**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**(РУТ (МИИТ)**

«Высшая инженерная школа»

**ОТЧЁТ**

по дисциплине

**«Информационная безопасность»**

**Выполнил:**

студент группы \_\_\_\_\_\_\_\_\_

Осипенко А. И.

**Принял:**

преподаватель АВИШ

Коваль Н. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2025

Оглавление

[**Отчет по разработке Telegram-бота для мониторинга активности на компьютере** 3](#_Toc197909834)

[1. Цели и задачи проекта 3](#_Toc197909835)

[2. Техническая спецификация 3](#_Toc197909836)

[3. Настройка системного окружения 4](#_Toc197909837)

[3.1 Аудиты, настроенные через auditpol 4](#_Toc197909838)

[3.2 Аудиты, настроенные через локальные политики безопасности (secpol.msc) 5](#_Toc197909839)

[3.3 Разница между настройкой через auditpol и локальные политики 6](#_Toc197909840)

[3.4 Настройка Sysmon 7](#_Toc197909841)

[4. Выбор технологий и подходов 7](#_Toc197909842)

[5. Архитектура и реализация 8](#_Toc197909843)

[5.1 Алгоритм работы 8](#_Toc197909844)

[5.2 Структура проекта 8](#_Toc197909845)

[6. Руководство по установке и эксплуатации 13](#_Toc197909846)

[7. Ограничения и возможности оптимизации 14](#_Toc197909847)

[8. Приобретенные знания и навыки 14](#_Toc197909848)

[Заключение 15](#_Toc197909849)

**Отчет по разработке Telegram-бота для мониторинга активности на компьютере: https://github.com/NSAlexeyMK/noblindezone**

## Цели и задачи проекта

Цель проекта — Отслеживать ключевые системные события в операционной системе Windows. Для этого требуется разработать Telegram-бота с возможностью отправки уведомлений в Telegram и генерации PDF-отчетов. Основные задачи:

* Сообщать о включении компьютера (Event ID 6005).
* Отслеживать входы в систему (Event ID 4624, типы 2, 7, 10) и повышение привилегий (Event ID 4672).
* Уведомлять о создании задач в планировщике (Event ID 4698).
* Следить за установкой и изменением служб (Event IDs 4697, 7045).
* Проверять запущенные процессы через VirusTotal (Sysmon Event ID 1).

Проект направлен на изучение возможностей обеспечения контроля за безопасностью и активностью системы.

## 2. Техническая спецификация

Бот разработан для среды Windows 10 Pro (версия 10.0.19045, сборка 19045) на виртуальной машине (VirtualBox, 8 CPU, 12,281 МБ RAM, 60 ГБ диск). Использованы следующие технологии:

* **Язык**: Python 3.12.9.
* **Библиотеки**:
  + - python-telegram-bot 22.0 — отправка сообщений и документов в Telegram.
    - pywin32 310 — чтение журналов Windows.
    - requests 2.32.3 — запросы к VirusTotal API.
    - xmltodict 0.14.2 — парсинг XML-событий Sysmon.
    - reportlab 4.3.1 — генерация PDF-отчетов.
* **Требования**:
  + - Права администратора для чтения журналов безопасности.
    - API-ключи для Telegram и VirusTotal.
    - Установленный Sysmon64 для мониторинга процессов.

## 3. Настройка системного окружения

Для мониторинга событий настроены аудиты через командную утилиту auditpol и локальные политики безопасности (secpol.msc). Эти механизмы обеспечивают фиксацию системных событий в журналах Windows (System, Security, Microsoft-Windows-Sysmon/Operational).

### 3.1 Аудиты, настроенные через auditpol

Следующие подкатегории аудита включены для отслеживания событий, указанных в задании:

