BIS JOURNAL

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ БИЗНЕСА

№3 (62) 2026

Игорь КАЧАЛИН
АНО НТЦ ЦК
«Наша школа остается сильнейшей»
→ 24

Станислав СМЫШЛЯЕВ
КриптоПро
«Криптография — фундамент без трещин»
→ 5

Криптография. Эпохи меняются — ценности остаются
→ 5

SOC. ИИ меняет правила → 34

РБПО. Сертификация как осознанная необходимость → 58

Дмитрий ГУСЕВ
ИнфоТеКС
«Экономьте время!»
→ 8

Дмитрий ГОРЕЛОВ
Актив
«Пора клонировать сотрудников»
→ 10

Карина НАПАДОВСКАЯ
АО «Лаборатория Касперского»
11 вопросов после 11-го Certification Day
→ 58

«ДОВЕРИЕ НЕЛЬЗЯ СЕРТИФИЦИРОВАТЬ — ЕГО МОЖНО ТОЛЬКО ПОСТРОИТЬ»

11 ВОПРОСОВ ПОСЛЕ 11-ГО CERTIFICATION DAY

Карина НАПАДОВСКАЯ
руководитель центра сертификации и соответствия стандартам АО «Лаборатория Касперского»

После одиннадцатого Certification Day BIS Journal решил отказаться от привычного разговора о количестве участников, программе и итогах мероприятия. Вместо этого мы задали одиннадцать вопросов человеку, который уже одиннадцать лет наблюдает, как меняется система сертификации средств защиты информации изнутри.

1. Карина, одиннадцать лет назад вы создали Certification Day. Чего тогда, на ваш взгляд, не хватало отрасли?

— В то время хороших отраслевых конференций уже было достаточно. Но всё время оставалось ощущение, что отрасли не хватает совсем другой площадки. Не той, где рассказывают о том, что уже получилось. А той, где можно спокойно говорить о том, что пока не получается. Где разработчики, регулятор, испытательные лаборатории и органы по сертификации могут вместе обсуждать вопросы, на которые ещё нет готовых ответов.

Я никогда не любила разговоры о том, почему что-то невозможно. Мне всегда было интереснее спросить: «Что мы можем сделать, чтобы это стало возможным?»

Именно с этой мысли и начался Certification Day. Мне хотелось создать не ещё одну конференцию, а профессиональную среду, где можно открыто обсуждать сложные вопросы, вместе искать решения и помогать новым подходам становиться частью реальной практики.

2. Одиннадцать лет спустя — получилось? Certification Day стал именно такой площадкой, какой вы когда-то хотели его видеть?

— Да. Наверное, лучшим подтверждением стало то, что наш разговор не заканчивается вместе с мероприятием. Продолжаются дискуссии на поднятые темы. Появляются рабочие группы. Новые идеи проходят проверку на практике. Через год люди возвращаются уже с результатами совместной работы, которая началась на предыдущем Certification Day. Сегодня я вижу, что многие вопросы, которые раньше обсуждались в узком кругу или в кулуарах, теперь становятся предметом профессиональной дискуссии.

Для меня это, наверное, и есть главный результат этих одиннадцати лет.

3. Как вы понимаете, что идея действительно созрела для масштабирования?

— Если оглянуться назад, становится заметно, что случайностей почти не было. Практически все успешные инициативы проходили один и тот же путь. Сначала появлялась идея или вопрос, на который ещё не было готового ответа. Затем начиналась профессиональная дискуссия. Очень важно, чтобы в ней участвовали все стороны процесса, потому что каждая видит ситуацию по-своему. Но сама по себе дискуссия ничего не меняет. Следующий этап — практическая апробация. Если идея действительно жизнеспособна, она должна пройти проверку в реальных условиях. Как правило, сначала на одном или нескольких вендорах, готовых первыми опробовать новый подход на практике. И только после этого начинается самое интересное. Если решение действительно работает, его начинают применять другие участники рынка. В этот момент оно перестаёт быть инициативой отдельных энтузиастов и становится отраслевой практикой.

Именно так и рождаются устойчивые изменения. Не потому, что появилась хорошая идея. А потому, что она прошла путь от профессиональной дискуссии до практики, которой начинает доверять вся отрасль.

4. Можете привести пример идеи, которая на ваших глазах прошла весь этот путь — от профессиональной дискуссии до отраслевой практики?

— Таких примеров за одиннадцать лет накопилось немало. Но если выбирать один, я, наверное, назову электронную поставку сертифицированных продуктов.

Когда эта идея только появилась, она вызывала немало сомнений. Все привыкли, что сертифицированный продукт обязательно связан с физическим комплектом поставки. Но про-

граммное обеспечение обновлялось всё быстрее, поскольку количество новых уязвимостей, включая уязвимости в сторонних компонентах, ежегодно росло. Только за последнее десятилетие число ежегодно регистрируемых уязвимостей и дефектов безопасности (Common Vulnerabilities and Exposures, CVE) увеличилось почти в четыре раза.

