using System;
using System.Collections.Generic;
using System.IO;
using System.IO.Compression;
using System.Linq;

```

```

namespace PdfFileAnalyzer
{
    public class PdfIndirectObject : PdfObject
    {
        public int StreamFilePosition {get; internal set;}
        public bool IsHidden { get; set; }
        public bool IsTimeField { get; private set; }
        public bool ContainsJavaScript { get; private set; }
        public bool ContainsEmbeddedFile { get; private set; }
        public List<string> EmbeddedFileNames { get; } = new
List<string>();
        public bool ContainsLaunchAction { get; private set; }
        public int StreamLength {get; internal set;}
        public PdfIndirectObject[] ContentsArray {get; internal
set;}

        public int Generation { get; internal set; }
        internal PdfReader Reader;
        internal PdfIndirectObject(PdfReader Reader,int
ObjectNumber,int FilePosition)
        {
            // save link to main document object
            this.Reader = Reader;
            this.ObjectNumber = ObjectNumber;
            this.FilePosition = FilePosition;
            return;
        }

        internal PdfIndirectObject(PdfReader Reader,int
ObjectNumber,int ParentObjectNo,int ParentObjectIndex)
        {
            // save reader parent
            this.Reader = Reader;

            // save object number
            this.ObjectNumber = ObjectNumber;

            // save parent number and position
            this.ParentObjectNo = ParentObjectNo;
            this.ParentObjectIndex = ParentObjectIndex;

            return;
        }

        public byte[] PageContents()
        {
            // build contents array
            if(ContentsArray == null)
                BuildContentsArray();

            // page has no contents
            if(ContentsArray.Length == 0)
                return new byte[0];
        }
    }
}

```

```

        // array with one item
        if (ContentsArray.Length == 1)
        {
            // read contents stream
            byte[] StreamArray =
ContentsArray[0].ReadStream();
            StreamArray =
ContentsArray[0].DecompressStream(StreamArray);
            if(StreamArray == null)
                throw new ApplicationException("Page
contents decompress error");
            return StreamArray;
        }

        // read all contents streams
        byte[] ByteArray = null;
        foreach(PdfIndirectObject ContObj in
ContentsArray)
        {
            // stream is empty
            if(ContObj.StreamLength == 0)
                continue;

            // read contents stream
            byte[] StreamArray = ContObj.ReadStream();
            StreamArray =
ContObj.DecompressStream(StreamArray);
            if(StreamArray == null)
                throw new ApplicationException("Page
contents error");

            // first contents
            if(ByteArray == null)
            {
                ByteArray = StreamArray;
                continue;
            }

            // append stream array to byte array
            int OldLen = ByteArray.Length;
            Array.Resize<byte>(ref ByteArray, OldLen +
StreamArray.Length + 1);
            ByteArray[OldLen] = (byte) '\n';
            Array.Copy(StreamArray, 0, ByteArray, OldLen
+ 1, StreamArray.Length);
        }

        if(ByteArray == null)
            ByteArray = new byte[0];

```

```

        return ByteArray;
    }

    internal void ReadObject(bool minimal = false)
    {
        // если объект удален (generation == 65535)
        if (Generation == 65535)
            return;

        // skip if done already or child of object stream
        if (ObjectType != ObjectType.Free ||
ParentObjectNo != 0)
            return;

        // set file position
        Reader.SetFilePosition(FilePosition);

        // read first byte
        Reader.ParseFile.ReadFirstChar();

        // first token must be object number "nnn 0 obj"
        if (Reader.ParseFile.ParseObjectRefNo() !=
ObjectNumber)
            throw new ApplicationException("Reading
object header failed");

        // read next token
        Value = Reader.ParseFile.ParseNextItem();

        // we have a dictionary
        if (Value.IsDictionary)
        {
            // set object value type to dictionary
            ObjectType = ObjectType.Dictionary;
            Dictionary = (PdfDictionary) Value;
            Value = null;

            // set object type if available in the
dictionary
            string ObjectTypeStr =
Dictionary.FindValue("/Type").ToName;

            if (ObjectTypeStr == "/OCG")
            {
                if (ObjectTypeStr == "/OCG" &&
Reader.HiddenOCGs != null &&
Reader.HiddenOCGs.Contains(this.ObjectNumber))
                {
                    IsHidden = true;

```

```

    }
}

// 1) Аннотация-виджет
if (Dictionary.FindValue("/Subtype").ToName
== "/Widget" &&