* **Изменение состояния безопасности** ({0CCE9210-69AE-11D9-BED3-505054503030}):
  + - **Команда**: auditpol /set /subcategory:"{0CCE9210-69AE-11D9-BED3-505054503030}" /success:enable /failure:enable
    - **Назначение**: фиксирует событие 6005 (запуск службы журнала событий), указывающее на включение компьютера. Генерирует минимальное количество логов из-за редкости событий.
* **Вход в систему** ({0CCE9215-69AE-11D9-BED3-505054503030}):
  + - **Команда**: auditpol /set /subcategory:"{0CCE9215-69AE-11D9-BED3-505054503030}" /success:enable /failure:enable
    - **Назначение**: отслеживает событие 4624 (успешный вход) для типов входа 2 (интерактивный), 7 (разблокировка), 10 (удаленный). Требует фильтрации из-за большого объема логов.
* **Объект-задание** ({0CCE921F-69AE-11D9-BED3-505054503030}):
  + - **Команда**: auditpol /set /subcategory:"{0CCE921F-69AE-11D9-BED3-505054503030}" /success:enable /failure:enable
    - **Назначение**: фиксирует событие 4698 (создание задач в планировщике). Позволяет выявлять потенциально вредоносные задачи. Генерирует умеренное количество логов.
* **Другие события доступа к объектам** ({0CCE9227-69AE-11D9-BED3-505054503030}):
  + - **Команда**: auditpol /set /subcategory:"{0CCE9227-69AE-11D9-BED3-505054503030}" /success:enable /failure:enable
    - **Назначение**: отслеживает событие 4697 (установка новой службы). Помогает выявлять несанкционированные службы. Может генерировать много логов в активных системах.
* **Использование прав, затрагивающее конфиденциальные данные** ({0CCE9228-69AE-11D9-BED3-505054503030}):
  + - **Команда**: auditpol /set /subcategory:"{0CCE9228-69AE-11D9-BED3-505054503030}" /success:enable /failure:enable
    - **Назначение**: фиксирует событие 4672 (назначение привилегий, например, SeTcbPrivilege). Критично для мониторинга повышения прав. Генерирует умеренное количество логов.
* **Использование прав, не затрагивающее конфиденциальные данные** ({0CCE9229-69AE-11D9-BED3-505054503030}):
  + - **Команда**: auditpol /set /subcategory:"{0CCE9229-69AE-11D9-BED3-505054503030}" /success:enable /failure:enable
    - **Назначение**: также отслеживает событие 4672 (привилегии, например, SeImpersonatePrivilege). Дополняет мониторинг, но генерирует больше логов.
* **Создание процесса** ({0CCE922B-69AE-11D9-BED3-505054503030}):
  + - **Команда**: auditpol /set /subcategory:"{0CCE922B-69AE-11D9-BED3-505054503030}" /success:enable /failure:enable
    - **Назначение**: поддерживает Sysmon Event ID 1 (создание процесса). Дополняет мониторинг процессов, но требует фильтрации из-за большого объема логов.

### 3.2 Аудиты, настроенные через локальные политики безопасности (secpol.msc)

Помимо auditpol, в локальных политиках безопасности (secpol.msc) включены следующие категории аудита:

* **Аудит входа в систему**: Успех, Отказ.
  + - **Назначение**: соответствует подкатегории {0CCE9215-69AE-11D9-BED3-505054503030}, фиксируя событие 4624. Обеспечивает мониторинг всех попыток входа, включая неуспешные (событие 4625).
* **Аудит использования привилегий**: Успех, Отказ.
  + - **Назначение**: соответствует подкатегориям {0CCE9228-69AE-11D9-BED3-505054503030} и {0CCE9229-69AE-11D9-BED3-505054503030}, фиксируя событие 4672. Отслеживает назначение привилегий, включая действия, связанные с задачами и службами.
* **Аудит управления учетными записями**: Успех, Отказ.
  + - **Назначение**: фиксирует изменения учетных записей (например, создание, удаление, изменение паролей), что может быть связано с действиями, создающими задачи или службы. Хотя напрямую не требуется для событий задания, включение повышает общую безопасность мониторинга.

### Разница между настройкой через auditpol и локальные политики

Auditpol - Командная утилита для точечной настройки подкатегорий аудита. Позволяет включать или отключать конкретные подкатегории (например, {0CCE921F-69AE-11D9-BED3-505054503030} для задач) с детализацией успеха и сбоя.

**Преимущества**:

* Высокая точность: можно настроить только необходимые подкатегории, минимизируя объем логов, что нам на руку.
* Гибкость: поддерживает автоматизацию через скрипты (например, BAT-файл).
* Доступ к расширенным подкатегориям, не всегда доступным в secpol.msc.

**Недостатки**:

* Требует знания GUID подкатегорий и команд особенно для русской системы для корректного выполнения.
* Менее интуитивно для пользователей, предпочитающих графический интерфейс, но нам это не страшно.

**Использование в проекте**: Основной метод для настройки всех подкатегорий, так как позволяет точно выбрать события (6005, 4624, 4672, 4698, 4697).

**Локальные политики безопасности (**secpol.msc**)** - Графический интерфейс для настройки категорий аудита через "Локальная политика безопасности" → "Локальные политики" → "Политика аудита". Работает с более общими категориями (например, "Вход в систему"), которые включают несколько подкатегорий.

**Преимущества**:

* Простота: интуитивный интерфейс, не требующий знания команд или GUID.
* Удобство для базовой настройки, особенно для категорий, таких как вход или привилегии.

