Runtime-контроль ИИ-агентов: единица контроля не модель, а действие

Источник. Блог компании INFERA Security на HABR

https://habr.com/ru/companies/infera_security/articles/1080536/

Пока рынок спорил, достаточно ли «умного» фильтра на промпт, агенты уже начали вызывать инструменты, ходить в MCP, трогать платёжные API и уносить ключи с рабочих станций. Для нас это был момент, когда защита модели перестала быть достаточным контуром.

Промпт можно инспектировать, но действие нужно останавливать до исполнения.

В этой статье опишем рабочую модель runtime-контроля, которую мы заложили в INFERA AI.SafeAgent. Это не каталог фич. Это то, как устроен контур, который видит эксплуатация: где перехватывается вызов, как компилируется политика, какой вердикт выносится до исполнения действия и почему «нашли всех агентов» ещё не равно «контролируем агентов».

За агентом всегда есть контур сущностей: скиллы и расширения, MCP-серверы и их инструменты, модели, машины и эндпоинты, сессии, инструменты вызова (tool-call), обращения в сеть, граф межагентного взаимодействия.

Если сотрудник с ноутбука вышел за контролируемый перечень ИИ-моделей и сходил во внешнюю LLM – это тоже событие контура, которое попадает в блок «теневого» ИИ (Shadow AI). Чтобы это было контролируемо – нужно понимать, что происходит на рабочей станции сотрудника. Без агента на хосте остаётся только то, что проходит через шлюз.

Два контура защиты, а не один «умный фильтр»

Архитектура намеренно разделена между LLM/AI Firewall и контролем ИИ-агентов.

Контур агента (INFERA AI.SafeAgent) реализует быстрые проверки: полномочия, белые списки инструментов, ограничения аргументов, MCP-обёртка, локальные предохранители, запрет при недоступности контроля (fail-closed).

Контур шлюза (INFERA AI.Firewall) забирает семантику: инъекции, обход ограничений, токсичность, длинные тексты, маршрутизация к внешним моделям.

На агенте мы специально не гоняем тяжёлые ML-классификаторы. Иначе runtime начнёт тормозить сам сценарий, ради которого агента и ставили. Семантический трафик уходит на шлюз. Политики при этом компилируются на control plane и раздаются агентам уже в исполняемом виде.

Это следствие инженерии, а не вкуса. **Точка перехвата живёт рядом с действием.** Модель-судья живёт там, где есть резерв по задержке и полный текст.

Точки перехвата разные, потому что агенты живут в разных средах:

- MCP-wrapper – обёртка серверов инструментов клиента;
- hook-адаптеры для IDE/CLI (Claude Code, Cursor, Codex и аналоги);
- eBPF на Linux для контроля системных вызовов;
- ETW на Windows;
- INFERA AI.SafeAgent на хосте;
- принудительный egress через INFERA AI.Firewall (base_url шлюза).

Покрытие не одинаковое, поэтому важно понимать, что если нет точки перехвата, то нет контроля.

Три слоя покрытия, а не бинарное «защищён / не защищён»

Агенты в контуре живут на трёх уровнях.

- **У0 – только обнаружение:** агент не считается безопасным, он виден, но карточки и политики ещё нет.
- **У1 – допущен и просканирован:** компоненты проверены, агент выведен из чистого Shadow AI, но исполнение ещё не замкнуто политикой.
- **У2 – подключён к политике:** назначен профиль доступа, вызовы идут через движок допуска.

Это принципиально. Многие внедрения ломаются на фразе «мы нашли всех агентов». Находка без политики – это инвентаризация, а не контроль. Пока агент не на У2, у него нет запрета по умолчанию (deny-by-default) на инструменты.

Метрика, которую стоит смотреть – это не «сколько агентов в реестре», а **доля агентов на У2** и доля теневых сущностей, которые ещё не получили карточку.