Стало очевидно, что существующая модель уже не успевает за скоростью развития программного обеспечения. Электронная поставка была не просто новой идеей. Она стала неизбежным ответом на изменение реальности. Подход был апробирован на практике.

Сегодня он — естественная часть жизненного цикла сертифицированного продукта. Пользователь может оперативно обновиться до безопасной версии, не выводя информационную систему из аттестованного состояния.

5. А есть ли пример, где изменилась не практика, а сама философия сертификации?

— Таким примером стала модель самостоятельной сертификации продуктов разработчиком. Идея была в следующем: разработчик лучше всех знает собственный продукт. Особенно когда речь идёт о небольших изменениях, обновлениях безопасности или исправлении отдельных уязвимостей. Поэтому вполне логично использовать эту экспертизу там, где она действительно позволяет сделать процесс быстрее и эффективнее. Такая модель возможна только тогда, когда процессы разработки действительно зрелые, прозрачные и способны обеспечивать стабильное качество результата.

Именно здесь и произошло самое важное изменение. Сертификация всё меньше отвечает только на вопрос: «Безопасен ли этот продукт сегодня?» И всё больше — на вопрос: «Способна ли организация создавать безопасные продукты завтра?» Именно в этом и заключается главное изменение философии сертификации.

6. Такие изменения невозможно реализовать в одиночку. Что, на ваш взгляд, делает их возможными?

— В системе сертификации средств защиты информации у каждого своя роль. Регулятор определяет направление развития и задаёт требования. Разработчики создают продукты, соответствующие этим требованиям. Испытательные лаборатории проводят независимую оценку их соответствия. Органы по сертификации подтверждают соответствие. Регулятор принимает решение о выдаче сертификата на основе результатов испытаний и экспертизы. Каждый выполняет свою часть работы. И именно так система должна работать.

Если оглянуться назад, самым важным результатом этих одиннадцати лет стали даже не новые подходы к сертификации. Самым сложным оказалось создать профессиональную среду, в которой люди начинают доверять друг другу настолько, чтобы вместе искать решения общей задачи

Но за одиннадцать лет я поняла ещё одну важную вещь. По-настоящему сильные решения появляются тогда, когда между всеми участниками возникает профессиональный диалог. Для того, чтобы лучше понять друг друга, увидеть практические особенности применения новых подходов и вместе найти наиболее эффективные способы их реализации.

7. Можете привести пример, когда именно такой профессиональный диалог помог развитию системы сертификации?

— Таких примеров было несколько. Одним из первых примеров стала работа над Методикой выявления уязвимостей и недекларированных возможностей. Для любой новой методики важно не только качество самой идеи, но и понимание того, насколько она применима на практике. Именно поэтому была организована её апробация с участием ИСП РАН, «Лаборатории Касперского», компаний «Код Безопасности» и «Фобос». Мы не обсуждали методику как теоретический документ. Мы проверяли, насколько она действительно работает в реальных сертификационных испытаниях.

По такому же принципу строилась работа и по другим инициативам.

Практика применения Требований доверия позволила профессиональному сообществу накопить опыт, увидеть вопросы, возникающие при их реализации, и сформулировать предложения, которые были представлены регулятору и учтены при дальнейшем развитии документа.

Аналогичным образом создавалась и новая редакция ГОСТ Р 56939–2024 «Защита информации. Разработка безопасного программного обеспечения». Документ разрабатывался совместно регулятором, рабочей группой вендоров и ИСП РАН. Такое взаимодействие позволило объединить регуляторное видение, научную экспертизу и практический опыт разработки безопасного программного обеспечения.

8. Сегодня всё больше внимания уделяется сертификации процессов безопасной разработки. Почему именно сейчас?

— Я считаю это естественным этапом развития отрасли. Современное программное обеспечение постоянно развивается: выходят новые версии, исправляются уязвимости. В этих условиях становится недостаточно ответить на вопрос: безопасен ли продукт в момент проведения сертификационных испытаний? Важно понимать, способна ли организация сохранять этот уровень безопасности на протяжении всего жизненного цикла продукта. Именно поэтому всё большее значение приобретает оценка процессов безопасной разработки. Сегодня такие сертификаты становятся новой отраслевой практикой.

И здесь главный вопрос уже заключается не в количестве выданных сертификатов. Важно сохранить их содержательную ценность. Сертификат должен быть не маркетинговой наклейкой и не формальным подтверждением соответствия. Он должен быть доказательством зрелости процессов разработчика. Именно поэтому для меня сертификация процессов — это не про ещё один сертификат. Это про уверенность, что организация умеет системно создавать безопасные продукты.

9. Какой вопрос, на ваш взгляд, станет для отрасли следующим большим профессиональным вызовом?

— Конечно, безопасное внедрение технологий искусственного интеллекта. Их развитие происходит слишком быстро, чтобы устоявшиеся подходы к регулированию успевали формироваться с той же скоростью. Поэтому следующим большим профессиональным вызовом станет поиск механизмов, которые позволят безопасно проверять на практике новые технологии ещё до того, как они станут массовыми.

