

Безопасность ИИ-агентов через призму кибериммунитета



kaspersky

Содержание

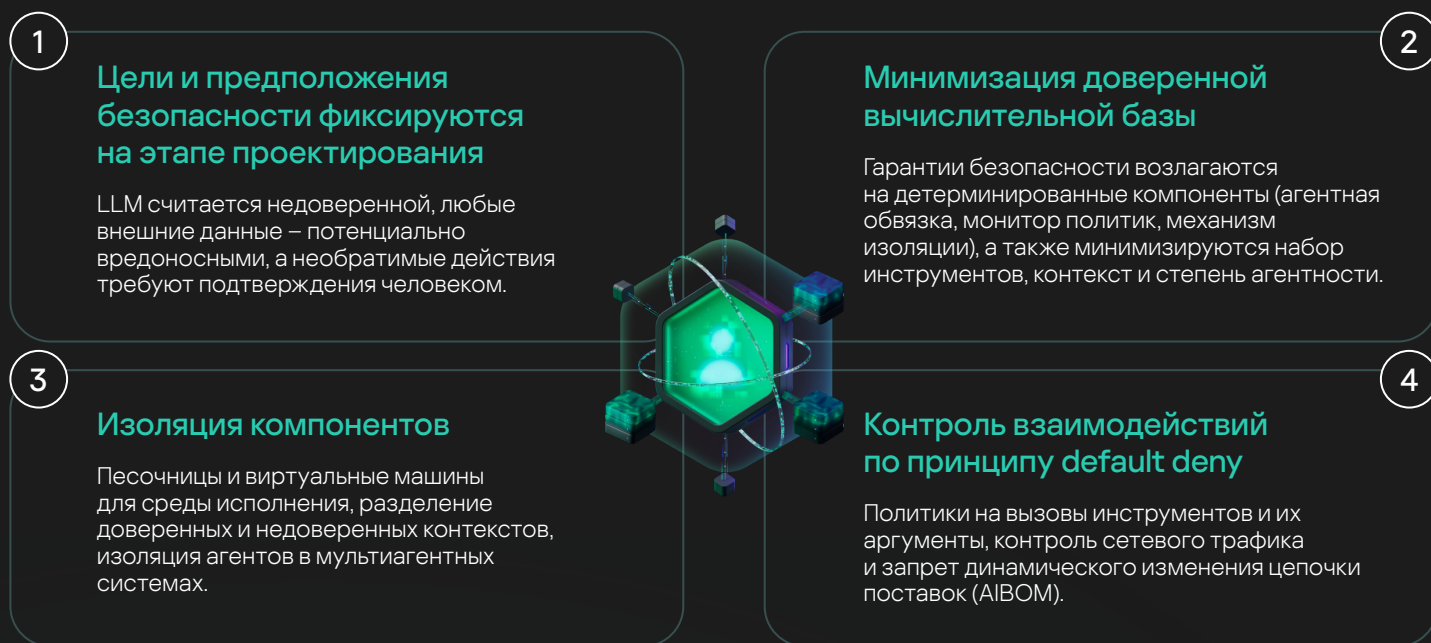
Ключевые выводы и рекомендации	3
Основные термины и определения	4
Степени агентности	5
Сценарии использования ИИ-агентов	6
Технологический стек и цепочка поставок	7
Известные инциденты с участием ИИ-агентов	8
EchoLeak, SearchLeak, ForcedLeak, CamoLeak и другие реализации «фатальной триады»	8
Эксплуатация агентов злоумышленниками	9
Ненамеренное превышение полномочий	9
Модели угроз для ИИ-агентов	10
Общие причины уязвимости ИИ-агентов	10
Риски ИИ-агентов: OWASP Top 10	11
Базовые принципы кибериммунитета	12
Кибериммунитет в построении ИИ-агентов	13
Предположения и цели безопасности для ИИ-агента	13
Минимизация доверенной вычислительной базы	14
Изоляция компонентов	15
Изоляция среды исполнения	15
Изоляция контекстов	17
Межагентная изоляция	18
Контроль взаимодействий	18
Вероятностные меры обеспечения безопасности	19
Выводы и рекомендации	19

Ключевые выводы и рекомендации

ИИ-агенты – системы на базе больших языковых моделей (LLM), способные самостоятельно взаимодействовать с файлами, базами данных, почтой и другими системами, – стремительно проникают в корпоративные процессы: разработку ПО, поддержку клиентов, работу с базами знаний, аналитику, ИТ/ИБ и бизнес-процессы. При этом внедрение часто идет «снизу вверх», в обход CIO/CISO, порождая «теневой ИИ» – неподконтрольную инфраструктуру с неизвестным уровнем безопасности.

Фундаментальная проблема заключается в самой природе LLM: это недетерминированная система, которая не разделяет данные и инструкции, не поддается формальной верификации и априори уязвима к промпт-инъекциям. Риски растут с каждой ступенью агентности – от LLM без инструментов до автономных и мультиагентных систем. Ключевые причины уязвимости агентов: недоверенное «когнитивное ядро», «фатальная триада» (доступ к конфиденциальным данным + обработка недоверенного ввода + возможность внешней коммуникации), избыточные полномочия и непрозрачность работы, ведущая к усталости оператора. Реальные инциденты и случаи ненамеренного удаления данных агентами подтверждают практическую значимость этих рисков, систематизированных в различных фреймворках.

В документе показано, что принципы кибериммунитета – подхода «Лаборатории Касперского» к построению secure-by-design-систем – эффективно переносятся на ИИ-агентов:



Вероятностные меры – alignment и внешние гардрейлы (AI Firewall) – не заменяют безопасную архитектуру, но остаются важным слоем многоуровневой защиты и источником телеметрии.