**Недостатки**:

* Меньшая детализация: настройка категорий (например, "Использование привилегий") включает все связанные подкатегории, что приводит к излишним логам.
* Ограниченный доступ к некоторым подкатегориям (например, {0CCE921F-69AE-11D9-BED3-505054503030} для задач не настраивается напрямую).

**Использование в проекте**: дополняет auditpol, обеспечивая настройку общих категорий ("Вход в систему", "Использование привилегий", "Управление учетными записями") для повышения надежности мониторинга.

**Комбинированный подход**:

* В проекте использованы оба метода для обеспечения надежности. auditpol применен для точной настройки подкатегорий, критичных для событий задания. secpol.msc использован для включения общих категорий, что упрощает начальную настройку и добавляет мониторинг учетных записей.
* Различия в подходах учтены: настройки secpol.msc могут перекрывать подкатегории, заданные через auditpol, поэтому проверка через auditpol /get /category:\* была проделана.

### 3.4 Настройка Sysmon

**Sysmon64** установлен для мониторинга процессов (Sysmon Event ID 1) с минимальной конфигурацией, отслеживающей запуск notepad.exe. Это снижает шум в журнале Microsoft-Windows-Sysmon/Operational. Конфигурация включает правило <ProcessCreate onmatch="include" />. Для реального мониторинга рекомендованы расширенные конфигурации (например, SwiftOnSecurity), исключающие легитимные процессы (svchost.exe) и отслеживающие подозрительные (powershell.exe, cmd.exe). Sysmon64 выбран для совместимости с 64-битной системой.