Dictionary.FindValue("/F").GetInteger(out int aflags))
{
    IsHidden = (aflags & (1 | 2)) != 0;
}

// 2) Поле AcroForm
if (Dictionary.FindValue("/FT").IsName &&
Dictionary.FindValue("/Ff").GetInteger(out int fflags))
{
    IsHidden |= (fflags & 0x2000) != 0;
}

// 3) Поле слой
var ocBase = Dictionary.FindValue("/OC");
if (ocBase.IsReference)
{
    int ocNo =
((PdfReference)ocBase).ObjectNumber;
    if (Reader.HiddenOCGs != null &&
Reader.HiddenOCGs.Contains(ocNo))
    {
        IsHidden = true;
    }
}

// 4) Определяем, является ли это текстовым
полем «Time»
//      FT=/Tx и по имени поля /T, содержащему
слово time
if (Dictionary.FindValue("/FT").ToName ==
"/Tx")
{
    var fieldName =
Dictionary.FindValue("/T").ToText;
    if (!string.IsNullOrEmpty(fieldName) &&
fieldName.IndexOf("time", StringComparison.OrdinalIgnoreCase) >=
0)
    {
        IsTimeField = true;
    }
}

```

```

// 5) Если это не time-поле, проверяем, есть
ли у объекта JavaScript-действия
    if (!IsTimeField)
    {
        ContainsJavaScript =
DetectJavaScriptActions();
    }

// 6) Проверка вложенных файлов: Filespec или
EF
// Сам Filespec-объект
if (Dictionary.FindValue("/Type").ToName ==
"/Filespec")
{
    ContainsEmbeddedFile = true;
    var fname =
Dictionary.FindValue("/F").ToName ??
Dictionary.FindValue("/UF").ToText;
    if (!string.IsNullOrEmpty(fname))
        EmbeddedFileNames.Add(fname);
}
// 6.2) «Embedded Files» у аннотации или
формы (/EF)
var efBase = Dictionary.FindValue("/EF");
if (efBase.IsDictionary)
{
    ContainsEmbeddedFile = true;
    foreach (var kv in
efBase.ToDictionary.KeyValueArray)
    {
        var refVal = kv.Value;
        if (refVal.IsReference)
        {
            var fsObj =
Reader.ToPdfIndirectObject((PdfReference)refVal);
            var name =
fsObj?.Dictionary.FindValue("/F").ToName;
            if
(!string.IsNullOrEmpty(name))
                EmbeddedFileNames.Add(name);
        }
    }
}

// 7) Детекция Launch-action (S /Launch)
// Ищем /A, /AA, /OpenAction на /Launch
bool IsLaunchAction(PdfDictionary act)=>
act.FindValue("/S").ToName == "/Launch";
var aBase =
Dictionary.FindValue("/A").ToDictionary;

```

```

        var aaBase =
Dictionary.FindValue("/AA").ToDictionary;
        var openBase =
Dictionary.FindValue("/OpenAction").ToDictionary;
        if ((aBase != null && IsLaunchAction(aBase))
|| (aaBase != null && aaBase.KeyValueArray.Any(kv =>
{
            var d = kv.Value.ToDictionary; return d
!= null && IsLaunchAction(d);
        }))) || (openBase != null &&
IsLaunchAction(openBase)))
        {
            ContainsLaunchAction = true;
        }

// set special object
if (ObjectTypeStr != null)
    PdfObjectType = ObjectTypeStr;

// read next token after the dictionary
Keyword Keyword =
Reader.ParseFile.ParseNextItem().ToKeyword;

// test for stream (change object from
dictionary to stream)
if(KeyWord == Keyword.Stream)
{
    // set object value type to stream
    ObjectType = ObjectType.Stream;

    // save start of stream position
    StreamFilePosition =
Reader.GetFilePosition();
}

// if it is no stream test for endobj
else if(KeyWord != Keyword.EndObj)
    throw new ApplicationException("'endobj'
token is missing");
}

// object is not a dictionary and not a stream
else
{
    ObjectType = ObjectType.Other;

    // test for endobj
    if(Reader.ParseFile.ParseNextItem().ToKeyword
!= Keyword.EndObj)
        throw new ApplicationException("'endobj'
token is missing");
}

```

```

    }

    // exit
    return;
}

internal void ProcessObjectsStream()
{
    // 1) Считываем и расшифровываем/декомпрессируем
поток
    byte[] streamBytes = ReadStream();
    streamBytes = DecompressStream(streamBytes);