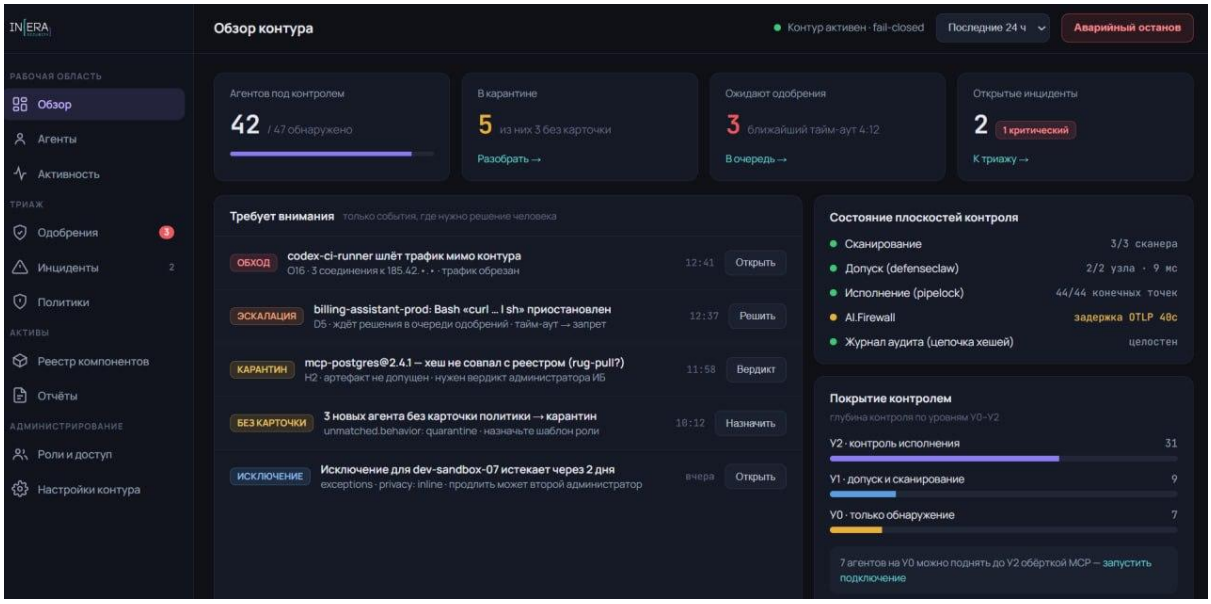

Рис.1. INFERA AI.SafeAgent

Варианты действий:

1. **Сканирование** компонентов до того, как агент получит к ним доступ.
2. **Наблюдение** и сохранение рассуждений, запросов, tool-call, сетевых обращений, сессий.
3. **Управление** и политика, которая на каждый перехваченный вызов даёт вердикт: разрешить, записать, замаскировать, эскалировать человеку, запретить или остановить агента.

Например: агент без карточки → карантин, новый MCP или скилл → карантин и скан, инструмент вне списка → запрет и оповещение.

Как оценивается один вызов:

1. Агент планирует действие: вызывает инструмент (tool-call), MCP, обращается к модели, дает команду ОС.
2. Срабатывает точка перехвата, вызов останавливается **до** исполнения.
3. Движок допуска применяет скомпилированную политику этого агента (конкретную версию профиля).
4. Сначала белый список полномочий, если вне списка – запрет.
5. Затем действуют правила рисков по контексту: инструмент, risk_score, пользователь, MCP, скилл, класс данных, источник.
6. Потом DLP на аргументах и выводе, сеть, бюджет, предохранители.
7. Вердикт фиксируется, ссылается на версию профиля и уходит в control plane.

На тестах движок допуска на стороне агента обычно укладывается в 10 миллисекунд. Это сознательный потолок, поэтому тяжёлая семантика делается на шлюзе.

Три вердикта «разрешить / запретить / спросить человека» – слишком грубая модель для агента, который одновременно читает файл, вызывает MCP и пишет в платёжный API. Поэтому в INFERA SafeAgent их шесть.