Именно поэтому тема регуляторных песочниц представляется мне одной из наиболее перспективных. Я рассматриваю их не как способ смягчить требования, а как возможность для накопления практического опыта в контролируемых условиях. По сути, это продолжение той же самой логики, которой отрасль придерживалась все последние годы: сначала профессиональный диалог, затем апробация и только потом — широкое внедрение новых подходов.

10. Что за эти одиннадцать лет осталось для вас неизменным?

— Убеждение, что самые сложные вопросы нужно не обходить, а обсуждать. Проблемы надо решать, а не делать вид, будто их не существует.

11. Если бы вам было нужно сформулировать главный итог этих одиннадцати лет одной мыслью, какой бы она была?

— Если оглянуться назад, самым важным результатом этих одиннадцати лет стали даже не новые подходы к сертификации. Самым сложным оказалось создать профессиональную среду, в которой люди начинают доверять друг другу настолько, чтобы вместе искать решения общей задачи.

Доверие нельзя сертифицировать. Его можно только построить.

Вопросы задавал
Александр Абрамов

ИИ-БЕЗОПАСНОСТЬ СТАНОВИТСЯ РЕАЛЬНОСТЬЮ

РОССИЙСКИЙ БИЗНЕС НА ПОРОГЕ НОВЫХ КИБЕРРИСКОВ

Юрий ШАБАЛИН
директор по развитию технологий искусственного интеллекта Swordfish Security

К 2026 году ИИ стал частью управления бизнес-процессами. Вместе с многократным ускорением процессов в повседневность компаний вошли и новые киберугрозы. Расширение поверхности атак — от компрометации данных до манипуляции поведением моделей — требует от бизнеса перехода к стратегии MLSecOps.

Искусственный интеллект — это не линейный алгоритм. Именно из этой непрозрачности рождаются угрозы, которые невозможно закрыть привычными инструментами информационной безопасности.

Среди наиболее характерных для ИИ-систем атак можно выделить три основные категории. Первая — промпт-инъекции, когда пользователь пытается обойти встроенную логику модели с помощью хитро сформулированных запросов. Вторая — отравление данных, то есть намеренное внесение вредоносной информации в обучающую выборку модели. Третья — утечки через ответы, когда нейросеть в ходе обычного диалога неожиданно выдаёт конфиденциальные сведения компании, которые не должна была раскрывать.

Для ИИ-моделей угрозой является не код в привычном понимании, а смысл — семантика запроса и контекст диалога. Именно поэтому на первый план сегодня выходят так называемые ИИ-шлюзы, или AI Gateways,— новая архитектурная прослойка, которая должна выполнять функцию контрольно-пропускного пункта между миром корпоративных данных и вычислительной мощью нейросетей.

Отвечая на этот запрос рынка, компания AppSec Solutions представила собственное решение — AppSec.AIGate. Это файрвол, созданный специально для «общения» с искусственным интеллектом. AppSec.AIGate анализирует семантику и контекст обращения к модели.

Технически система работает как «умный посредник» — прокси-шлюз, встроенный между пользователем и языковой моделью. В режиме реального времени он проверяет каждый запрос, направленный к LLM, и каждый ответ, полученный от неё. Если пользователь пытается заставить модель нарушить корпоративную политику или если модель в ответ на, казалось бы, невинный вопрос вдруг начинает выдавать персональные данные клиентов,— AppSec.AIGate блокирует такое взаимодействие мгновенно, не дожидаясь, пока информация покинет периметр компании.

Отдельного внимания заслуживает встроенный модуль Anti-Evasion, отвечающий за защиту от попыток обхода ограничений. Злоумышленники непрерывно ищут способы «обмануть» встроенные фильтры безопасности модели.

Модуль Anti-Evasion в составе AppSec.AIGate работает как своего рода «детектор лжи»: он распознаёт попытки манипуляции моделью независимо от того, насколько изощрённой была формулировка, и не позволяет злоумышленнику взломать логику нейросети.

В отличие от файрволов для классического ПО, AppSec.AIGate анализирует не сам запрос, а контекст запроса, его семантику. То есть он анализирует, что пользователь хочет получить от модели и что он туда передаёт. Сервис предусматривает набор кастомных правил, настроенных с учётом внутренней политики компании, при которых запрос, содержащий определённые ключевые слова, будет заблокирован.

Отдельный специализированный модуль внутри системы отвечает за обнаружение и маскирование чувствительных данных, которые разработчики нередко неосознанно передают модели в процессе работы — тем самым система предупреждает возможные утечки ещё до того, как они произойдут.

Современные требования к масштабируемости учтены в самой конструкции продукта: решение AppSec Solutions позволяет не только защищать взаимодействие с моделью, но и управлять нагрузкой, поддерживая одновременную работу с несколькими моделями.