    // 2) Получаем N и First
    if (!Dictionary.FindValue("/N").GetInteger(out int
objectCount) || objectCount <= 0)
        throw new ApplicationException("Object
stream: count (/N) is missing");

    if (!Dictionary.FindValue("/First").GetInteger(out
int firstOffset))
        throw new ApplicationException("Object
stream: first byte offset (/First) is missing");

    // 3) Обработка цепочки object streams (/Extends)
    PdfBase extendsBase =
Dictionary.FindValue("/Extends");
    if (extendsBase.IsReference)
        ParentObjectNo =
((PdfReference)extendsBase).ObjectNumber;

    // 4) Подготовим парсер по массиву байт
    var parser = new PdfByteArrayParser(Reader,
streamBytes, false);
    parser.ReadFirstChar();

    // 5) Массив ссылок на дочерние объекты
    PdfIndirectObject[] children = new
PdfIndirectObject[objectCount];

    // 6) Первый проход: читаем пары (ObjNo, ObjPos)
    for (int i = 0; i < objectCount; i++)
    {
        // номер объекта
        if (!parser.ParseNextItem().GetInteger(out
int objNo))
            throw new ApplicationException("Cross
reference object stream: object number error");

        // смещение внутри потока

```

```

        if (!parser.ParseNextItem().GetInteger(out
int objPos))
            throw new ApplicationException("Cross
reference object stream: object offset error");

        // 6.1) Пытаемся найти в ObjectTable
        if (!Reader.ObjectTable.TryGetValue(objNo,
out var versions))
            continue;

        // 6.2) Ищем ту запись, у которой parent ==
this и индекс == i
        var child = versions.FirstOrDefault(o =>
            o.ParentObjectNo == this.ObjectNumber &&
            o.ParentObjectIndex == i &&
            o.ObjectType == ObjectType.Free // ещё
не разобранный
        );
        if (child == null)
            continue;

        // 6.3) Сохраняем на эту позицию
        children[i] = child;
        child.FilePosition = firstOffset + objPos;
    }

    // 7) Второй проход: действительно разбираем
каждый дочерний объект
    foreach (var child in children)
    {
        if (child == null)
            continue;

        parser.SetPos(child.FilePosition);
        parser.ReadFirstChar();

        PdfBase parsed = parser.ParseNextItem();
        if (parsed.IsDictionary)
        {
            child.ObjectType =
ObjectType.Dictionary;
            child.Dictionary =
(PdfDictionary)parsed;

            // Опционально: сохраняем тип объекта из
/Type
            string typeName =
child.Dictionary.FindValue("/Type").ToName;
            if (typeName != null)
                child.PdfObjectType = typeName;
        }
    }

```

```

        else
        {
            child.ObjectType = ObjectType.Other;
            child.Value = parsed;
        }
    }

    internal static byte[] FlateDecode(byte[] InputBuffer)
    {
        // get ZLib header
        int Header = (int) InputBuffer[0] << 8 |
InputBuffer[1];

        // test header: chksum, compression method must be
deflated, no support for external dictionary
        if(Header % 31 != 0 || (Header & 0xf00) != 0x800
&& (Header & 0xf00) != 0 || (Header & 0x20) != 0)
            throw new ApplicationException("ZLIB file
header is in error");

        // output buffer
        byte[] OutputBuf;

        // decompress the file
        if((Header & 0xf00) == 0x800)
        {
            // create input stream
            MemoryStream InputStream = new
MemoryStream(InputBuffer, 2, InputBuffer.Length - 6);

            // create output memory stream to receive the
decompressed buffer
            MemoryStream OutputStream = new
MemoryStream();

            // deflate decompression object
            DeflateStream Deflate = new
DeflateStream(InputStream, CompressionMode.Decompress);
            Deflate.CopyTo(OutputStream);

            // decompressed file length
            int OutputLen = (int) OutputStream.Length;

            // create output buffer
            OutputBuf = new byte[OutputLen];

            // copy the compressed result
            OutputStream.Seek(0, SeekOrigin.Begin);
            OutputStream.Read(OutputBuf, 0, OutputLen);
            OutputStream.Close();

```

```

    }
    else
    {
        // no compression
        OutputBuf = new byte[InputBuffer.Length - 6];
        Array.Copy(InputBuffer, 2, OutputBuf, 0,
OutputBuf.Length);
    }