Вердикт	**Что происходит**
Разрешить (allow)	вызов идёт дальше
Журнал (log)	вызов идёт дальше, но остаётся в ленте
Маскировать (redact)	чувствительные поля маскируются
Эскалация (escalate)	очередь HITL (Human-in-the-Loop), человеку локально и/или на консоль
Блокировать (block)	запрет до исполнения
Останов (kill)	агент гасится, снимается криминалистический снимок

Из чего состоит профиль доступа

Профиль – это не «роль агента», а скомпилированный контур контроля. Секции соответствуют слоям, которые реально исполняются на агенте и шлюзе.

Полномочия. Белый список методов и инструментов. Всё, что не разрешено явно, запрещается. Явный запрет сильнее разрешения. Ограничения задаются проверяемым форматом: glob-пути, домены, числовые лимиты, предикаты. Для файловой системы работает отрицание: разрешить рабочий каталог и запретить ~/.aws/ и ~/.ssh/**. Опасные шаблоны (rm -rf /, curl ... | sh) закрываются отдельным набором.

Правила рисков. Конструктор «атрибут → оператор → значение → вердикт». Свободный текст в условии исключён: такое правило нельзя исполнить на стороне агента предсказуемо.

DLP. Инспекция аргументов и вывода по классам: ПДн, учётные данные, секреты, исходный код, платёжные (PCI), API-ключ. Реакция: блок, маскирование или журнал. Классификатор локальный, данные не обязаны покидать контур. Имя каждого секрета из брокера автоматически попадает в класс «Секреты».

Сеть. Разрешённые домены, принудительный base_url LLM через INFERA AI.Firewall, перечень моделей. Egress к внешней модели мимо шлюза – это не «ещё один лог», это обход контура.

MCP-реестр. Режим registry_only: неизвестный сервер уходит в карантин и на скан, а не в прод. Допуск фиксируется по хешу (имя@sha256). Это закрывает тайпсквот вроде gihub-mcp вместо github-mcp и подмену содержимого при неизменном имени и версии.

Ограничения. Максимум итераций и токенов за сессию. Kill-условия только из измеримых триггеров: серия срабатываний DLP, серия запретов, порог risk_score, повтор действия, обход контура, превышение бюджета. HITL с маршрутом «админ / локально / оба» и правилом двух лиц на необратимые операции. Дневной бюджет токенов на агента, а денежные лимиты контролируются на стороне шлюза.

Секреты. Агент не хранит ключ. В профиле ссылка, а значение подставляет брокер на границе egress. Хранилище: Vault, OS keyring или шифрованный файл. Найденный при обнаружении ключ импортируется и заменяется ссылкой, без ручного ввода в конфиг агента.

Это и есть least privilege для агента.

Теневой ИИ – это отдельный контур

Самый частый сюрприз на пилоте: формально утверждён один Copilot, а на рабочих станциях сотрудников уже живут Cursor, личные ключи OpenAI, самописный MCP и браузерные сессии с внешними моделями.

Теневой ИИ в INFERA AI.SafeAgent собирается из того, что не требует тяжёлого realtime-разбора на хосте:

- запущенные процессы;
- артефакты браузера;
- сохранённые ключи;
- журнал сетевых обращений.

Сетевой анализ на станции сейчас не ставит inline-блокировку каждого пакета. Мы восстанавливаем, куда ходила станция, и видим выход за доверенный перечень моделей. Такие обращения журналируются и разбираются в карточке агента. Блокировка исполнения включается там, где есть точка перехвата и политика. Путать эти два режима нельзя: иначе на демо обещают «мы режем любой внешний ChatGPT с ноутбука», а в проде оказывается, что без установленного сенсора и без шлюза резать нечем.

Карточка агента всегда привязана к эндпоинту. Туда складываются сессии, инциденты, полномочия и граф взаимодействия. Если агент начал ходить в другого агента, это должно быть видно не в логе приложения, а в контуре безопасности.

Реакции на теневую сущность разные: карантин, оповещение SOC, блокировка egress через шлюз. Это не один и тот же рычаг.