Программное обеспечение для защиты LLM-моделей от атак сегодня по праву можно назвать одним из самых ожидаемых продуктов на российском рынке кибербезопасности. Актуальность этой разработки подтверждается конкретными отраслевыми данными: в конце 2025 года Ассоциация ФинТех совместно с ГК Swordfish Security опубликовала совместное исследование, в выборку которого вошли крупнейшие компании финансового сектора. Согласно его результатам, 75% финтех-организаций отдельно выделили утечку конфиденциальных данных как ключевую угрозу безопасности при работе с ИИ.

Эти цифры показывают, что финансовая отрасль как один из наиболее чувствительных к утечкам данных секторов экономики особенно остро нуждалась в инструментах защиты ИИ-систем. Эпоха искусственного интеллекта в России уже наступила, и рынок кибербезопасности уже предлагает зрелые решения для его защиты.

АТАКИ НА ИЗВЛЕЧЕНИЕ ОБУЧАЮЩИХ ДАННЫХ

ПОЧЕМУ НЕЙРОСЕТЬ МОЖЕТ «ПРОБОЛТАТЬСЯ» О ТОМ, ЧЕМУ ЕЁ УЧИЛИ

Анна ДАНИЛОВА
руководитель направления статического анализа безопасности приложений Swordfish Security

Денис ДАНИЛОВ
руководитель группы обеспечения безопасности приложений Swordfish Security

Компании всё активнее внедряют модели машинного обучения (ML) и большие языковые модели (LLM), обучая их на внутренних данных — от клиентских баз до переписок и коммерческих договоров. При этом мало кто задумывается о том, что обученная модель — это не «чёрный ящик», скрывающий свои источники, а полноценный актив, из которого при определённых условиях можно восстановить фрагменты исходных данных. Разберёмся, как это работает и что с этим делать.

Общее представление о машинном обучении звучит примерно так: модель обобщает данные и изучает закономерности, а сами исходные примеры бесследно «размазываются» в миллионах параметров. На практике это не совсем так. Если модель переобучена (overfitting), обучена на небольшом или недостаточно разнообразном датасете, либо в её обучающей выборке есть часто повторяющиеся или уникальные записи, она может буквально «запомнить» их и воспроизвести при определённых условиях.

Эта угроза для классических ML-моделей (классификаторов, рекомендательных систем) известна научному сообществу уже более десяти лет. Она получила название Membership Inference Attack (атака на определение принадлежности данных к обучающей выборке). Злоумышленник, не имея прямого доступа к датасету, подаёт модели тестовые примеры, а затем по её поведению (уверенности предсказания, вероятностям на выходе) определяет, участвовала ли конкретная запись в обучении. Казалось бы, безобидный приём. Но если модель обучалась на данных пациентов больницы, сам факт присутствия конкретного человека в обучающей выборке уже технически является утечкой чувствительной информации.

С приходом больших языковых моделей проблема вышла на новый уровень. LLM обучаются на огромных корпусах текстов, включающих код, документы, переписку, а нередко и данные, случайно попавшие в выборку без разрешения правообладателей и должной фильтрации. Исследователи неоднократно демонстрировали, что при определённых запросах модель способна дословно воспроизвести фрагменты обучающих текстов: персональные данные, номера телефонов, приватный код или отрывки авторских произведений. Это и есть атака на извлечение обучающих данных (Training Data Extraction).

Отдельно стоит упомянуть Model Extraction (иногда её называют Model Stealing). Эта атака близка по механике, но направлена не на данные, а на саму модель как продукт. Злоумышленник массово опрашивает публичный API целевой модели и на основе пар «запрос — ответ» обучает собственную модель-клон, которая с приемлемой точностью воспроизводит поведение оригинала. Формально никакие данные при этом не «крадутся» — присваивается результат многолетней работы команды аналитиков, стоимость сбора и разметки датасета, вычислительные ресурсы, потраченные на обучение, и, по сути, вся конкурентная ценность модели как актива. Самым ярким примером является создание дистиллятов на основе моделей от Anthropic.

Еще один вектор риска возникает, когда компания дообучает базовую модель на собственных закрытых данных, а затем предоставляет к ней доступ через API или чат-интерфейс подрядчикам, партнерам или в виде публичного продукта. В этом случае поверхность атаки шире: злоумышленник может попытаться извлечь не только данные исходной «базовой» модели, но и специфичные для компании сведения, добавленные на этапе дообучения, которые зачастую оказываются гораздо более чувствительными.

Важно понимать, что для большинства подобных атак не требуется доступ к весам модели или инфраструктуре обучения. Достаточно легально подключиться к готовому API или чат-интерфейсу, то есть барьер входа для злоумышленника минимален.

Извлечение обучающих данных нельзя рассматривать исключительно как техническую особенность нейронных сетей. Для организации атака может иметь те же последствия, что и компрометация информационной системы. Помимо раскрытия персональных сведений возможна утечка

данных компании. Внутренние документы, код и переписки, использованные для дообучения корпоративного ассистента, потенциально могут быть извлечены пользователем через тщательно сформулированные запросы. А если модель обучал внешний подрядчик и использовал для этого закрытые данные заказчика, вопрос о том, кто несёт ответственность за возможную утечку через готовую модель, требует отдельной проработки в договоре и соглашении о неразглашении (Non-Disclosure Agreement, NDA). Очевидно, что само по себе NDA от подобной атаки не защищает.