    // ZLib checksum is Adler32
    int ReadPtr = InputBuffer.Length - 4;
    if((((uint) InputBuffer[ReadPtr++] << 24) |
((uint) InputBuffer[ReadPtr++] << 16) |
        ((uint) InputBuffer[ReadPtr++] << 8) |
((uint) InputBuffer[ReadPtr++])) != Adler32.Checksum(OutputBuf))
        throw new ApplicationException("ZLIB
file Adler32 test failed");

    // successful exit
    return OutputBuf;
}

internal static byte[] LZWDecode(byte[] InputBuffer)
{
    // decompress
    return LZW.Decode(InputBuffer);
}

/// <summary>
/// Apply predictor decode
/// </summary>
internal byte[] PredictorDecode(byte[] InputBuffer)
{
    // test for /DecodeParams
    PdfDictionary DecodeParams =
Dictionary.FindValue("/DecodeParams").ToDictionary;

    // none found
    if(DecodeParams == null) return InputBuffer;

    // look for predictor code. if default (none or 1)
do nothing

    if(!DecodeParams.FindValue("/Predictor").GetInteger(out int
Predictor) || Predictor == 1) return InputBuffer;

    // we only support predictor code 12
    if(Predictor != 12) return null;

    // get width

```

```

        DecodeParms.FindValue("/Columns").GetInteger(out
int Width);
        if(Width < 0) throw new
ApplicationException("/DecodeParms /Columns is negative");
        if(Width == 0) Width = 1;

        // calculate rows
        int Rows = InputBuffer.Length / (Width + 1);
        if(Rows < 1) throw new
ApplicationException("/DecodeParms /Columns is greater than
stream length");

        // create output buffer
        byte[] OutputBuffer = new byte[Rows * Width];

        // reset pointers
        int InPtr = 1;
        int OutPtr = 0;
        int OutPrevPtr = 0;

        // first row (ignore filter)
        while(OutPtr < Width) OutputBuffer[OutPtr++] =
InputBuffer[InPtr++];

        // decode loop
        for(int Row = 1; Row < Rows; Row++)
        {
            // first byte is filter
            int Filter = InputBuffer[InPtr++];

            // we support PNG filter up only
            if(Filter != 2) throw new
ApplicationException("/DecodeParms Only supported filter is 2");

            // convert input to output
            for(int Index = 0; Index < Width; Index++)
                OutputBuffer[OutPtr++] = (byte) (OutputBuffer[OutPrevPtr++] +
InputBuffer[InPtr++]);
        }

        return OutputBuffer;
    }

    internal static byte[] Ascii85Decode(byte[]
InputBuffer)
    {
        // array of power of 85: 85**4, 85**3, 85**2,
85**1, 85**0
        uint[] Power85 = new uint[] {85*85*85*85,
85*85*85, 85*85, 85, 1};

```

```

// output buffer
List<byte> OutputBuffer = new List<byte>();

// convert input to output buffer
int State = 0;
uint FourBytes = 0;
for(int Index = 0; Index < InputBuffer.Length;
Index++)
{
    // next character
    char NextChar = (char) InputBuffer[Index];

    // end of stream "~>"
    if(NextChar == '~')
        break;

    // ignore white space
    if(PdfBase.IsWhiteSpace(NextChar))
        continue;

    // special case of four zero bytes
    if(NextChar == 'z' && State == 0)
    {
        OutputBuffer.Add(0);
        OutputBuffer.Add(0);
        OutputBuffer.Add(0);
        OutputBuffer.Add(0);
        continue;
    }

    // test for valid characters
    if(NextChar < '!' || NextChar > 'u')
        throw new ApplicationException("Illegal
character in ASCII85Decode");

    // accumulate 4 output bytes from 5 input
bytes
    FourBytes += Power85[State++] * (uint)
    (NextChar - '!');

    // we have 4 output bytes
    if(State == 5)
    {
        OutputBuffer.Add((byte) (FourBytes >>
24));
        OutputBuffer.Add((byte) (FourBytes >>
16));
        OutputBuffer.Add((byte) (FourBytes >>
8));
        OutputBuffer.Add((byte) FourBytes);
    }
}

```

```

        // reset state
        State = 0;
        FourBytes = 0;
    }

    // if state is not zero add one, two or three
terminating bytes
    if(State != 0)
    {
        if(State == 1)
            throw new ApplicationException("Illegal
length in ASCII85Decode");

        // add padding of 84
        for(int PadState = State; PadState < 5;
PadState++)
            FourBytes += Power85[PadState] * (uint)
('u' - '!');