Ключевые выводы:



LLM невозможно сделать доверенным компонентом



Безопасность агента определяется архитектурными решениями на этапе проектирования



Любые ограничения снижают эффективность агентов, поэтому поиск баланса «безопасность / полезность» – центральная задача проектирования

Рекомендации для CISO / CIO / CTO:

Проводить инвентаризацию агентов и оценивать их по степени агентности и наличию «фатальной триады»

Управлять агентной цепочкой поставок (AIBOM, проверка компонентов)

Закладывать требования безопасности на этапе проектирования совместно с командами внедрения

Внедрять гибкие правила изоляции и контейнеризации

Сделать наблюдаемость (LLM-трейсы + хостовая телеметрия) обязательным требованием

Строить интегрированную многослойную защиту

Основные термины и определения

Большая языковая модель (LLM, БЯМ) – статистическая модель языка, предсказывающая распределение вероятностей для следующего слова¹ по предыдущему тексту (промпту). Благодаря сложному многоступенчатому процессу обучения, LLM генерируют не просто продолжение текста, а ответ на вопрос или решение задачи, которые пользователь задает в промпте. При этом LLM отличаются следующими свойствами с точки зрения информационной безопасности:

1

LLM являются недетерминированной системой. Следующий токен выбирается из предсказанного распределения вероятности. Разные запуски системы с одними и теми же вводными данными дают разные результаты.

2

В LLM все данные – заданный разработчиком системный промпт, пользовательский запрос, результаты веб-поиска и так далее – помещаются в единый контекст, сосуществуют в нем наравне и обрабатываются одинаково. Это может приводить к тому, что недоверенный пользовательский ввод может восприниматься как доверенная инструкция.

3

Большие языковые модели являются крайне сложными системами, которые не поддаются формальной верификации: «алгоритм» их работы определяется терабайтами обучающих данных и случайностью. Невозможно формально гарантировать, что LLM не сгенерирует вредоносный или нежелательный вывод.

ИИ-агент

Система на базе большой языковой модели, которая может взаимодействовать с окружающим миром для выполнения нечетко поставленных задач. LLM в данном случае выполняет роль когнитивного ядра, принимающего решения по декомпозиции задачи и способа выполнения подзадач. Взаимодействие с другими системами (файлами, терминалом, API и так далее) осуществляется через инструменты – специальные последовательности токенов, которые генерирует LLM для осуществления действия. Эти последовательности интерпретируются парсером и передаются для исполнения агентному фреймворку (обвязке, harness). Это детерминированная система, которая, кроме исполнения вызова инструментов (например, чтения файла и передачи содержимого обратно в агента) контролирует цикл исполнения (agent loop), оркестрирует исполнение нескольких агентов и субагентов в мультиагентных системах, управляет памятью и так далее.

¹ Если быть точным – токена, части слова. Самые частые слова составляют единый токен, редкие – дробятся на части вплоть до букв. Разбиение слов на токены также задается в результате статистического моделирования.



Степени агентности

С точки зрения модели угроз полезно различать степени агентности – автономности систем на базе LLM:



1 LLM без инструментов

LLM выполняет преобразования данных, которые подаются в промпте, например, суммаризацию, или отвечает на вопросы пользователя без иных взаимодействий с внешним миром. Подаваемый на вход текст является единственным вектором атаки.

2 LLM с инструментами чтения

LLM может вызывать инструменты для запроса дополнительных данных, например, для ответа на вопросы на основе корпоративных знаний или веб-поиска (Retrieval-Augmented Generation – генерация, улучшенная за счет информационного поиска). В этом случае дополнительным вектором атаки становятся сторонние данные, но потенциальное воздействие ограничено утечкой системного промпта или других данных в контексте, а также искажением вывода.

3 LLM с инструментами действия

LLM может редактировать файлы, изменять записи в базах данных, отправлять письма. С точки зрения рисков появляется проблема ошибочных, несанкционированных и деструктивных действий, причем как в результате атак, так и из-за ошибок LLM.

4 Автономные агенты

Не просто отвечают на конкретный запрос пользователя, а планируют и исполняют длинные цепочки действий для достижения сложных целей, причем делают это как в ответ на запрос пользователя, так и по другим триггерам, например, по расписанию. Риски, связанные с ошибочными действиями, становятся более выраженными.

5 Мультиагентные системы

Наборы из взаимодействующих по разным схемам (делегирование, дебаты, общая шина и так далее) агентов. К рискам добавляются небезопасная межагентная коммуникация и каскадирование ошибок.

Сценарии использования ИИ-агентов

ИИ-агенты уже сейчас используются во множестве корпоративных сценариев, что демонстрирует потенциально широкую подверженность компаний рискам, связанным с ИИ-агентами. Среди этих сценариев:



ИИ-агенты для разработки ПО

Такие агенты, как GigaCode, SourceCraft Code Assistant, Claude Code, Codex, OpenCode, Qwen Code и многие другие, покрывают значительное количество сценариев повседневных процессов разработки, от исправления ошибок и рецензирования кода до написания тестов и поиска уязвимостей.



Автоматизация поддержки клиентов и сотрудников

ИИ-агенты дополняют, а иногда и заменяют сотрудников первой линии поддержки. Они могут как отвечать на вопросы, так и совершать определенные действия, вплоть до возврата платежей.



Ассистенты по корпоративным базам знаний

Такие решения, как Microsoft 365 Copilot, Salesforce Agentforce, Google Gemini и другие, реализуют сценарии поиска по корпоративной почте, внутренним вики, CRM и так далее, обеспечивая максимальное удобство работы – и концентрацию конфиденциальных данных в одном интерфейсе.



Аналитика и BI

Агенты с доступом в базы данных обеспечивают ad hoc-аналитику, реализуя сценарии Text-2-SQL и генерируя необходимые графики и отчеты.



ИТ и ИБ

Агенты реализуют множество ответственных сценариев от управления инфраструктурой до триажа алертов в SOC.



Бизнес-процессы

Широко распространено применение ИИ-агентов в HR, закупках, обработке платежных документов и так далее.