При этом полностью исключить теоретическую возможность подобных атак сложно, но риск можно снизить до приемлемого уровня организационными и техническими мерами.

На этапе подготовки данных:

- проводить деперсонализацию и минимизацию данных перед обучением (не включать в датасет данные, без которых модель может обойтись);
- избегать дублирования уникальных и редких записей в обучающей выборке, так как именно их чаще всего запоминает модель;
- вести учёт источников информации и их классификацию по чувствительности, чтобы понимать, какие риски несёт конкретный датасет;
- по возможности использовать синтетические данные там, где это не критично для качества модели.

На этапе обучения:

- применять техники Differential Privacy — добавление контролируемого шума в процесс обучения. Это математически ограничивает влияние отдельной записи на итоговую модель и снижает риск как атак на членство, так и извлечения данных;
- контролировать степень переобучения модели — ранняя остановка обучения снижает склонность модели «запоминать» отдельные примеры.

На этапе эксплуатации:

- ограничивать доступ к модели: закрытые API, аутентификация, лимиты на количество и частоту запросов — большинство атак на этом этапе требует множества итераций;
- внедрять мониторинг аномального поведения пользователей API. Резкие всплески похожих по структуре запросов или множественные попытки «продолжить» фрагменты текста могут быть признаком атаки на извлечение;
- фильтровать и модерировать ответы модели на предмет случайного раскрытия персональных данных, номеров карт, ключей и прочих чувствительных паттернов (по аналогии с Data Loss Prevention (DLP)-системами, но применительно к выходу модели);
- регулярно проводить Red Team тестирование собственных моделей на предмет извлечения обучающих данных (так же, как пентест инфраструктуры).

На организационном уровне:

- закреплять в договорах с подрядчиками, обучающими или дообучающими модели на данных заказчика, требования по их безопасной обработке и порядок ответственности за утечки;
- включать оценку рисков извлечения данных в процесс приёмки ML/LLM-решений наравне с традиционными ИБ-проверками (статический анализ (Static Application Security Testing, SAST), динамический анализ (Dynamic Application Security Testing, DAST) безопасности приложений, анализ состава ПО (Software Composition Analysis, SCA)) — по аналогии с подходом, применяемым для обычного ПО.

Атаки на извлечение обучающих данных — пока не самая обсуждаемая, но вполне реальная угроза, которая становится тем актуальнее, чем активнее бизнес использует собственные сведения для обучения и дообучения моделей. Обученная модель — не «чёрный ящик» в смысле безопасности, а ещё один носитель информации, который нуждается в оценке рисков, контроле доступа и мониторинге так же, как и любая база данных или файловое хранилище. Компаниям, работающим с ML/LLM, стоит уже сейчас включать этот риск в модель угроз своих ИИ-систем до того, как ее продемонстрирует кто-то извне.

ОТЯГОЩЁННОЕ ИНТЕЛЛЕКТОМ

СЛОЖНОСТИ ИСПЫТАТЕЛЬНОЙ ЛАБОРАТОРИИ ПРИ ОЦЕНКЕ ЗАЩИЩЁННОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, РАЗРАБОТАННОГО С ИСПОЛЬЗОВАНИЕМ ИИ

Егор РЫХЛОВ
эксперт Центра оценки соответствия и тестирования АО «НПО «Эшелон»

Данил НОВИКОВ
эксперт Центра оценки соответствия и тестирования АО «НПО «Эшелон»

Практика разработки ПО всё больше смещается в сторону кода, написанного с помощью ИИ-ассистентов. В статье показано, как систематические ошибки моделей проявляются в трёх базовых видах проверок безопасности – поиске секретов, статическом и композиционном анализе и почему для каждого из них применение ИИ создаёт свою специфическую сложность. Отдельно рассмотрены меры снижения рисков со стороны разработки и со стороны лаборатории.

ВВЕДЕНИЕ

Сегодня практика разработки стремительно меняется под влиянием ИИ-ассистентов и распространения вайбкодинга. По данным исследования «Т-технологий»[1], 58% опрошенных инженеров уже пишут код с использованием ИИ, а доля такого кода в крупных организациях достигает 20–30% от общего объёма. Число пользователей одного лишь GitHub Copilot превысило 26 миллионов. Практическое подтверждение связанных с этим рисков даёт исследование компании Escape.tech, проанализировавшей свыше 1400 продакшн-приложений, созданных преимущественно с помощью ИИ, в котором 65% приложений содержали проблемы безопасности, 58% – как минимум одну критичную уязвимость, а в совокупности было обнаружено более 400 раскрытых секретов и 175 случаев раскрытия персональных данных[2].