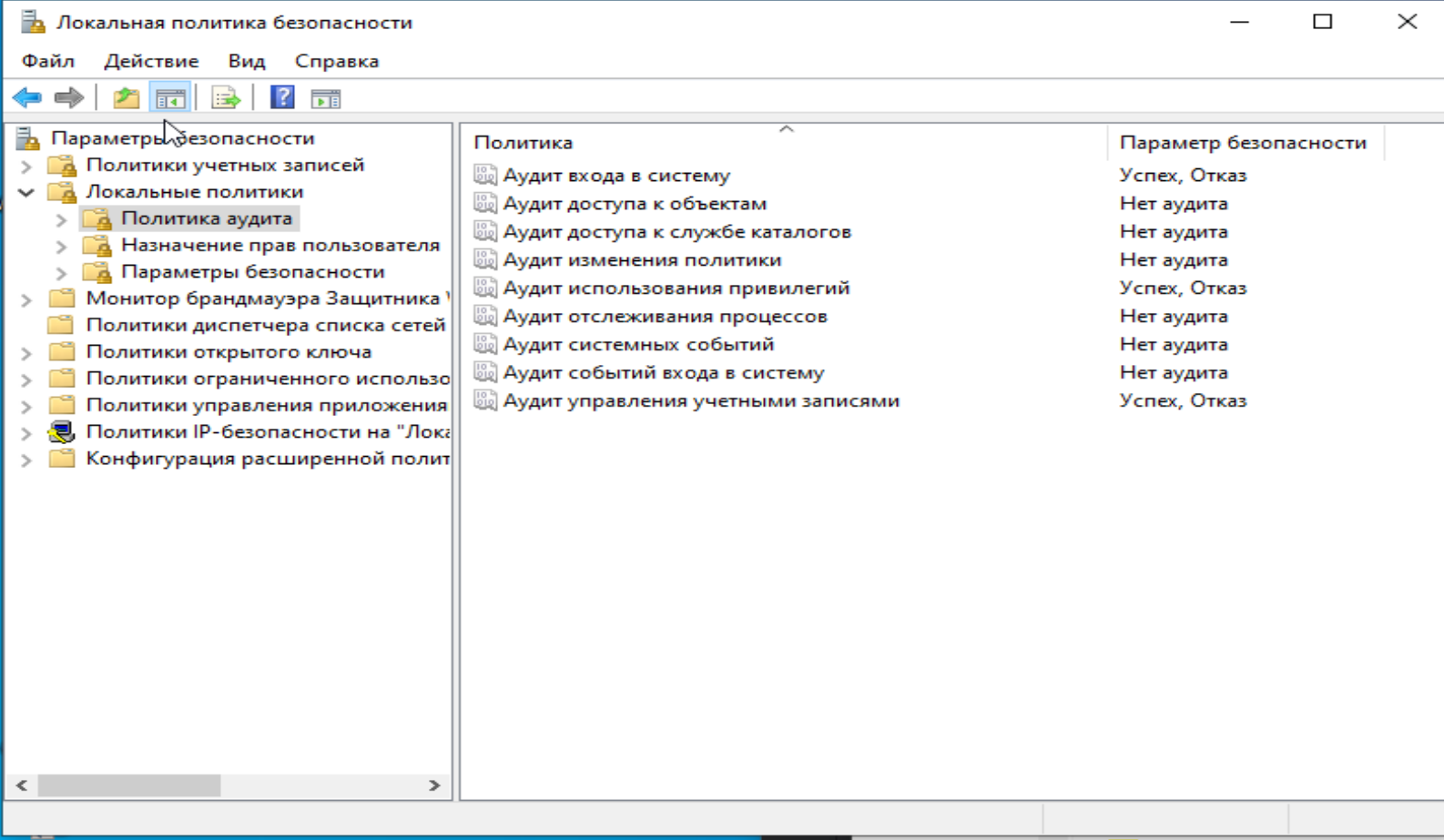


Рисунок 1 - настройки аудита в secpol.msc

## 4. Выбор технологий и подходов

Проект реализован на Python из-за удобных библиотек для работы с Windows и Telegram. Альтернативы, такие как PowerShell или BAT, не использовались, так как это излишне для нашей системы, а также из-за сложности обработки журналов и интеграции с API. Редактор реестра не применялся, так как настройки аудита выполнялись через auditpol и secpol.msc. XPath использовался только для парсинга Sysmon-событий из-за их XML-формата, а для остальных журналов — прямой доступ через pywin32.

Этот подход обеспечил простоту разработки и гибкость, хотя потребовал точной настройки аудитов для минимизации шума.

## 5. Архитектура и реализация

### 5.1 Алгоритм работы

Бот работает по следующему алгоритму:

1. Открывает журналы Windows (System, Security, Microsoft-Windows-Sysmon/Operational) через pywin32.
2. Ищет события за последнюю минуту (6005, 4624, 4672, 4698, 4697, 7045, Sysmon ID 1).
3. Обрабатывает данные: извлекает параметры (пользователь, время, хеши), исключает дубли с помощью логов (last\_\*.log, sysmon\_seen.log).
4. Для Sysmon ID 1 парсит XML через xmltodict, извлекает SHA256 и проверяет через VirusTotal API.
5. Отправляет уведомления в Telegram через python-telegram-bot.
6. Генерирует PDF-отчеты при включении ПК в новом дне, используя reportlab.

**Логирование(комбинированное)**:

* **Одиночное логирование** (last\_\*.log, sysmon\_seen.log): хранит время последнего события или ProcessGuid для исключения дублирования.
* **Накопительное логирование** (events\_\*.json): сохраняет все события в JSON для PDF-отчетов, очищается после отправки.

### 5.2 Структура проекта

Проект состоит из четырех файлов, реализованных модульно для четкого разделения функциональности.