        // add one, two or three terminating bytes
        OutputBuffer.Add((byte) (FourBytes >> 24));
        if(State >= 3)
        {
            OutputBuffer.Add((byte) (FourBytes >>
16));
            if(State >= 4)
                OutputBuffer.Add((byte) (FourBytes
>> 8));
        }
    }

    // exit
    return OutputBuffer.ToArray();
}

```

Файл « PdfParser.cs»

```

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Text;
using PdfFileAnalyzer.Element;

namespace PdfFileAnalyzer
{
    public class PdfParser
    {
        internal PdfReader Reader;
        internal bool ContentsStream;
    }
}

```

```

internal int NextChar;
internal int LastParsedGen = 0;

internal const int EOF = -1;

public int ParseObjectRefNo()
{
    SkipComments();
    if (NextChar < '0' || NextChar > '9') return 0;

    // Читаем номер объекта
    StringBuilder NextItem = new
StringBuilder(((char)NextChar).ToString());
    while ((NextChar = ReadChar()) != EOF &&
!PdfBase.IsDelimiter(NextChar))
        NextItem.Append((char)NextChar);

    int ObjNo;
    if (!int.TryParse(NextItem.ToString(), out ObjNo)
|| ObjNo <= 0) return 0;

    // Пропускаем пробелы
    if (!PdfBase.IsWhiteSpace(NextChar)) return 0;
    while ((NextChar = ReadChar()) != EOF &&
PdfBase.IsWhiteSpace(NextChar)) ;

    // Читаем номер поколения (Generation)
    StringBuilder GenItem = new StringBuilder();
    while (NextChar != EOF &&
char.IsDigit((char)NextChar))
    {
        GenItem.Append((char)NextChar);
        NextChar = ReadChar();
    }

    int Generation;
    if (!int.TryParse(GenItem.ToString(), out
Generation) || Generation < 0)
        return 0;

    // Пропускаем пробелы и "obj"
    if (!PdfBase.IsWhiteSpace(NextChar)) return 0;
    while ((NextChar = ReadChar()) != EOF &&
PdfBase.IsWhiteSpace(NextChar)) ;

    if (NextChar != 'o' || ReadChar() != 'b' ||
ReadChar() != 'j') return 0;
    NextChar = ReadChar();
    if (!PdfBase.IsDelimiter(NextChar)) return 0;

    // Сохраняем номер поколения

```

```

        this.LastParsedGen = Generation;

        return ObjNo;
    }
    public PdfBase ParseNextItem()
    {
        // loop in case of one or more comments
        SkipComments();

        // end of file
        if (NextChar == EOF)
            return PdfBase.Empty;

        // string
        if (NextChar == '(')
            return ParseString();

        // array
        if (NextChar == '[')
            return ParseArray();

        // hex string or dictionary
        if (NextChar == '<')
        {
            // test for dictionary
            if (ReadChar() == '<')
                return ParseDictionary(false);

            // move pointer back
            StepBack();

            // hex string
            return ParseHexString();
        }

        // next content element
        StringBuilder NextItem = new StringBuilder();
        NextItem.Append((char) NextChar);

        // add more characters until next delimiter
        while ((NextChar = ReadChar()) != EOF &&
            !PdfBase.IsDelimiter(NextChar))
            NextItem.Append((char) NextChar);

        // convert next item to string token
        string Token = NextItem.ToString();

        // name
        if (Token[0] == '/')
        {
            // empty name

```

```

        if(Token.Length == 1)
            throw new ApplicationException("Empty
name token");

        return new PdfName(Token);
    }

    // integer
    if(int.TryParse(Token, out int IntVal))
    {
        // if parsing non contents streams, an
integer can be the start of indirect reference number
        if(!ContentsStream && IntVal > 0 &&
TestReference())
            return new PdfReference(IntVal,
LastParsedGen);

        // integer
        return new PdfInteger(IntVal);
    }

    // real number with period as decimal separator
regardless of region
    if(float.TryParse(Token,
        NumberStyles.AllowDecimalPoint |
NumberStyles.AllowLeadingSign, NumFormatInfo.PeriodDecSep, out
float RealVal))
    {
        // if real number is an integer return PdfInt
object
        int TestInt = (int) Math.Truncate(RealVal);
        if(RealVal == (double) TestInt)
            return new PdfInteger(TestInt);
        return new PdfReal(RealVal);
    }

    // false
    if(Token == "false")
        return new PdfBoolean(false);

    // true
    if(Token == "true")
        return new PdfBoolean(true);

    // null
    if(Token == "null")
        return new PdfNull();