Такая тенденция неизбежно сказывается на качестве кода, в первую очередь с точки зрения безопасности. В условиях повсеместного внедрения ИИ-ассистентов стандартные методы проверки безопасности, такие как поиск секретов, статический и композиционный анализ, начинают работать с новыми паттернами ошибок, и каждый из них заслуживает отдельного рассмотрения.

ПОИСК СЕКРЕТОВ В ИСХОДНОМ КОДЕ

Поиск секретов представляет собой вид работ, направленный на выявление в исходном коде и истории коммитов «хардкоженных» учётных данных, ключей доступа к API, токенов и паролей, как правило, с использованием сигнатурного и энтропийного анализа строк средствами наподобие gitleaks или TruffleHog.

Использование ИИ увеличивает количество секретов в коде по двум независимым причинам. Во-первых, языковая модель способна воспроизводить реальные учётные данные, встреченные в обучающей выборке. Так, 1/3 предложений автодополнения GitHub Copilot содержала валидные по формальным признакам секреты, часть из которых оказалась действующими ключами доступа[3]. Во-вторых, при генерации примеров конфигурации или кода подключения к внешним сервисам модель по умолчанию склонна подставлять значение секрета непосредственно в код, а не использовать переменные окружения или менеджер секретов, поскольку такой упрощённый вариант чаще встречается в обучающих дан-

[1] https://www.vedomosti.ru/technologies/industries_and_markets/articles/2026/06/30/1210109-rinok-ii

[2] https://escape.tech/blog/methodology-how-we-discovered-vulnerabilities-apps-built-with-vibe-coding/

[3] Your Code Secret Belongs to Me: Neural Code Completion Tools Can Memorize Hard-Coded Credentials. Y Huang, Y Li, W Wu, J Zhang, MR Lyu. Proceedings of the ACM on Software Engineering, 2024

ных, и не предлагает более безопасную альтернативу, если разработчик не запросил её явно.

Совокупный эффект этих механизмов подтверждается отраслевой статистикой. Репозитории с включённым Copilot содержат утечки секретов с частотой 6,4% против 4,6% в среднем по всем публичным репозиториям, а частота утечек в коммитах с участием ИИ примерно вдвое выше базовой: 3,2% против 1,5%[4]. Отдельно отмечен рост на 81% за год числа утечек ключей доступа к самим ИИ-сервисам, а в конфигурационных файлах для протокола MCP обнаружено свыше 24 тысяч уникальных секретов, из которых более двух тысяч оказались действующими.

Для испытательной лаборатории данная динамика не меняет принципиальную работоспособность инструментов поиска секретов, поскольку хардкоженная строка обнаруживается независимо от того, кто её написал. Сложность заключается в ином. Резко растёт объём находок, подлежащих ручному триажу при том же согласованном бюджете времени, а появление новых типов секретов, в первую очередь ключей ИИ-сервисов и конфигураций MCP, требует отдельного расширения используемых сигнатур детектора, поскольку данные форматы могут не входить в стандартный набор правил на момент его последнего обновления.

Репозитории с включённым Copilot содержат утечки секретов с частотой 6,4% против 4,6% в среднем по всем публичным репозиториям, а частота утечек в коммитах с участием ИИ примерно вдвое выше базовой: 3,2% против 1,5%

СТАТИЧЕСКИЙ АНАЛИЗ КОДА

Статический анализ исходного кода (SAST) представляет собой вид работ, направленный на поиск дефектов в коде без его фактического выполнения, включая поиск небезопасных конструкций, таких как SQL-инъекции, межсайтовый скриптинг, небезопасная десериализация и слабая криптография, по заранее определённым правилам.

В исследовании[5], охватившем свыше ста языковых моделей и более восьмидесяти тестовых задач на языках Java, Python, C# и JavaScript, при выборе между безопасным и небезопасным способом реализации функции модель в среднем выбирала небезопасный вариант почти в половине случаев, критичные недостатки были выявлены в 45% протестированных случаев. Наиболее уязвимым языком оказалась Java с долей неудачных проверок свыше 70%, а для межсайтового скриптинга доля неудачных проверок достигла 86%, что примерно в 2,74 раза выше частоты аналогичного недостатка в коде, написанном человеком. Существенно, что отчёт не выявил связи между размером и датой выпуска модели и качеством генерируемого кода с точки зрения безопасности, а обновление отчёта весной 2026 года подтвердило, что доля успешных проверок остаётся на стабильном уровне около 55%, то есть проблема носит структурный, а не временный характер.

Для испытательной лаборатории конкретный небезопасный паттерн остаётся тем же самым паттерном независимо от того, кем он написан, и обнаруживается тем же набором правил SAST. Сложность здесь схожа с описанной для поиска секретов: объём находок на тот же объём кода заметно растёт, что увеличивает трудозатраты на присвоение критичности и подготовку описания каждой находки с указанием Common Weakness Enumeration (CWE), класса типовых дефектов безопасности. Дополнительно проявляется системный характер предпочтений модели: конкретные классы уязвимостей повторяются с высокой и предсказуемой частотой, что в перспективе может использоваться для приоритизации правил анализа, однако требует целенаправленного пересмотра используемого набора сигнатур.