* **config.py**: хранит конфигурационные переменные: TELEGRAM\_TOKEN (получен через BotFather), CHAT\_ID (определен через тестовое сообщение боту), VIRUSTOTAL\_API\_KEY (получен при регистрации на VirusTotal).
* **Работа**: обеспечивает доступ к API без изменения кода. Переменные импортируются в bot.py и event\_logger.py.
* **Технологии**: Python, без внешних зависимостей.
* **Фильтрация**: не выполняется, так как это файл конфигурации.
* **bot.py**:

**Функции**:

1. send\_message(message):

* **Работа**: асинхронно отправляет текстовое сообщение в Telegram-канал, указанный в CHAT\_ID. Использует python-telegram-bot для взаимодействия с Telegram API. Логирует успешную отправку или ошибки в консоль.
* **Технологии**: python-telegram-bot 22.0, библиотека telegram для асинхронных запросов.
* **Фильтрация**: проверяет валидность CHAT\_ID при инициализации бота. Игнорирует системные ошибки (например, временную недоступность Telegram), логируя их.

1. send\_document(file\_path):

* **Работа**: асинхронно отправляет PDF-файл (отчет) в Telegram. Открывает файл в бинарном режиме, передает его через bot.send\_document, логирует результат.
* **Технологии**: python-telegram-bot 22.0, стандартная библиотека Python для работы с файлами.
* **Фильтрация**: проверяет наличие файла и обрабатывает ошибки доступа, отправляя уведомление об ошибке через send\_message.
* **Особенности**: инициализирует объект Bot при запуске, проверяя TELEGRAM\_TOKEN и CHAT\_ID. Использует асинхронность для минимизации задержек.
* **event\_logger.py**: Основной модуль, отвечающий за чтение журналов Windows, обработку событий, логирование и генерацию отчетов. Использует pywin32 для доступа к журналам, xmltodict для парсинга Sysmon-событий, requests для VirusTotal, reportlab для PDF.