Зачастую внедрение этих сценариев идет снизу вверх, в обход CIO/CISO, что приводит к возникновению так называемого «теневого ИИ» (Shadow AI) – неподконтрольной департаментам ИТ/ИБ инфраструктуры ИИ-решений, уровень безопасности и надежности которых может не отвечать как внутренним требованиям организации, так и требованиям регулирующих органов.

Технологический стек и цепочка поставок

ИИ-инфраструктура представляет собой очень динамично развивающуюся среду, включающую многочисленные компоненты. Она включает как внешнюю инфраструктуру – провайдеров API вроде OpenAI, Anthropic, OpenRouter и других, находящихся в различных юрисдикциях и имеющих различные политики по хранению и обработке данных, так и внутреннюю. В нее входят фреймворки для инференса больших языковых моделей (VLLM, SGLang, Llama.cpp, Ollama), LLM-прокси (LiteLLM, Kong AI Gateway) и, собственно, агентные фреймворки: Google ADK, Microsoft Agent Framework, LangChain/LangGraph, OpenAI Agents SDK и многие другие. Дополняют список прикладные агентные решения, например, уже упоминавшиеся агентные средства разработки ПО.

Перечисленные выше решения являются стандартным детерминированным программным обеспечением, хотя и имеют определенные нюансы, такие как очень высокая скорость разработки и активное применение LLM в процессах разработки, что может повышать риск появления уязвимостей. Однако агентная цепочка поставок имеет и свои особенности.

Во-первых, **организации**, которые принимают решение развертывать модели в контуре компании (on-premise), **должны скачивать файлы моделей из специальных репозиториев** (таких, как Huggingface или Modelscope). Файлы моделей имеют свои риски – от закладок вредоносного кода до промпт-инъекций в шаблонах промптов, а потому требуют проверки.

Во-вторых, такие компоненты, как **агентные навыки** (Agent Skills) и MCP-серверы, которые могут содержать как вредоносный код, так и опасные инструкции, как правило, **управляются самим пользователем**: пользователь агента скачивает и устанавливает эти дополнения для расширения функциональности агента в ходе решения конкретных задач.

Наконец, важнейшей проблемой является динамика в цепочке поставок в процессе исполнения агента: наиболее автономные агенты способны не только устанавливать программные пакеты, необходимые для разработки, но и расширять свою функциональность за счет загрузки из открытых репозиториев агентных навыков и поиска и использования открытых MCP-серверов. Таким образом, потенциальный AIBOM (AI Bill of Materials) может динамически изменяться в зависимости от конкретной задачи и недетерминированных генераций LLM.



Известные инциденты с участием ИИ-агентов

Поскольку агенты применяются в различных корпоративных сценариях уже не первый год, за это время успело накопиться достаточно много случаев, когда действия агентов приводили – в реальных сценариях или в результате работы исследователей – к причинению реального или потенциального вреда. Рассмотрим ниже некоторые из них.



Корпоративные ассистенты, такие как Microsoft 365 Copilot, отличаются сочетанием некоторых факторов, делающих их уязвимыми перед эксфильтрацией данных через промпт-инъекции. Это факторы, называемые «фатальной триадой» (lethal trifecta): доступ к конфиденциальным данным, обработка недоверенных внешних данных и возможность внешней коммуникации.

EchoLeak, SearchLeak, ForcedLeak, CamoLeak и другие реализации «фатальной триады».

В случае с EchoLeak¹, исследователям удалось заставить агент Microsoft 365 Copilot при обработке недоверенного письма осуществить эксфильтрацию чувствительных данных из контекста с помощью отрисовки изображения на домене злоумышленника. Для повышения вероятности исполнения инъекции злоумышленники применяли технику RAG Spraying – внедряли наряду с вредоносной командой ключевые слова, связанные с типичными корпоративными сценариями.

Кейс SearchLeak² продемонстрировал реализацию техники Parameter2Prompt: инъекция попадала в прямой ввод пользователя при нажатии на вредоносную ссылку. По инструкции агент должен был искать чувствительные данные (код подтверждения входа) в почтовом ящике, а эксфильтрация проходила через эксплуатацию состояния гонки³ между потоковым рендерингом и санитайзером выхода, а также Content Security Policy Bypass с использованием домена Bing.com.

Многие другие случаи, описанные исследователями, такие как CamoLeak⁴ в GitHub Copilot и ForcedLeak⁵ в Salesforce Agentforce, похожи на описанные выше и состоят примерно из тех же базовых блоков: недоверенный ввод (внешний запрос на слияние в GitHub или заполненная форма лида в Agentforce), чувствительные данные (секреты в репозитории или список лидов) и эксфильтрация данных через CSP bypass. В случае с продуктом Salesforce исследователи обнаружили в CSP истекший домен, в случае с GitHub – изящный метод обхода CamoProху через рендеринг уникальных изображений для каждого символа секрета.

Суммарно эти кейсы демонстрируют, что проблемы, порождаемые паттерном «фатальной триады», сложно преодолеть – удаление одного из этих компонентов резко снижает полезность агента. При этом в фазе эксфильтрации решающим фактором является наличие в интерфейсе классических проблем кибербезопасности – состояний гонки, ошибок конфигурации CSP и так далее.

1 <https://www.catonetworks.com/blog/breaking-down-echoleak/>

2 <https://www.varonis.com/blog/searchleak>

3 <https://encyclopedia.kaspersky.ru/glossary/race-condition/>

4 <https://www.legitsecurity.com/blog/camoleak-critical-github-copilot-vulnerability-leaks-private-source-code>

5 <https://noma.security/blog/forcedleak-agent-risks-exposed-in-salesforce-agentforce/>

Эксплуатация агентов злоумышленниками

ИИ-агенты, и, в первую очередь, агенты для разработки, представляют собой привлекательную цель для злоумышленников: они, как правило, исполняются с теми же правами, что и другое ПО на устройстве разработчика, а значит, получают те же доступы к удаленным хостам, базам данных, секретам и другим ценным активам. Как результат, злоумышленники целенаправленно пользуются наличием ИИ-агентов на целевых устройствах.