    // parse all but contents stream
    if(!ContentsStream)
    {

```

```

        // stream special case
        if(Token == "stream")
        {
            // stream must be foloowed by NL or CR
and NL
            //
            if(NextChar == '\n' || NextChar == '\r'
&& ReadChar() == '\n') return new PdfKeyword(KeyWord.Stream);
            if(NextChar == '\n')
                return new
PdfKeyword(KeyWord.Stream);
            if(NextChar == '\r')
            {
                // the PDF spec is very clear that
stream must be foloowed by NL or CR and NL
                // CR by itself is not acceptable
                if(ReadChar() != '\n')
                {
                    // HP Scanners Scanned PDF
does not conform to PDF standards
                    // step back to allow re-
parsing of the last character
                    StepBack();
                    Reader.InvalidPdfFile = true;
                }
                return new
PdfKeyword(KeyWord.Stream);
            }

            throw new ApplicationException("Stream
word must be followed by EOL");
        }

        // endstream
        if(Token == "endstream")
            return new
PdfKeyword(KeyWord.EndStream);

        // endobj
        if(Token == "endobj")
            return new PdfKeyword(KeyWord.EndObj);

        // xref
        if(Token == "xref")
            return new PdfKeyword(KeyWord.XRef);

        // xref n
        if(Token == "n")
            return new PdfKeyword(KeyWord.N);

        // xref f
        if(Token == "f")

```

```

        return new PdfKeyword(KeyWord.F);

        // trailer
        if(Token == "trailer")
            return new PdfKeyword(KeyWord.Trailer);
    }

    // parse contents stream
    else
    {
        // search for contents operator
        int OpIndex =
Array.BinarySearch(OpCtrl.OpCtrlArray, new OpCtrl(Token));

        // not found
        if(OpIndex < 0)
            throw new ApplicationException("Parsing
failed: Unknown contents operator");

        // operator enumeration
        Operator OpCode =
OpCtrl.OpCtrlArray[OpIndex].OpCode;

        // inline image
        if(OpCode == Operator.BeginInlineImage)
            return ParseInlineImage();

        // PDF operator object
        if(OpCode != Operator.BeginInlineImageData &&
OpCode != Operator.EndInlineImage)
            return new PdfOp(OpCode);
    }