**Функции**:

* + 1. *read\_last\_event\_time(file\_name):*
* **Работа**: читает время последнего обработанного события из файла (last\_\*.log). Парсит строку в формате ISO, возвращает объект datetime или None, если файл пуст или содержит некорректные данные.
* **Технологии**: Стандартные библиотеки Python (os, re), datetime для парсинга времени.
* **Фильтрация**: проверяет формат строки через регулярное выражение, удаляет файл при ошибках, обрабатывает исключения (например, блокировку файла).
  + - write\_last\_event\_time(file\_name, event\_time, send\_message\_func):
* **Работа**: записывает время события в формате ISO в файл (last\_\*.log). Используется для отслеживания последнего обработанного события, чтобы избежать дублирования.
* **Технологии**: Стандартные библиотеки Python (os), асинхронная отправка ошибок через send\_message\_func.
* **Фильтрация**: обрабатывает ошибки записи, уведомляя через Telegram.
  + 1. *log\_event\_to\_json(file\_name, event\_data, send\_message\_func):*
* **Работа**: Добавляет событие в JSON-файл (events\_\*.json) для накопительного логирования. Читает текущий список событий, добавляет новое, записывает обратно. Используется для хранения данных для PDF-отчетов.
* **Технологии**: json для работы с JSON, os для файловых операций.
* **Фильтрация**: проверяет существование файла, создает пустой список, если файл новый. Обрабатывает ошибки записи, уведомляя через Telegram.
  + 1. *clear\_event\_logs(send\_message\_func):*
* **Работа**: Очищает все JSON-логи (events\_\*.json) после отправки PDF-отчета, записывая пустой список. Предотвращает накопление устаревших данных.
* **Технологии**: json, os.
* **Фильтрация**: обрабатывает ошибки доступа, уведомляя через Telegram.
  + 1. *generate\_pdf\_report(date, send\_message\_func, send\_document\_func):*
* **Работа**: Создает PDF-отчет за указанную дату, собирая события из JSON-логов (events\_\*.json). Фильтрует события по дате, формирует документ с заголовками ("Включение ПК", "Входы" и т.д.) и деталями (время, краткое описание). Использует шрифт DejaVuSans для кириллицы. Отправляет отчет через send\_document\_func и очищает логи через clear\_event\_logs.
* **Технологии**: reportlab 4.3.1 для PDF, json для чтения логов, datetime для фильтрации по дате.
* **Фильтрация**: выбирает события только за указанную дату, игнорирует пустые или некорректные JSON-файлы. Обрабатывает ошибки чтения/отправки, уведомляя через Telegram.
  + 1. *check\_system\_startup(send\_message\_func, send\_document\_func):*
* **Работа**: ищет событие 6005 (включение ПК) в журнале System за последнюю минуту через pywin32. Сравнивает время события с последним сохраненным (last\_startup.log). Если событие новое, отправляет уведомление через send\_message\_func. При смене дня (по сравнению с last\_startup.log) вызывает generate\_pdf\_report. Логирует событие в events\_6005.json.
* **Технологии**: pywin32 310 для чтения журнала, datetime для временных проверок, json для накопительного логирования.
* **Фильтрация**: ограничивает поиск минутным окном (TIME\_RANGE\_MINUTES), сравнивает с last\_startup.log для исключения дубликатов. Игнорирует события старше порога.
  + 1. *check\_security\_events(send\_message\_func):*
* **Работа**: обрабатывает события 4624 (входы), 4672 (привилегии), 4698 (задачи) в журнале Security за последнюю минуту. Извлекает данные через pywin32 (пользователь, домен, тип входа, привилегии, имя задачи). Фильтрует системные события (SID S-1-5-18, пользователь "СИСТЕМА"). Сравнивает с последними временами (last\_logon.log, last\_privilege.log, last\_task.log), отправляет новые события в Telegram и логирует в events\_4624.json, events\_4672.json, events\_4698.json.
* **Технологии**: pywin32 310, datetime, json.
* **Фильтрация**: ограничивает поиск минутным окном. Для 4624 фильтрует типы входа (2, 7, 10), для 4672 и 4698 исключает системные SID/пользователей. Проверяет время против логов для исключения дубликатов.
  + 1. *check\_service\_modification(send\_message\_func):*
* **Работа**: ищет события 4697 (новая служба) в журнале Security и 7045 (изменение службы) в журнале System за последнюю минуту. Извлекает параметры (имя службы, путь, тип запуска, пользователь) через pywin32. Фильтрует системные события ("СИСТЕМА"). Сравнивает с last\_service.log, отправляет новые события в Telegram и логирует в events\_service.json.
* **Технологии**: pywin32 310, datetime, json.
* **Фильтрация**: Минутное окно, исключение системных пользователей, проверка времени против last\_service.log. Для 4697 преобразует числовой тип запуска в читаемый формат.
  + 1. *check\_sysmon\_process(send\_message\_func):*
* **Работа**: ищет Sysmon Event ID 1 (создание процесса) в журнале Microsoft-Windows-Sysmon/Operational за последнюю минуту через pywin32. Парсит XML-события с помощью xmltodict, извлекает ProcessGuid, Image, SHA256, CommandLine. Проверяет ProcessGuid против sysmon\_seen.log для исключения дубликатов. Запрашивает VirusTotal по SHA256 через requests, кэшируя результаты в vt\_cache.json. Отправляет уведомление с деталями (процесс, аргументы, VirusTotal) и логирует в events\_sysmon.json.
* **Технологии**: pywin32 310, xmltodict 0.14.2, requests 2.32.3, json, datetime.
* **Фильтрация**: Минутное окно, проверка ProcessGuid в sysmon\_seen.log, очистка старых GUID (старше 24 часов). Игнорирует события без ProcessGuid или с некорректной датой.
* **main.py**: Асинхронно вызывает функции из event\_logger.py (check\_system\_startup, check\_security\_events, check\_service\_modification, check\_sysmon\_process) для проверки всех событий. Выводит начальное сообщение о проверке в консоль.
* **Технологии**: asyncio для асинхронности, os для работы с файлами.
* **Фильтрация**: не выполняет фильтрацию, делегирует её функциям event\_logger.py.
* **Особенности**: Точка входа программы. Проверяет наличие config.py и использует файл блокировки (lockfile.lock) для предотвращения множественных запусков. Создает и удаляет блокировку, обеспечивая однократное выполнение.