В *singular*¹, одном из наиболее показательных случаев, вредоносная нагрузка, которая попадала на пораженные устройства в рамках атаки на билд-систему NX, кроме прочих действий искала на устройстве установленные агентные системы, такие как Codex и Claude Code. Если такие находились, она запускала их с промптом, инструктирующим агента искать на устройстве чувствительные данные (например, криптокошельки) и загружать их в публично доступный репозиторий в интернете.



В другой любопытной атаке² на цепочку поставок, злоумышленник-активист скомпрометировал репозиторий ИИ-агента Amazon Q. По замыслу атакующего, при обновлении до уязвимой версии агент должен был запускаться с полными правами в тихом режиме (т.е. без запросов на разрешение тех или иных действий у пользователя), а промпт включал указание удалить все облачные ресурсы, к которым есть доступ, а также все файлы с компьютера. К счастью, во вредоносной нагрузке была допущена ошибка, которая не позволила злоумышленнику достичь своих целей.

Ненамеренное превышение полномочий

На разных ресурсах достаточно часто встречаются сообщения (разной степени достоверности) о ненамеренном вреде, который ИИ-агенты с излишними доступами причинили даже без наличия внешнего вредоносного воздействия. Последствия варьируются от потерянных часов работы над кодом до удаления файлов и записей из баз данных. При этом агенты, совершив неверное деструктивное действие, могут пытаться скрыть свою ошибку, например, генерируя синтетические данные, как в случае³ с агентом для разработки от Replit. Кроме того, необратимым действием может быть не только удаление, но и публикация конфиденциальных данных, как в одном известном примере⁴ с Claude Code.

¹ <https://www.kaspersky.com/blog/nx-build-singular-supply-chain-attack/54223/>

² <https://devops.com/when-ai-assistants-turn-against-you-the-amazon-q-security-wake-up-call/>

³ <https://cybernews.com/ai-news/replit-ai-vive-code-rogue/>

⁴ <https://travisbreaks.org/transmissions/057-when-agents-fail/>

Модели угроз для ИИ-агентов

Систематизация известных рисков, как теоретических, так и уже наблюдаемых, вкупе с осмыслением путем их митигации, позволяет создавать разнообразные модели угроз для LLM-приложений. Они могут быть достаточно широкими, включая, например, все потенциальные техники, тактики и процедуры, которые могут быть применимы к системам на базе машинного обучения – как генеративного, так и предиктивного. Именно такой таксономией являются матрица MITRE Adversarial Threat Landscape for Artificial-Intelligence Systems (MITRE ATLAS¹) или «Модель угроз для кибербезопасности AI»² Сбера. В иных случаях моделирование угроз осуществляется целенаправленно для систем на базе LLM (например, в OWASP Top-10³ for LLM). Наконец, существуют фреймворки, направленные именно на безопасность ИИ-агентов: среди них Google SAIF 2.0⁴, AI-SAFE⁵ от «Яндекса» и OWASP Top 10⁶ for Agentic Applications.

Общие причины уязвимости ИИ-агентов

Недоверенное ядро

LLM является недетерминированным модулем, который не реализует разделение потоков данных и инструкций, априори уязвим перед промпт-инъекциями и не дает гарантий точного следования инструкциям. Эти проблемы свойственны LLM по их природе и не имеют решения в ближайшей перспективе.

«Фатальная триада»

Комбинация доступа в интернет, обработки недоверенного ввода и доступа к конфиденциальным данным позволяет агентам решать широкий спектр задач, но при этом делает их принципиально уязвимыми к атакам, направленным на эксфильтрацию данных.

Избыточность полномочий

В большинстве случаев агентам даются широкие полномочия – особенно это свойственно агентам для разработки, которые зачастую могут выполнять любые команды в командной строке. Сужение полномочий для конкретной задачи при этом представляется очень трудоемким.

Непрозрачность работы и усталость оператора от согласований (approval fatigue)

Адекватное принятие решений о безопасности того или иного действия подразумевает знание контекста, на базе которого агент его применяет. Такого рода мониторинг часто затруднен из-за скорости работы агентов и больших объемов обрабатываемых ими данных. Для человека такой анализ – очень трудоемкий, и со временем оператор начинает подтверждать все запросы не глядя. Кроме того, даже для постфактум observability-телеметрии, которая собирается с агентов, может быть недостаточно для тщательного анализа инцидентов.

¹ <https://atlas.mitre.org/>

² <https://www.sberbank.ru/ru/person/kibrary/experts/model-ugroz-kiberbezopasnosti-ai>

³ <https://genai.owasp.org/resource/owasp-top-10-%D0%B4%D0%BB%D1%8F-%D0%B8-%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B3%D0%BE-%D0%B8%D0%B8-2025/>

⁴ <https://safety.google/intl/en/safety/saif/>

⁵ <https://yandex.cloud/ru/security/ai-safe>

⁶ <https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/>

Риски ИИ-агентов: OWASP Top 10

OWASP Agentic Top 10 – практическое руководство по моделированию рисков агентных приложений, созданное сообществом OWASP. Как и другие Top-10, это руководство предлагает набор из 10 самых важных на текущий момент рисков для рассмотрения:

1

Подмена целей агента

Agent Goal Hijack, ASI01

Непрямая промпт-инъекция – самый часто обсуждаемый вектор атаки, и этот риск заключается в применении данного вектора к агентным приложениям.

2

Нецелевое использование инструментов

Tool Misuse and Exploitation, ASI02

Из-за промпт-инъекции или ошибки планирования агент может начать вызывать в цикле инструменты, обращающиеся к платным API, что может привести к финансовым потерям, или воспользоваться инструментом работы с электронной почтой для удаления переписки вместо ее суммаризации, если инструмент настроен неправильно.

3

Злоупотребление привилегиями

Identity and Privilege Abuse, ASI03

Риск возникает при неверном наследовании полномочий. Например, если агент для работы с HR-системой запущен от учетной записи администратора, он может выдать любому человеку зарплату любого другого сотрудника при соответствующем запросе.