КОМПОЗИЦИОННЫЙ АНАЛИЗ

Композиционный анализ (SCA) представляет собой вид работ, основанный на формировании перечня зависимостей программного обеспечения и выявлении в них известных уязвимостей и иных недостатков, как правило на основе SBOM-файла и сверки его содержимого с базами данных уязвимостей.

Использование ИИ формирует для данного вида проверки риск, не сводящийся к уязвимостям в существующих компонентах. Явление, получившее название slopsquatting, состоит в том, что модель при генерации кода предлагает правдоподобное, но несуществующее имя пакета (см. рис. 1). Злоумышленник, заранее зарегистрировавший такое имя с вредоносным содержимым, получает возможность скомпрометировать проект в момент установки зависимости. Согласно исследованию[6] на выборке из 576 тысяч фрагментов кода на Python и JavaScript, сгенерированных шестнадцатью распространёнными моделями, частота галлюцинированных зависимостей сильно различалась между моделями, от менее 5% у GPT-4 и GPT-4 Turbo до более 15% у DeepSeek и свыше 25% у Code Llama 7B, при

[4] https://www.gitguardian.com/state-of-secrets-sprawl-report-2026

[5] https://www.veracode.com/resources/analyst-reports/2025-genai-code-security-report/

[6] Spracklen J., Wijewickrama R., Sakib A. H. M. N., Maiti A., Viswanath B., Jadliwala M. We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs // Proceedings of the USENIX Security Symposium. 2025. arXiv:2406.10279