HITL – это операционная функция, а не кнопка «спросить человека»

Ручное одобрение легко превратить в ритуал, который все кликают «разрешить». Поэтому очередь HITL у нас отдельный экран, а не сноска в инцидентах.

Смотреть нужно не только «есть ли согласование», а:

- по каким агентам оно требуется слишком часто;
- какие типы вызовов постоянно эскалируются;
- как быстро принимают решение;
- не пора ли сузить полномочия, вместо того чтобы держать людей на конвейере.

Если HITL срабатывает на каждый второй вызов, политика собрана неправильно. Пользователи начнут пропускать согласования, и контур формально жив, а фактически нет.

Несколько жёстких правил, без которых HITL разваливается:

- согласование выполняется вне контекста агента, тогда агент не формирует и не правит текст запроса;
- по истечении SLA действие уже запрещено (fail-closed), а не «повисло и потом прошло»;
- для необратимых и платёжных операций действует правило двух лиц, и оно идёт через SOC, не через локальную утилиту;
- если очередь растёт, чинить нужно белый список инструментов, а не добавлять согласующих.

Есть и жёсткий контур: Kill Switch с Обзора. Область останова – один агент, все с высоким риском или весь контур. Действие пишется в подписанный журнал. Агента вне перехвата kill switch не догонит: его сначала надо изолировать как теневой.

Отдельно живёт останов сессии: прервать активную трассу, снять криминалистический снимок, не обязательно гася весь агент навсегда.

Сканеры: supply chain агентной среды

До runtime есть ещё один контур, без которого политика охраняет пустоту: реестр компонентов.

Сканер проверяет скиллы, MCP, модели как артефакты среды: сигнатуры и хеши, разрешения, секреты, SBOM, поведение в песочнице. Пресеты strict / balanced / lenient задают набор правил. Контрактные тесты гоняют эталонные случаи: rug-pull MCP, неизвестный инструмент, утечка секрета, проверка издателя.

Компонент, не прошедший проверку или отсутствующий в реестре, агенту недоступен. Подмена содержимого при неизменных имени и версии ловится сравнением контрольных сумм и уводит компонент в карантин. Отзыв допуска распространяется на конечные точки, включая уже идущие сессии.

Это ответ на агентный supply chain.

Какие требования к runtime-контролю являются ключевыми

Рынок сейчас описывает это языком Guardian Agents и агентных Top-10. Если отбросить маркетинговый слой, рабочий минимум runtime-контроля выглядит так.

1. **Перехват до исполнения (side effect).** Белые/чёрные списки инструментов недостаточны, если вызов уже ушёл в платёжный API.
2. **Запрет по умолчанию (*deny-by-default*) и запрет при недоступности контроля (*fail-closed*).** Инструмент вне списка не вызывается. Недоступность контроля не открывает дыру.
3. **Единица контроля – агент**, привязанный к эндпоинту, владельцу, профилю и версии политики.
4. **Три слоя покрытия.** Обнаружение ≠ допуск ≠ контроль исполнения.
5. **Разделение семантики.** Локальный движок допуска и шлюз для длинного текста.
6. **Версионируемая политика как код.** Пробный прогон на трассах, подпись, откат, запрет бесконечного ослабления.
7. **HITL как очередь с SLA и правилом двух лиц**, а не как checkbox в карточке агента.
8. **Реестр MCP/скиллов по хешу** и карантин неизвестного до первого использования.
9. **Принудительный egress к внешним моделям.** Иначе часть инъекций и утечек вы просто не увидите.
10. **Предохранители и kill switch.** Лимит итераций, токенов, серия DLP/запретов, аварийный останов области.
11. **Брокер секретов.** Агент не видит исходное значение ключа.
12. **Полная трасса сессии** с trace_id: пользователь → агент → инструмент → целевая система → вердикт.
13. **Инвентаризация теневого ИИ** по процессам, ключам, браузеру и журналу сети – отдельно от inline-блокировки.
14. **Честная карта покрытия** по средам и коннекторам. Не обещать Windows/SaaS тот же рычаг, что у Kubernetes.