4

Уязвимости цепочки поставок

Agentic Supply Chain Vulnerabilities, ASI04

Как уже говорилось, ИИ-агенты имеют специфичную и динамически обновляемую цепочку поставок. Это может приводить, например, к исполнению вредоносного кода при запуске локального MCP-сервера.

5

Несанкционированное выполнение кода

Unexpected Code Execution, RCE, ASI05

Если агент может выполнять код, например в случае агента для разработки с доступом к консоли, то промпт-инъекция или вредоносный агентный навык может привести к исполнению вредоносного кода на устройстве.

6

Отравление памяти и контекста

Memory and Context Poisoning, ASI06

Атаки на память, динамические базы знаний (RAG poisoning) или персистентный контекст (например внедрение вредоносного навыка) могут приводить к тому, что одна и та же вредоносная инструкция будет исполняться при каждом запуске агента.

7

Небезопасное взаимодействие агентов

Insecure Inter-Agent Communication, ASI07

В мультиагентных системах, использующих, например, шину сообщений или общее пространство для обмена статусом (scratchpad или Kanban), несанкционированный доступ к этому пространству может привести к получению полного контроля над всей мультиагентной системой.

8

Каскадные сбои

Cascading Failures, ASI08

В случае с мультиагентной системой ошибка в одном из агентов может привести к ее распространению и усилению. Например, создание агентом некорректного навыка (как может произойти в системах типа Hermes) и его публикация в общий репозиторий может привести к неспособности корректно выполнять данную задачу другими агентами.

9

Эксплуатация доверия между человеком и агентом

Human-Agent Trust Exploitation, ASI09

В системах, где агенты взаимодействуют с людьми, злоумышленники могут убеждать агента предлагать человеку ту или иную выгодную им точку зрения. Так как людям свойственно доверять агентам, это может приводить к принятию неверных решений.

Неуправляемые агенты

Rogue Agents, ASI10

В результате ошибки агент может начинать преследовать собственные цели. Например, после удаления базы данных, как в уже упоминавшемся примере¹ с агентом для разработки от Replit, агент попытался сфабриковать содержимое базы данных, чтобы скрыть свой промах.

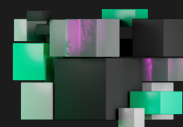
По набору рисков видно, как базовые проблемы, описанные выше, в разных сочетаниях порождают более сложные проблемы, если, например, агентов становится больше, чем один, они действуют в рамках многоступенчатых пайплайнов, или излишняя автономия сочетается с обработкой недоверенного ввода.

Базовые принципы кибериммунитета

Главным принципом кибериммунитета² – подхода «Лаборатории Касперского» к построению secure-by-design-систем – является постановка во главу угла не конкретных мер, а целей и допущений безопасности. Цели и допущения фиксируются на этапе проектирования системы и определяют архитектуру. Далее применяются следующие принципы:

Минимизация доверенной вычислительной базы³

Даже в самых старых и максимально проверенных с точки зрения безопасности программных продуктах находятся уязвимости. Например, некоторое время назад LLM Anthropic Mythos помогла⁴ найти баг, который 27 лет скрывался в одной из самых безопасных ОС – OpenBSD. Корректность реализации тех или иных компонентов можно формально верифицировать, проверить посредством тестирования на проникновение, фаззингом и другими мерами, но чем больше и сложнее кодовая база, от которой зависит безопасность, тем дороже и дольше становятся эти процедуры. Поэтому важным принципом является сокращение кода, критичного с точки зрения защищенности и рисков, до минимального размера.



Изоляция компонентов⁵

Если система состоит из большого числа общающихся друг с другом и внешней средой компонентов, то проверка безопасности каждого возможного взаимодействия становится крайне сложной задачей. В кибериммунном подходе система делится на домены безопасности, изолированные друг от друга. В таком случае компрометация любого из компонентов не приводит автоматически к компрометации всей системы, которая может продолжать работу в штатном режиме.



Контроль взаимодействий

Все взаимодействия между компонентами проверяются на соответствие политикам безопасности и блокируются, если они не входят в список допустимых (default deny). Проверкой политик при этом занимается доверенный компонент. Это позволяет избежать нежелательных взаимодействий между компонентами, которые могут привести к развитию атаки при компрометации одного из них.



¹ <https://cybernews.com/ai-news/replit-ai-vive-code-rogue/>

² <https://encyclopedia.kaspersky.ru/glossary/cyberimmunity/>

³ <https://os.kaspersky.ru/soczialnye-seti/principy-minimizacii-doverennoj-kodovoj-bazy/>

⁴ <https://red.anthropic.com/2026/mythos-preview>

⁵ <https://os.kaspersky.ru/blog/architectural-and-design-requirements-isolation-and-control/>



Кибериммунитет в построении ИИ-агентов

Несмотря на то, что концепция кибериммунитета не была разработана для ИИ-агентов, ее принципы хорошо переносятся на продвинутое LLM-системы. Работая с принципиально недетерминированным и недоверенным ядром – большой языковой моделью – разработчики должны закладывать доверие в архитектуру решения, минимизируя потенциальные последствия его ошибки или компрометации.

Предположения и цели безопасности для ИИ-агента

Предположения безопасности описывают те условия, в которых будет работать система. Для ИИ-агентов это могут быть следующие допущения:

- LLM априори считается недоверенной: мы предполагаем, что она может быть скомпрометирована вредоносной инструкцией или совершить ошибочное действие.
- Мы считаем, что оператор ИИ-агента является доверенным, и риск прямого указания вредоносной инструкции нерелевантен.
- Любые данные, попадающие в систему извне, являются недоверенными, так как могут содержать непрямую инъекцию промпта.
- Пользователь-оператор, который подтверждает действия агента, подвержен усталости и может ошибаться при большой частоте запросов на одобрение высокорискованных действий.