    throw new ApplicationException("Parsing failed:
Unknown token: " + Token);
}
}

```

Приложение В
(Обязательное)

Приложение анализа безопасности pdf-файлов
Презентация

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						134
Изм.	Лист	№ докум.	Подпись	Дата		



Приложение анализа безопасности PDF-файлов

Чтение, разбор и выявление скрытых угроз

Автор работы:

студент гр. 21ВВП1 Макаров Илья Сергеевич Научный

Руководитель:

к.т.н., доцент Дорошенко Иван Николаевич

Слайд № 1

Цель работы

Целью данной работы является исследование структуры PDF формата и разработка программного средства анализа PDF документов, выполняющего обнаружение скрытого и вредоносного содержимого.

Слайд № 2

Анализ актуальности темы

PDF — один из наиболее популярных форматов текстовых документов, что привело к увеличению количества угроз, связанных с его использованием, таких как:

- увеличение количества уязвимостей, связанных с форматом PDF [Kaspersky Lab. PDF Exploits...];
- скрытие ресурсов и внедрение вредоносного кода благодаря гибкой структуре;
- использование PDF как точки проникновения в корпоративные и государственные системы.

Слайд № 3

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.05.001 ПЗ

Лист

135

Формат PDF



- Структура PDF-документа и его компоненты:
 - заголовок (Header);
 - тело документа (Body);
 - таблица кросс-ссылок (Cross-Reference Table или XRef-stream);
 - трейлер (Trailer);
- Безопасность PDF.
 - инкрементальные обновления;
 - шифрование: RC4, AES-128, AES-256;
 - цифровые подписи и целостность.

Слайд № 4

Существующие инструменты и библиотеки для работы с PDF

RUPS (iText)



PDFDebugger



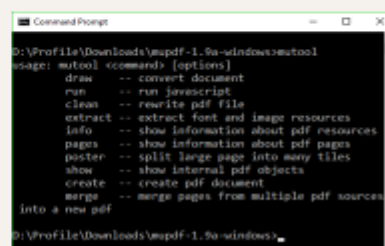
Слайд № 5

Существующие инструменты и библиотеки для работы с PDF

IPA



MuPDF



Слайд № 6

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.05.001 ПЗ

Лист
136

Сравнительный анализ существующих инструментов и библиотек

	RUPS	PDFDebugger	MuPDF	IPA
Поддержка структуры ObjStm и XRef stream	+	+/-	+	-
Получение сводки (метаданные, trailer)	+	-	-	+
Работа с зашифрованными PDF	+	+	+	-
Анализ скрытых или вредоносных объектов	-	-	-	-

Слайд № 7

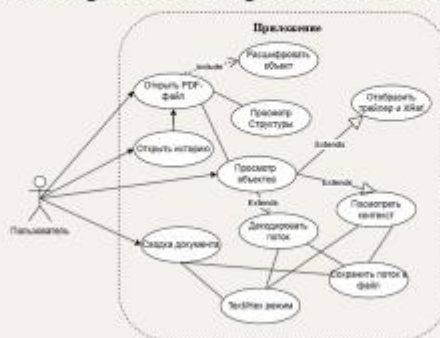
Используемые программные средства

- Язык программирования – C#
- Платформа – NET Framework
- Платформ графического интерфейса – WinForms
- Среда разработки – Visual Studio



Слайд № 8

UML диаграмма вариантов использования

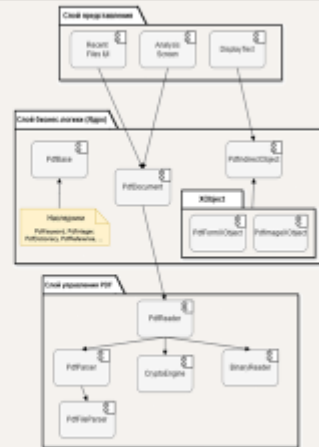


Слайд № 9

Архитектура решения

Программа разделена на три взаимосвязанных подсистемы:

- слой представления;
- слой бизнес-логики;
- слой управления PDF.



Слайд № 10

Тестирование с выявлением скрытого поля

Object No.	Object	Type	Subtype	Offset	Offset Index	Object Pos	Stream Pos	Stream Len
1	Dictionary	Page		487 (0x193)				
2	Dictionary	Page		540 (0x21C)				
4	Dictionary	Page		540 (0x21C)				
6	Dictionary	Page		10 (0x000)				
7	Dictionary	Page		277 (0x115)				
8	Stream	Content		88 (0x058)				

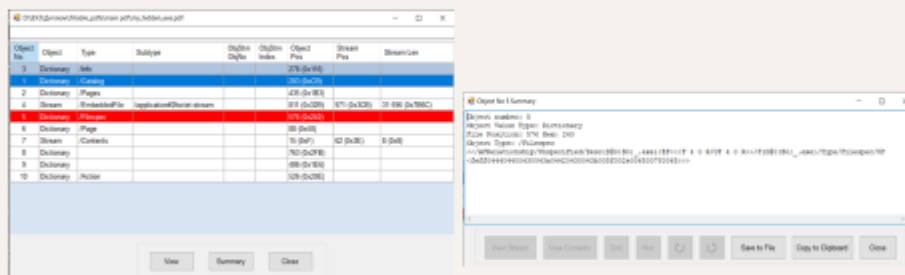
Слайд № 11

Тестирование с выявлением JS скрипта

Object No.	Object	Type	Subtype	Offset	Offset Index	Object Pos	Stream Pos	Stream Len
1	Dictionary	Page		205 (0x0D5)				
2	Dictionary	Page		277 (0x115)				
4	Dictionary	Page		541 (0x21D)				
6	Stream	Content		15 (0x00F)		42 (0x02A)		
7	Dictionary	Page		150 (0x096)				
10	Dictionary	Page		425 (0x2A9)				
11	Dictionary	Page		518 (0x206)				
12	Dictionary	Page		519 (0x207)				
13	Stream	Content		1 083 (0x42B)		1 208 (0x4B8)	88 (0x058)	

Слайд № 12

Тестирование с выявлением EXE файла



Слайд № 13

Заключение

В ходе работы было разработано приложение с графическим пользовательским интерфейсом, которое предоставляет возможность проводить анализ PDF-документов на предмет содержания скрытых объектов и исполняемого кода.

Слайд № 14

Дальнейшее развитие

Для повышения функциональности и улучшения пользовательских характеристик возможны следующие доработки:

- расширение перечня проверяемых элементов PDF-документа;
- обеспечение поддержки формата PDF 2.0 и выше;
- использование ИИ для анализа скрытых угроз, включая скрипты и инкрементные обновления;
- поддержание новых методов шифрования.

Слайд № 15

**Спасибо за
внимание!**

Слайд № 16

					ПГУ1.090301.05.001 ПЗ	Лист
						140
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Г
(справочное)

Приложение анализа безопасности pdf-файлов
Безопасность исполнения и защита от вредоносного кода