По OWASP Top 10 for LLM и OWASP Top 10 for Agentic Applications мы сознательно не обещаем закрыть всё одним модулем. Tool misuse, excessive agency, supply chain компонентов, rogue/shadow agents, unbounded consumption и аварийный останов закрываются здесь, в runtime. Prompt injection на длинном тексте – на шлюзе. Обучение модели и часть poisoning – в MLSecOps. Часть рисков вообще организационная.

Это нормально. Плохо, когда один продукт притворяется, что он закрывает весь каталог рисков и защищает от всех угроз.

Какие кейсы рекомендуем закрывать первыми

Не начинайте с «защитить всех агентов компании». Начинайте с контуров, где действие уже имеет цену и серьезный риск для бизнеса компании.

1. Теневой ИИ на рабочих станциях разработки

Cursor, Claude Code, Codex, личные API-ключи, самовольно поднятые MCP. Это самый быстрый способ сделать инвентаризацию и понять реальный масштаб. Первый результат внедрения почти всегда не «мы защитили прод», а «у нас в два раза больше агентов, чем думал CISO».

Пока теневой слой не разобран, политика будет охранять только тех, кто и так уже под контролем.

2. MCP-supply chain

Подмена сервера, скрытый tool, тайпсквот, смена хеша при той же версии. В реестре компонент живёт с доверием и контрольной суммой. Режим registry_only должен включаться сразу: неизвестный MCP – сразу в карантин, а не в прод.

3. Агенты с правом записи и выполнения платежей

Платёжный агент, refund, выгрузка ПДн, изменение заявки, смена прав. Здесь шаблон strict: deny by default, DLP на ПДн/PCI/секретах, HITL с двумя лицами, fail-closed, телеметрия хешем, предохранители, kill при обходе контура.

Если пилот начинать с IDE-агента разработчика, бизнес не увидит риска. Если начать с billing-агента, то риск очевиден всем.

Практический каркас такого профиля у нас выглядит так:

- payments.refund: эскалация, правило двух лиц;
- payments.charge: блок сверх лимита;
- sql.select: только нужные таблицы;
- DLP: ПДн маскировать, PCI и секреты блокировать;
- MCP только stripe-mcp@sha256:…;
- 40 итераций / 200 000 токенов на сессию;
- kill при серии DLP или обходе шлюза.

4. Принудительный выход к внешним моделям через шлюз

Пока LLM-трафик может уйти мимо LLM Firewall, часть инъекций и утечек вы не увидите. INFERA AI.SafeAgent на хосте и INFERA AI.Firewall на egress – это два слоя одного контура. Для SaaS-агентов без хостового перехвата это вообще единственный рабочий рычаг.

5. Runaway и обход контура

Лимит итераций, лимит токенов, условие останова по серии запретов или срабатываний DLP, детект смены base_url и обхода шлюза. Агент, который зациклился и начал жечь бюджет или долбить API, должен гаситься политикой, а не тикетом в FinOps на следующий день.

Kill Switch проверяйте до инцидента, а не во время.

И только после этого можно переходить к массовому подключению сервис-агентов.

Порядок важен. Если сразу повесить строгий HITL на всех IDE-агентов, команда обойдёт контур за неделю. Если сначала не закрыть Shadow AI и реестр MCP, политика будет декорацией.

Смотрите не «есть ли ИИ-безопасность», а покрытие У2, долю теневых агентов, частоту HITL, наличие kill switch, долю MCP в режиме registry_only и факт, что внешние модели не ходят мимо шлюза. Это те же вопросы, которые рынок описывает как Guardian Agents: видно ли действие, можно ли его остановить, совпадает ли оно с намерением.

Не начинайте программу с «подключим всех». Начните с контура, где у действия есть конкретная цена – деньги, ПДн, запись в прод – и с инвентаря того, что уже живёт на станциях без вашего ведома.

Если коротко: агенту можно разрешить думать, но нельзя разрешать действовать без политики.