На первом шаге применения кибериммунного подхода необходимо определить цели и предположения безопасности агентной системы. Они зависят от конкретного сценария применения ИИ-агента и обрабатываемых им данных. Примеры целей:

- 1 Конфиденциальные данные (персональные данные, коммерческая тайна) ни при каких условиях не покидают заданный контур (компании, подразделения).
- 2 Любые необратимые или деструктивные действия (финансовые транзакции, удаление данных, отправка внешних коммуникаций) осуществляются только после подтверждения человеком.
- 3 Агент не может расширять свои полномочия (набор инструментов, доступы) самостоятельно, без решения извне.
- 4 Компрометация одного из агентов не влияет на работоспособность других агентов в мультиагентной системе.
- 5 Все действия агента журналируются и являются атрибутируемыми, связываются с соответствующими промптами и размышлениями.

Важно

Списки целей и предположений являются примерными и требуют адаптации к вашему конкретному сценарию и условиям функционирования ИИ в организации. Например, в случае с кейсом SearchLeak применение техники Parameter2Prompt позволяет атакующему представить недоверенный ввод как прямой ввод от пользователя, что нарушает одно из представленных выше предположений.

Минимизация доверенной вычислительной базы

Именно доверенная вычислительная база осуществляет выполнение целей безопасности. В отличие от кибериммунной операционной системы, в которой ядро (микроядро) является доверенным компонентом, ядро агентной системы, а именно большая языковая модель, выносится за ее пределы. Несмотря на все усилия по выравниванию (alignment), обучению моделей не выполнять внедренные в контекст инструкции с помощью иерархии инструкций¹ или обучения специальных адаптеров, эти меры остаются вероятностными и не гарантируют невыполнения опасных команд.

В зависимости от выбранных целей, в доверенную вычислительную базу могут входить:



Агентная обвязка (harness)

Это детерминированный компонент, который выполняет вызов инструментов и работу с контекстом LLM для выполнения заданных пользователем целей (применение навыков, обращение к MCP, реализацию памяти), а также обеспечивает межагентное взаимодействие. Компрометация данного компонента означает компрометацию всей системы. Многие современные обвязки являются крупными системами с большим количеством быстро развивающегося функционала, что может приводить к проблемам безопасности (пример – RCE-уязвимости в LangGraph² или в Microsoft Semantic Kernel³).



Монитор политик (policy engine)

Это доверенный компонент, проверяющий каждый вызов инструмента на его соответствие политикам безопасности. Например, он проверяет, является ли вызов инструмента данным агентом с заданным набором полномочий в данном сценарии (который подразумевает разрешения на определенные вызовы инструментов) соответствующим политикам, и либо разрешает его, либо запрещает с уведомлением центра мониторинга и реагирования на инциденты (SOC). Монитор политик может быть как частью harness, так и отдельным компонентом, работающим на LLM-egress-прокси или уровне MCP gateway.



Механизм изоляции

Ядро песочницы, гипервизор или специальный интерпретатор в случае реализации парадигмы CodeAct⁴, уязвимость в котором может привести к выполнению на хосте инструкций, которые определяются как безопасные только внутри слоя виртуализации.

1 <https://openai.com/index/the-instruction-hierarchy/>

2 <https://labs.cloudsecurityalliance.org/research/csa-research-note-langgraph-rce-chain-20260614-csa-styled>

3 <https://www.microsoft.com/en-us/security/blog/2026/05/07/prompts-become-shells-rce-vulnerabilities-ai-agent-frameworks/>

4 <https://machinelearning.apple.com/research/codeact>

Принцип минимизации в случае агентов можно расширить и на другие компоненты агентной системы. Например, можно минимизировать:

- 

Набор инструментов, предлагаемых агенту

Даже при наличии доверенного монитора политик ИИ-агенту, который генерирует BI-отчеты, нет необходимости иметь возможность проводить платежи. Даже в рамках инструмента для выполнения SQL-запросов агент для построения отчетов не должен иметь возможность выполнять иные команды, кроме SELECT.
- 

Контекст

Можно сравнить ситуацию, когда агент имеет доступ к файлу `.env`, в котором записаны все секреты, которые вообще нужны разработчику для работы над проектом, и, соответственно, может за одно чтение файла получить к ним полный доступ, и когда он может запрашивать секреты только по одному через специализированный инструмент, проверяющий обоснованность запроса. Отсутствие в агенте лишних секретов значительно снижает риск утечки конфиденциальных данных. Дополнительный компонент может также удалять секреты из контекста после того, как они перестали быть необходимы.
- 

Степень агентности

Выше мы говорили о разных степенях агентности, и каждый последующий шаг создает новые риски. Если агент выполняет фиксированный набор действий, ему ни к чему возможность расширять свои возможности за счет агентных навыков. Переход на каждую следующую ступень агентности должен быть осознанным выбором, влияющим на предположения безопасности.

Изоляция компонентов

Изоляция компонентов в случае с LLM-системами может происходить на разных уровнях: изоляция среды исполнения, контекстов и агентов в мультиагентных системах.

Изоляция среды исполнения

Стандартным подходом для обеспечения безопасности агентов, исполняющих код (в первую очередь – это агенты, которые применяются в разработке, но могут и быть и другие примеры, например, агенты для построения ad-hoc бизнес-отчетов, агенты на базе CodeAct и так далее), является изоляция такого кода от системы, на которой запущен процесс. Для этого может быть применено несколько разных подходов:

- 

Ограниченные интерпретаторы, которые поддерживают только «безопасное» подмножество языка, например, в smolagents¹ или CaMeL². Они позволяют ограничить выполнение рискованных команд, но при этом достаточно сильно ограничивают возможности агентов.
- 

Изоляция на уровне процесса, например, такие инструменты как seatbelt (Apple) или bubblewrap (Linux). Имеют достаточно низкие накладные расходы с точки зрения ресурсов, но при этом дают доступ к части файловой системы и системным вызовам хостовой ОС. Это стандартный способ изоляции, применяемый для кодовых ассистентов, таких как Claude Code.