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						141
Изм.	Лист	№ докум.	Подпись	Дата		

Одной из ключевых задач при работе с PDF-документами является защита от вредоносного кода, который может быть встроен в документ в виде скрытых скриптов, вредоносных объектов или манипуляций с метаданными. Нужно быть осторожным, особенно при загрузке PDF-файлов из непроверенных источников или подозрительных электронных писем. Всегда полезно проверить, откуда они берутся, и убедиться, что они в безопасности, прежде чем открывать их.

Основным риском является возможность исполнения вредоносных JavaScript-скриптов, которые могут быть активированы при открытии PDF-документа. Существует угроза эксплуатации уязвимостей в рендерерах PDF-ридеров, что позволяет злоумышленникам выполнять произвольный код. Защита от таких угроз включает несколько слоев безопасности, как на уровне самого PDF-документа, так и на уровне используемых программных средств. Следует помнить, что обеспечение безопасности не ограничивается только использованием инструмента для анализа PDF. Важно понимать, какие меры предосторожности можно применить при работе с PDF-документами. Чтобы не стать жертвой, стоит соблюдать данные правила при работе с PDF:

- один из самых эффективных способов защиты от вредоносных скриптов в PDF — это отключение поддержки JavaScript в PDF-ридере. Многие ридеры по умолчанию поддерживают выполнение JavaScript, что может быть использовано для выполнения вредоносных действий. Отключив JavaScript, можно значительно снизить риск исполнения нежелательных скриптов, встроенных в документ;

- важно использовать только те PDF-ридеры, которые регулярно обновляются и имеют встроенные механизмы безопасности. Например, современные версии Adobe Acrobat и другие популярные ридеры могут блокировать выполнение вредоносных JavaScript-скриптов и предотвращать

эксплуатацию известных уязвимостей. Обновления ридеров обеспечивают исправления для уязвимостей, которые могут быть использованы для атаки;

– если PDF-документ имеет цифровую подпись, необходимо обязательно проверять подлинность подписи и целостность документа. Цифровая подпись гарантирует, что документ не был изменен после подписания. Важно удостовериться, что подпись не является поддельной и что подписанный документ соответствует его заявленному содержанию;

– многие антивирусные программы могут сканировать PDF-документы на наличие вредоносных элементов, таких как скрытые скрипты, исполнимые файлы или вредоносные объекты. Регулярное использование антивирусного ПО и выполнение полного сканирования на наличие угроз поможет обнаружить и нейтрализовать потенциальные угрозы до того, как они смогут нанести вред;

– очень важно избегать открытия PDF-документов, полученных из ненадежных или непроверенных источников, таких как подозрительные электронные письма или неизвестные веб-сайты. Мошенники часто используют фишинг-атаки, отправляя вредоносные PDF-документы с целью установки вирусов или кражи данных. Открывать документы нужно только от доверенных отправителей и с проверенных источников;

– следует регулярно обновлять как сам PDF-ридер, так и операционную систему. Обновления часто включают исправления уязвимостей, которые могут быть использованы злоумышленниками для атак. Регулярные патчи и обновления обеспечивают защиту от наиболее известных угроз;

– для защиты документа от изменений и несанкционированного доступа можно использовать возможности PDF для ограничения прав на редактирование, копирование, печать и другие действия. Это можно сделать с помощью пароля или других механизмов защиты, встроенных в формат PDF. Важно не только применять такие ограничения, но и использовать сильные пароли, чтобы предотвратить взлом защиты;

– для более глубокого анализа и безопасности документов можно использовать специализированные инструменты для проверки скрытых угроз в PDF, такие как антивирусы, программы для анализа структуры документа или инструменты для анализа метаданных. Это поможет выявить потенциально опасные элементы до того, как документ будет открыт;

– многие PDF-документы могут содержать вложенные файлы, такие как изображения, шрифты или даже другие документы. Эти вложения могут быть использованы для внедрения вредоносных программ или эксплуатации уязвимостей. Перед открытием вложений важно убедиться, что они не содержат опасных элементов.

					ПГУ1.090301.05.001 ПЗ	Лист
						144
Изм.	Лист	№ докум.	Подпись	Дата		