**Разработка**: Каждое событие тестировалось отдельно: код для одного события (например, 4698) писался с выводом в консоль, проверялась фильтрация дубликатов, затем код распределялся по модулям. Это упростило отладку и обеспечило стабильность.

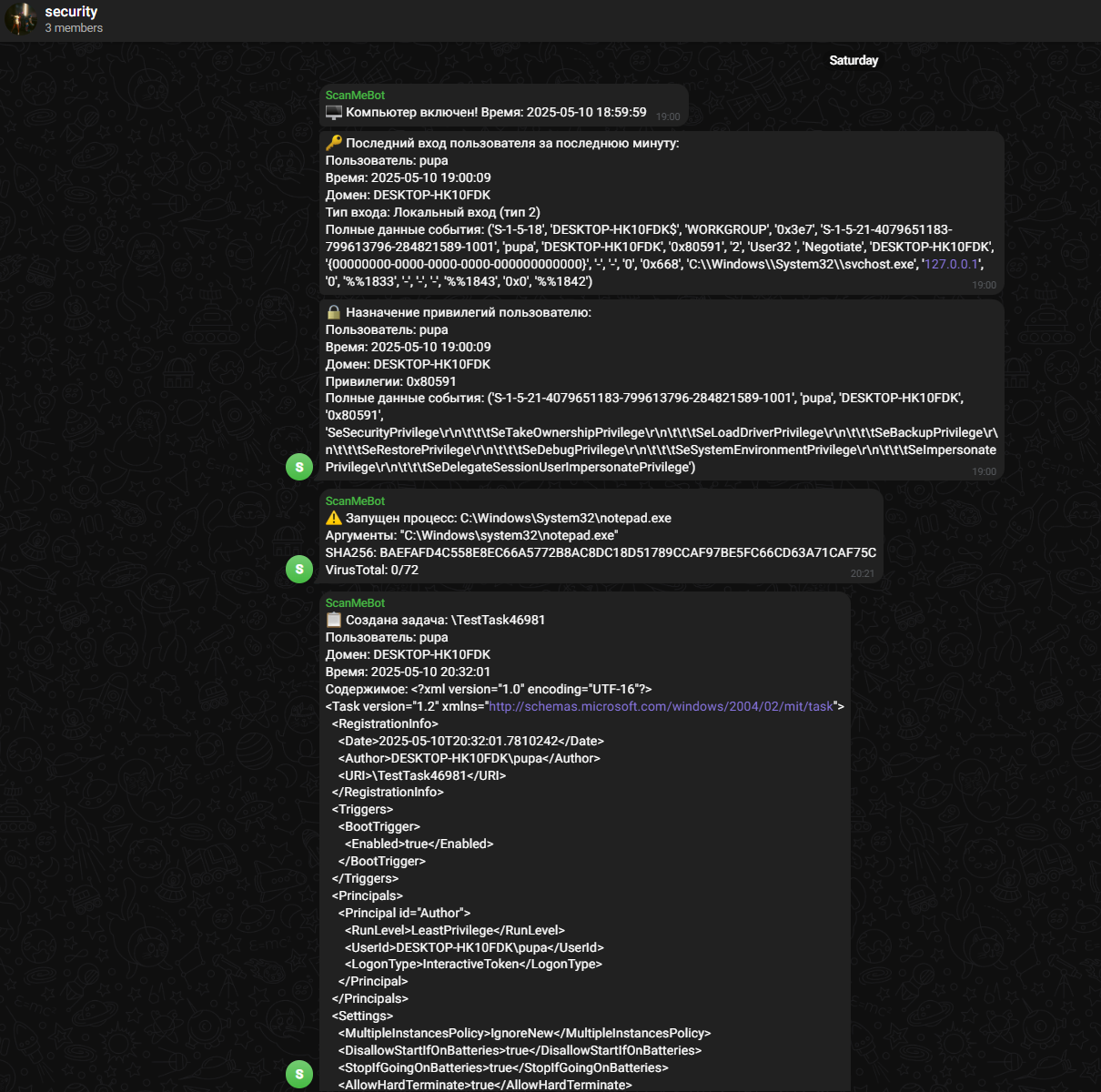


Рисунок 2 - Пример уведомлений в Telegram

## 6. Руководство по установке и эксплуатации

1. Установите Python 3.12.9 и библиотеки:

pip install python-telegram-bot==22.0 pywin32==310 requests==2.32.3 xmltodict==0.14.2 reportlab==4.3.1

1. Настройте config.py с API-ключами Telegram и VirusTotal.
2. Настройте аудиты через auditpol и secpol.msc (см. раздел "Настройка системного окружения").
3. Установите Sysmon64 с конфигурацией для notepad.exe или другой.
4. Создайте задачу в планировщике Windows:
   * Программа: pythonw.exe.
   * Аргумент: путь к main.py.
   * Запуск: при старте системы с задержкой 30 секунд, повтор каждую минуту.
   * Выбрать: выполнить с наивысшими правами.