1 https://huggingface.co/docs/smolagents/en/tutorials/secure_code_execution

2 <https://github.com/google-research/camel-prompt-injection/tree/main>

3

Контейнеры Docker/Podman. В случае с контейнерами агент получает достаточно изолированную среду, что, однако, требует определенных затрат с точки зрения ресурсов. При этом ошибки конфигурации такой среды (например доступ к файлам конфигурации контейнеров в случае RCE в AutoGPT¹) могут привести к побегу из контейнера и соответствующему ущербу.

4

Полноценные виртуальные машины и выделенные хосты. Появление OpenClaw в начале 2026 года спровоцировало дефицит² компьютеров Mac Mini, так как пользователи, заинтересованные в личном агенте, готовы были приобрести под него отдельный ПК, осознавая риски запуска агентов с широкими полномочиями на личном устройстве. В корпоративных сценариях виртуальная машина является наиболее надежным, но наименее гибким вариантом создания безопасной среды для агента.

При этом для большей надежности изоляция может дополняться и другими мерами. В частности, на изоляцию может накладываться требование эфемерности – уничтожения окружения после выполнения задачи и создания свежего окружения для решения следующей, чтобы избежать протечки секретов между сценариями. С другой стороны, снятие снимков состояния (снэпшотов) может помочь легко откатиться к предыдущему состоянию после потенциально деструктивных действий моделей и перезапустить задачу с корректного состояния.

1 <https://positive.security/blog/auto-gpt-rce>

2 <https://www.tomshardware.com/tech-industry/artificial-intelligence/openclaw-fueled-ordering-frenzy-creates-apple-mac-shortage-delivery-for-high-unified-memory-units-now-ranges-from-6-days-to-6-weeks>

Изоляция контекстов

Одним из мощных паттернов проектирования безопасных LLM-систем является **изоляция доверенных и недоверенных контекстов**. Одним из примеров реализации такого паттерна является «двойная LLM» (Dual LLM¹). В случае с двойной LLM система подразделяется на два контекста:



Привилегированный, в котором принимаются решения о вызове тех или иных инструментов



Карантинный, в котором происходит обработка недоверенного ввода

Первая имеет доступ к инструментам и может их вызывать, но никогда не видит внешнего недоверенного ввода. Она может запросить работу с этим вводом у карантинной LLM. Та помещает результаты обработки в переменные-указатели, и их названия (но не содержание) передаются в привилегированную LLM. Например, пользователь может попросить у агента отправить ему на почту краткую выжимку из документа. Привилегированная LLM запрашивает у карантинной резюме документа, и та возвращает переменную \$summary. Далее привилегированная LLM вызывает инструмент отправки письма с аргументами в виде \$summary и почтового адреса пользователя. Даже если в документе оказалась промпт-инъекция, которая могла бы спровоцировать агента, например, на спам-рассылку, эта инъекция никогда не оказывается в едином контексте с вызовом функций.

Развитием этой парадигмы является подход **CaMeL² от Google DeepMind**. Он предполагает, что система в целом реализует один из стандартных агентных подходов Plan-and-Execute: привилегированная LLM генерирует план выполнения задачи на языке Python, который выполняется в специальном защищенном интерпретаторе. Среди доступных команд – возможность воспользоваться инструментами и выполнить запрос к карантинной LLM для обработки недоверенных данных. При этом результаты вывода карантинной LLM должны отвечать четкой структуре, а вызовы инструментов подвергаются проверке с точки зрения политик. Например, агент может получить задачу «оплати счет из файла invoice.pdf». Файл может содержать невидимую инъекцию, которая требует у агента удалить все файлы из папки. Привилегированная LLM составит план, состоящий из двух команд: с помощью карантинной LLM извлечь из счета необходимые реквизиты, а затем вызвать инструмент оплаты. Карантинная LLM не может удалять данные, а структурированный вывод с большой вероятностью заставит ее игнорировать инъекцию и вернуть в основной поток исполнения реквизиты. С другой стороны, злоумышленник может добавить инъекцию, убеждающую агента, что номер счета получателя платежа изменен на его номер, а сумма увеличена в 10 раз. В таком случае инъекция может сработать, и карантинная LLM вернет искаженные данные, но политика на исполнение инструментов (например, «отправлять без подтверждения деньги только на доверенные счета» или «требовать ручного подтверждения на переводы больше определенной суммы») предотвратит ущерб.

Очевидно, что внедрение таких паттернов сильно усложняет разработку, повышает стоимость работы агента (в токенах), а также снижает эффективность агентов: на бенчмарках типа AgentDojo у CaMeL наблюдается достаточно сильная просадка по метрикам. Тем не менее, применение таких паттернов оправдано в хорошо структурируемых сценариях, связанных с высоким риском.

1 <https://simonwillison.net/2023/Apr/25/dual-llm-pattern/>

2 <https://arxiv.org/abs/2503.18813>

Межагентная изоляция

При масштабировании систем до мультиагентных появляются новые риски, что приводит к необходимости применять меры безопасности к межагентным взаимодействиям. Среди таких мер могут быть:

1

Применение политик безопасности и агентной identity к каждому новому агенту. Например, субагент не должен автоматически наследовать полный набор полномочий и identity оркестратора.

2

Признание вывода любого агента недоверенным при его переиспользовании другим агентом, потенциально – применение для агентов и субагентов подхода с двойной LLM для митигации рисков каскадирования ошибок (ASI08).

3

Применение политик доступа к разделяемым ресурсам, таким как долговременная память или базы знаний для RAG.

Контроль взаимодействий

Наконец, третьим принципом кибериммунитета является контроль взаимодействий между компонентами системы. Все взаимодействия между частями проверяются доверенным монитором взаимодействий, действующим по принципу «что явно не разрешено, то запрещено» (default deny). Контроль может осуществляться на различных уровнях и в разных точках.



