УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Защита многоагентных систем от цепных атак, несанкционированного выполнения инструментов и потери контекста

```
#INCLUDE <IOSTREAM>
INT MAIN() {
  STD::COUT << "ИНФОРМАЦИОННАЯ
БЕЗОПАСНОСТЬ";
}
```

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Цели работы:

Теоретическая:

Понять архитектурные уязвимости многоагентных систем (prompt injection в цепочках, несанкционированный вызов tools, эскалация привилегий).

Практическая:

Реализовать механизмы изоляции агентов, валидации запросов на выполнение инструментов и аудита всей цепочки принятия решений.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Что такое Retrieval-Augmented Generation?

Многоагентные архитектуры — это подход в ИИ, где несколько специализированных агентов (модели или процессы) работают вместе над задачей. Каждый агент может иметь свою роль, память и набор инструментов.

Основные принципы:

- Декомпозиция задачи
  (разбиваем сложной задачи на подзадачи)
- Специализация агентов
  (один пишет код, другой проверяет ошибки, третий ищет информацию в базе и т.д.)
- Координация
  (система (или "менеджер"-агент) управляет диалогом и распределяет работу)
- Итерации
  (агенты обмениваются сообщениями, уточняют шаги и приходят к решению)

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Угрозы для многоагентных архитектур:

- **Цепная промпт-инъекция:**
  Атака на первый агент с целью заставить его дать вредоносную инструкцию второму агенту

- **Злоупотребление инструментами (Tool Abuse):**
  Агент, имеющий доступ к инструменту «отправить email», может быть обманным путем направлен на спам-рассылку

- **Утечка контекста:**
  Агент может «забыть» свои системные инструкции под влиянием контекста от другого агента

Задача: **«Разработать и внедрить систему контроля и аудита для многоагентного приложения»**

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Этап 1: Архитектура с централизованным диспетчером (Orchestrator)

Задание: Изменить архитектуру приложения так, чтобы агенты не общались напрямую, а через единый центральный компонент — Orchestrator.

Цель: Централизовать управление, изолировать агентов, контролировать поток данных.

Orchestrator:

Получает задачу от пользователя → решает, какому агенту ее передать, основываясь на роли агента и политиках → получает ответ от агента, проверяет его и решает, передать ли следующему агенту или вернуть пользователю

Схема Централизованной архитектуры:

[Telegram Bot] — только UI, не знает логики
↓
[Orchestrator] — мозг системы: решает, какие агенты вызывать и в каком порядке
↓
[Модератор] → [RAG] → [LLM Gateway] — агенты вызываются только через оркестратор
↓
[Audit Service] — логирует все действия

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Этап 1: Архитектура с централизованным диспетчером (Orchestrator)

Как реализовано в проекте:

1) Telegram Bot — тонкий клиент, только принимает/отправляет сообщения
2) Orchestrator — единственный компонент, который знает всю цепочку:
Сначала модерация
Потом RAG
Потом генерация
3) Агенты изолированы — не могут вызывать друг друга напрямую
4) Нет прямого доступа к LLM — только через оркестратор.
Это предотвращает обход безопасности — пользователь или агент не может "перепрыгнуть" модерацию или RAG.

Преимущества:

Контроль потока данных — все запросы проходят через один "ворота"
Изоляция агентов — модератор не знает про RAG, RAG не знает про LLM
Лёгкое добавление новых шагов — например, "проверка на PII" перед генерацией

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Этап 2: Реализация политик доступа к инструментам (Tool Policies)

Задание: Для каждого агента и каждого его инструмента определить политику безопасности.

[Пользовательский запрос]
↓
[Orchestrator] → Проверяет политики:
- Модератор: может только классифицировать (не генерировать, не искать)
- RAG: может только искать (не вызывать API, не генерировать)
- LLM Gateway: может только генерировать (не читать файлы, не писать в БД)
- Audit: может только писать логи (не читать, не изменять)

Как реализовано в проекте:

- Модератор — может только вызывать YandexGPT с системным промптом "ответь ДА/НЕТ". Не может генерировать текст, искать документы.
- RAG — может только искать в FAISS. Не может вызывать внешние API, не может генерировать текст.
- LLM Gateway — может только генерировать ответ. Не может читать файлы, не может писать в БД.
- Audit — может только писать в файл. Не может читать запросы, не может изменять логи.

Это ролевая модель доступа — каждый агент имеет минимально необходимые права.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Этап 2: Реализация политик доступа к инструментам (Tool Policies)

Задание: Для каждого агента и каждого его инструмента определить политику безопасности.

Дополнительные политики:

- Ограничение входных данных — например, RAG принимает только query: str, не может получить доступ к системному промпту.

- Валидация параметров — например, maxTokens в LLM Gateway ограничен 1000 или проверить, что email отправляется только на корпоративные домены.

- Запрет на выполнение кода — нигде в проекте нет exec, eval, subprocess. Это предотвращает инъекции и побочные эффекты.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Этап 3: Внедрение сквозного аудита (Audit Logging)

Задание: Реализовать логирование всех действий в системе для последующего анализа инцидентов

Цель: Записывать всё — кто, что, когда, почему — чтобы можно было восстановить цепочку решений

Схема Сквозного аудита:

[Пользователь: "Как сбросить пароль?"]
↓
[Orchestrator] → [Audit: REQUEST_RECEIVED]
↓
[Модератор] → [Audit: MODERATION_ALLOWED]
↓
[RAG] → [Audit: RETRIEVED_CHUNKS (score=0.92)]
↓
[LLM Gateway] → [Audit: MODEL_USED=YandexGPT, TOKENS=127]
↓
[Orchestrator] → [Audit: RESPONSE_SENT]
↓
[Telegram Bot] → Ответ пользователю

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Этап 3: Внедрение сквозного аудита (Audit Logging)

Как реализовано в проекте:

1) Уникальный trace_id — генерируется при входе запроса, прокидывается во все сервисы.
2) Аудит каждого шага:

- Запрос получен
- Модерация: разрешено/заблокировано
- RAG: найдено N чанков
- LLM: использована модель, количество токенов
- Ответ отправлен

3) Структурированные логи — JSON с timestamp, trace_id, event, details.
4) Централизованное хранение — все логи пишутся в один файл (можно расширить до Elasticsearch, Loki).
Это позволяет восстановить полную цепочку решений для любого запроса.

Преимущества:

- Прозрачность — можно понять, почему был дан тот или иной ответ.
- Отслеживание аномалий — например, "почему RAG вернул пустой контекст?".
- Соответствие compliance — например, GDPR, ФЗ-152 — можно доказать, что запрос был обработан безопасно.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Итоговая схема безопасности:

[Пользователь]
↓
[Telegram Bot] — UI, генерирует trace_id
↓
[Orchestrator] — центр управления, применяет политики
├── [Модератор] — политика: только классификация
├── [RAG] — политика: только поиск
├── [LLM Gateway] — политика: только генерация
└── [Audit] — политика: только запись логов
↓
[Ответ пользователю]

Это защищённая, аудируемая, изолированная архитектура LLM-агентов.

Проект соответствует best practices для безопасных LLM-систем:

- Централизованное управление
- Минимальные привилегии
- Сквозной аудит
- Изоляция компонентов