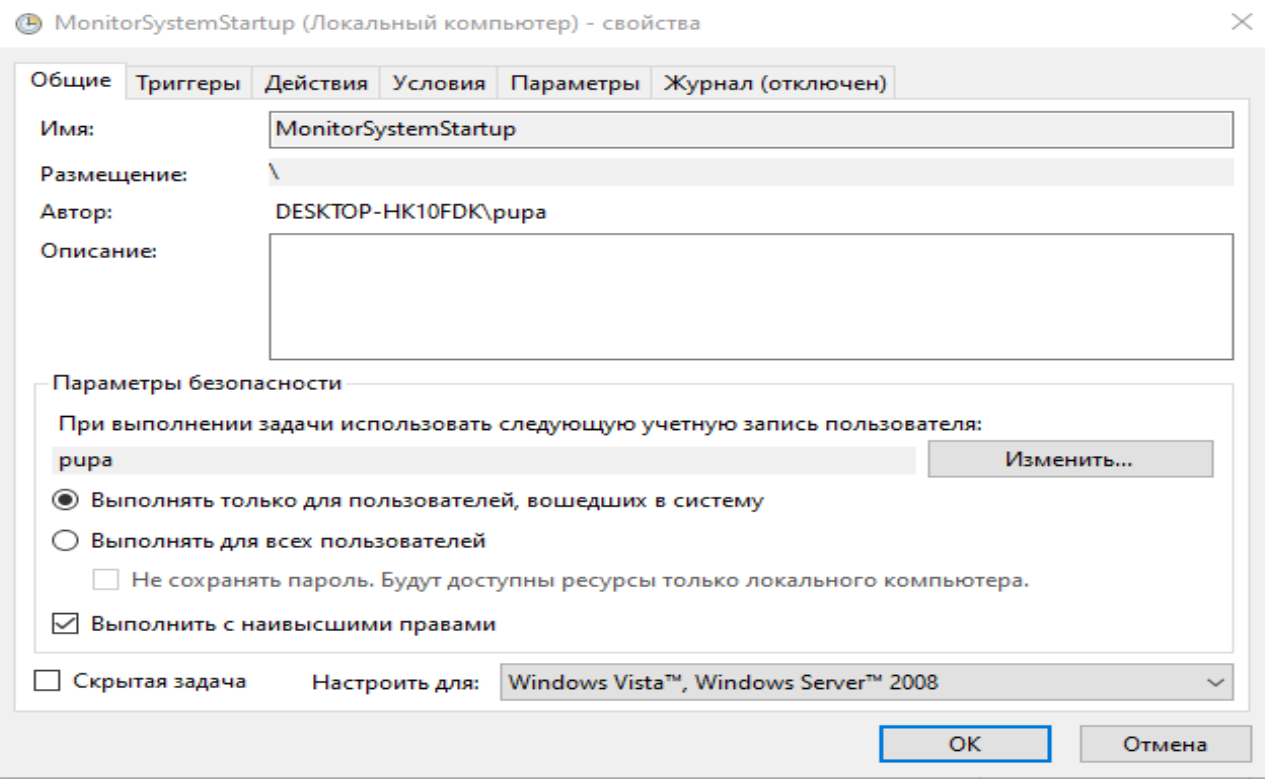


Рисунок 3 - Задача в планировщике задач

## 7. Ограничения и возможности оптимизации

* **Ограничения**:
  + Совместимость только с Windows 10 Pro (сборка 19045).
  + Лимит VirusTotal API (4 запроса/мин) замедляет проверку хешей.
  + Неявные зависимости между библиотеками и между событиями и аудитами.
  + Минимальная конфигурация Sysmon (только notepad.exe) ограничивает мониторинг.
* **Возможности оптимизации**:
  + Кэширование запросов VirusTotal для снижения нагрузки.
  + Использование расширенных конфигураций Sysmon (например, SwiftOnSecurity).
  + Асинхронная обработка событий для повышения производительности.
  + Адаптация для Windows 11 или других версий.

## 8. Приобретенные знания и навыки

Работа над проектом была увлекательным исследованием журналов Windows. Я научился:

* Настраивать аудиты через auditpol и secpol.msc, понимая, какие события можно отслеживать (входы, задачи, службы).
* Читать журналы Windows (System, Security, Sysmon) с помощью pywin32, раскрывая, как система логирует действия.
* Настраивать Sysmon64 для мониторинга процессов, изучая, как фильтровать шум и выделять важные события.
* Интегрировать Telegram и VirusTotal API для уведомлений и анализа безопасности.
* Писать модульный код, тестируя каждую функцию отдельно, что упростило отладку.

Проект показал, как глубоко можно заглянуть в систему через журналы, и вдохновил изучать безопасность дальше.

## Заключение

Бот полностью соответствует ТЗ: отслеживает события 6005, 4624, 4672, 4698, 4697, 7045 и Sysmon ID 1, отправляет уведомления в Telegram с деталями (пользователь, время, хеши) и генерирует PDF-отчеты при включении ПК в новом дне. Работает стабильно в Windows 10 Pro с настроенными аудитами и Sysmon64. Возможны доработки для универсальности (поддержка других версий Windows, оптимизация запросов VirusTotal), но текущая реализация выполняет задачу штатно.

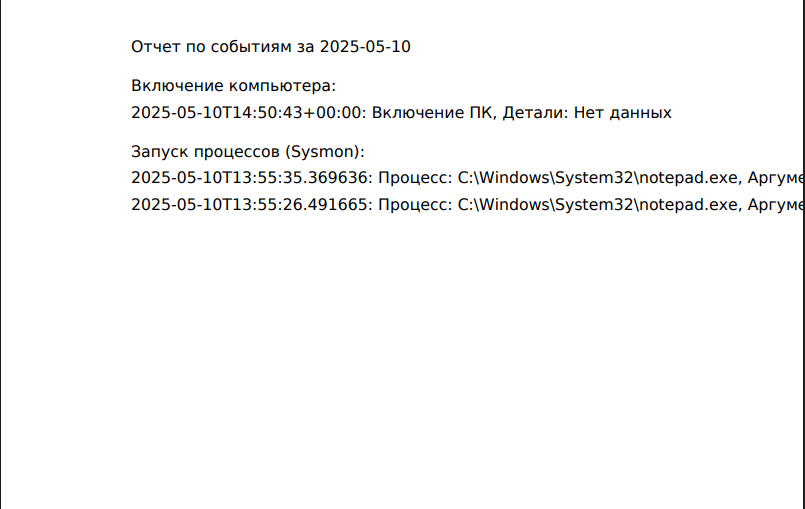


Рисунок 4 - Пример PDF-отчета