Контроль вызовов инструментов

Для каждой задачи имеется четкий список дозволенных инструментов. При этом на аргументы (как в примере с платежным агентом в CaMeL) накладываются параметризованные политики (максимальная сумма платежа, доступные для записи директории и так далее). Категоризация действий на обратимые (изменение файлов в рамках репозитория с системой контроля версий) и необратимые (отправка корреспонденции, криптовалютные транзакции) позволяет применять разные уровни контроля – от его полного отсутствия до необходимости вмешательства оператора. Наконец, на инструменты могут накладываться ограничения по частоте вызовов, что может уменьшить потенциальный ущерб.



Контроль сетевых взаимодействий

Меры могут варьироваться от постоянного аудита Content Security Policy для противодействия атакам, подобным ForcedLeak, до default deny на egress-роу с явным списком разрешенных для каждого конкретного агента адресов для исходящих сетевых соединений.



Контроль динамической цепочки поставок

Как правило, эта мера подразумевает запрет на установку любых внешних агентных навыков и подключение к MCP-серверам во время исполнения: AIBOM должен быть задан заранее и не меняться в процессе работы агента.

Вероятностные меры обеспечения безопасности

Среди наиболее обсуждаемых способов обеспечения безопасности LLM-приложений на данный момент выделяются два: выравнивание (alignment), то есть обучение модели противодействовать атакам на уровне весов, и внешние гардрейлы, то есть дополнительные модели или алгоритмы, оценивающие безопасность ввода и вывода и часто входящие в состав продуктов класса AI Firewall. Хотя было отмечено, что эти меры не могут являться достаточным условием кибериммунитета решения, их применение оправдано по ряду причин. Во-первых, повышенная устойчивость модели к атакам за счет выравнивания или гардрейлов снижает нагрузку на оператора при необходимости в запросе подтверждения, что снижает его усталость. Во-вторых, они являются необходимой подстраховкой при обнаружении изъянов в безопасной архитектуре, например ошибках в доверенной кодовой базе. В-третьих, они являются важным источником телеметрии даже о предотвращенных на архитектурном уровне атаках, что позволяет уточнять применяемые политики безопасности и модель угроз, что, разумеется, требует наличия развитых средств обеспечения наблюдаемости (observability). Таким образом решения типа AI Firewall, вместе с другими современными компонентами безопасности (sandbox, EDR, SIEM и так далее), хоть и не заменяют безопасную архитектуру, но являются важной частью многослойной защиты современных приложений.

Выводы и рекомендации

Анализ архитектуры агентных систем, существующих моделей угроз и известных инцидентов через призму кибериммунитета приводит к следующим принципиальным выводам:

1

LLM невозможно сделать доверенной частью вашей системы вне зависимости от ее способностей и происхождения из-за ее вероятностного характера. Это означает, что обеспечение гарантий безопасности нужно возлагать не на большую языковую модель, а на те детерминированные компоненты, которые с ней взаимодействуют.

2

Безопасность агента определяется на этапе проектирования. Наличие и допустимость «фатальной триады», определение степени агентности, применение паттернов вроде «двойной LLM» и так далее являются архитектурными решениями. Любые наложенные постфактум ограничения – политики для инструментов или гардрейлы на egress-proxy – могут митигировать риски, но не предотвращать их целиком.

3

Практически любые ограничительные меры приводят к падению скорости, удобства или способностей ИИ-агентов. Поиск баланса безопасности и эффективности – основная задача при проектировании кибериммунных агентов, и моделирование рисков путем определения целей безопасности – важный шаг к поиску правильного их соотношения.

4

Элементы кибериммунного подхода прекрасно подходят для моделирования защиты ИИ-агентов. Изоляция компонентов (за счет применения песочниц, паттерна «двойной LLM» и MCP-шлюзов), контроль взаимодействий (policy engine в CaMeL), минимизация доверенной базы (применение мощных агентных фреймворков на этапе проектирования и написание минимального специализированного кода для промышленной эксплуатации) – все эти меры возникли в дискурсе о безопасности ИИ-агентов независимо, но в сочетании могут приблизить агентную систему к состоянию иммунной.

Практические рекомендации для CISO / CIO / CTO для построения безопасных агентных архитектур

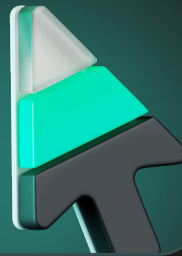
Проводите инвентаризацию агентов

OpenClaw, кодировщики, встроенные в облачные консоли ассистенты – составьте полный список и проанализируйте их по степени агентности, доступу к конфиденциальным данным, наличию «фатальной триады».



Управляйте агентной цепочкой поставок

Требуйте AIBOM для агентных решений, проверяйте безопасность компонентов цепочек поставок с помощью специализированных решений, отслеживайте проблемы с цепочкой поставок с помощью фидов.



Взаимодействуйте с командами,

занимающимися внедрением агентов для разработки¹ или созданием собственных агентов, для управления архитектурными требованиями безопасности, определения целей безопасности, управления уровнями агентности для внедряемых решений.



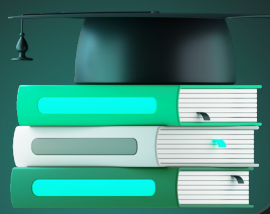
Кроме архитектурных требований,

прорабатывайте гибкие правила по изоляции и контейнеризации агентов, вырабатывая правильный баланс между уровнем изоляции и удобством использования. Защищайте соответствующие контейнерные среды.



Сделайте обязательным требованием наблюдаемость

Как сбор LLM-трейсов, так и получение хостовой телеметрии.



Помните о необходимости интегрированной многослойной защиты

Обеспечение безопасности рабочих станций, специальный egress-проxy для LLM, AI Firewall и другие решения, сочетание которых позволяет максимально снизить потенциальные риски агентных систем.



Авторы исследования:

Владислав Тушканов

Руководитель группы исследования технологий машинного обучения «Лаборатории Касперского»

Александр Лискин

Руководитель управления исследования угроз «Лаборатории Касперского»

¹ <https://lp.kaspersky.com/ru/ai/>