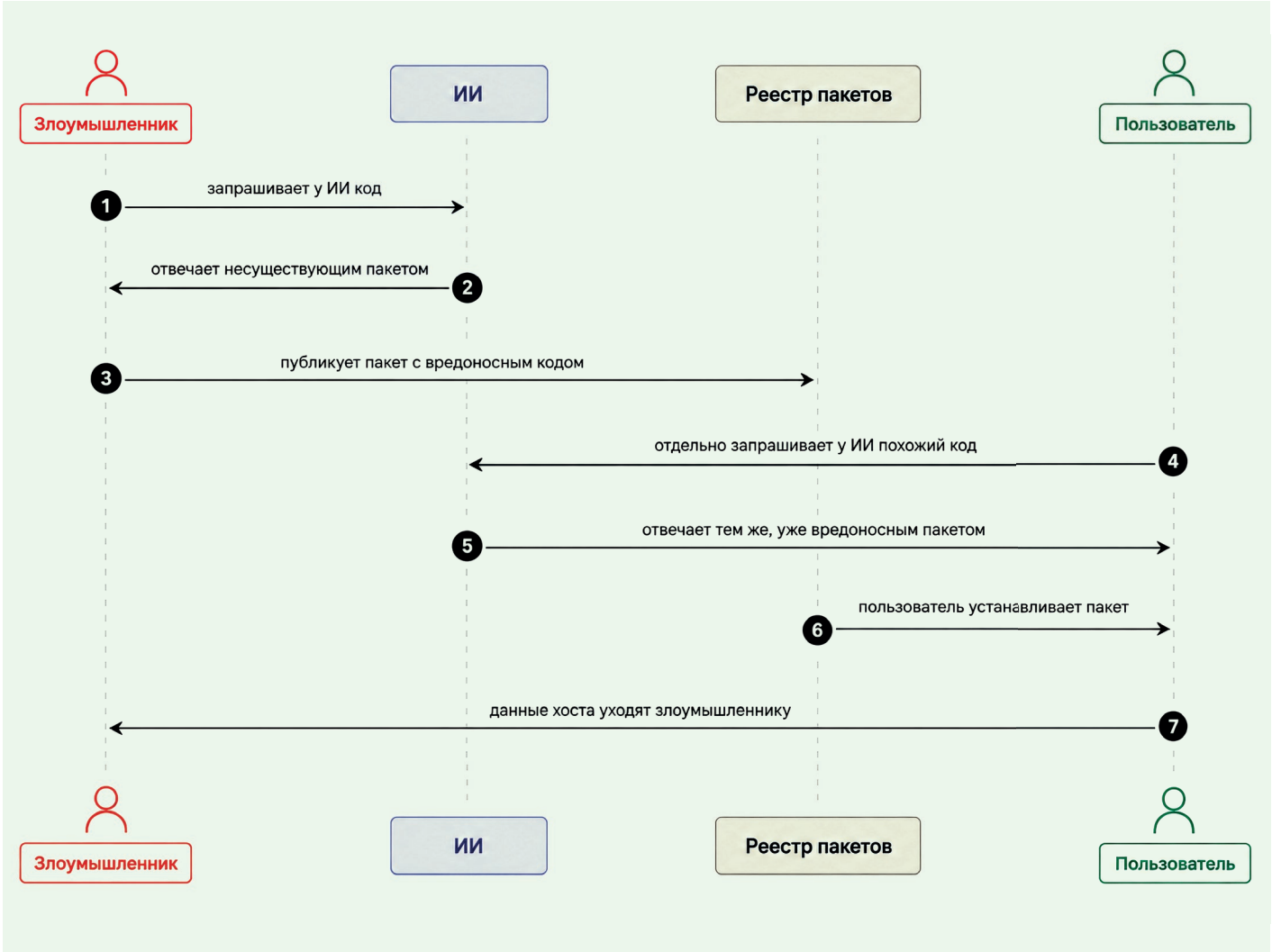

Рисунок 1. Схема атаки slopsquatting

этом код на JavaScript содержал вымышленные зависимости заметно чаще, чем код на Python (21% против 16%). Отдельного внимания заслуживает воспроизводимость эффекта: при повторной генерации одного и того же запроса 43% галлюцинированных названий повторялись при каждом из десяти прогонов, что делает такие названия предсказуемыми и пригодными для целенаправленной регистрации злоумышленником заранее.

Отдельно от галлюцинации названий существует смежная проблема, при которой модель рекомендует реально существующий, но устаревший или уязвимый пакет. Знания модели ограничены датой отсечения обучающих данных, поэтому она не осведомлена об уязвимостях, опубликованных позднее, и продолжает предлагать версию, для которой уже вышел патч. При этом выбор смещён в худшую сторону: среди уязвимых версий, рекомендованных моделями, критичные и высокие по опасности типовые уязвимости (Common Vulnerabilities and Exposures, CVE) составляют 63–75%, тогда как в общем массиве известных уязвимостей их доля около 29%[7].

Для испытательной лаборатории риск галлюцинации названий затрагивает саму методологическую основу композиционного анализа. Инструменты, такие как OWASP Dependency Track, Trivy, Grype и другие, сверяют состав всех компонентов (пакетов, библиотек), используемых в разрабатываемом ПО или необходимых для его работы (Software Bill of Materials, SBOM), с базами данных известных уязвимостей в существующих пакетах, но не отвечают на вопрос, существует ли пакет вообще и заслуживает ли он доверия. Это означает, что типовая методика композиционного анализа должна быть дополнена отдельным шагом верификации фактического существования и репутации каждой зависимости в официальном реестре, чего процесс SCA на основе одного лишь SBOM-файла не предусматривает.

Риск устаревших и уязвимых версий, в отличие от галлюцинации названий, не выходит за рамки классической методики. Инструменты SCA по определению сопоставляют версию пакета с базой известных CVE и обнаружат такую уязвимость тем же способом, что и внесённую человеком. Сложность здесь не методологическая, а связана со сме-

[7] Wang C. et al. Correct Code, Vulnerable Dependencies: A Large Scale Measurement Study of LLM Specified Library Versions. 2026. arXiv:2605.06279

Вид проверки	Что меняется в связи с использованием ИИ	Сложность для ИЛ
Поиск секретов	Модель воспроизводит реальные учётные данные и по умолчанию задаёт их прямо в код вместо переменных окружения	Резко возрастает количество таких находок на тот же объём кода; Сигнатуры детектора нужно отдельно дополнять под новые типы секретов
Статический анализ кода	Модель отдаёт предпочтение небезопасной и (или) устаревшей конструкции кода	Резко возрастает количество таких находок на тот же объём кода; Инструменты статического анализа не умеют различать, какая часть кода написана человеком, а какая сгенерирована ИИ
Композиционный анализ	Модель предлагает несуществующие зависимости (slopsquatting) либо реальные версии с непропорционально высокой долей критичных CVE, актуальных уже после даты отсечения обучающих датасетов	Для галлюцинаций классическая проверка по базам известных уязвимостей не покрывает риск и требует отдельного шага верификации существования пакета; Для устаревших версий инструмент срабатывает штатно, но частота таких срабатываний растёт, что требует немедленного реагирования

Таблица 1. Влияние использования ИИ разработчиком на три вида проверок и сложности для испытательной лаборатории

щением распределения находок, поскольку модель непропорционально часто предлагает версии именно с критичными и высокими по степени опасности уязвимостями, что увеличивает долю находок, требующих немедленной эскалации, и соответствующую нагрузку на приоритизацию в рамках уже упомянутого возросшего общего объёма.

Обобщённая картина по трём рассмотренным видам проверок представлена в таблице 1.

НАПРАВЛЕНИЯ СНИЖЕНИЯ РИСКОВ

Рассмотренные сложности не означают неприменимость ИИ при разработке. Они лишь демонстрируют, что такое использование должно сопровождаться дополнительными мерами контроля.

Со стороны разработки должна проводиться обязательная верификация сгенерированного кода. На практике она сводится к применению взаимодополняющих мер из лучших практик жизненного цикла разработки безопасного ПО (Secure Software Development Life Cycle, SSDLC): безопасному хранению секретов в специализированных решениях и переменных окружения вместо их размещения в коде, проверке каждой предложенной моделью зависимости на факт существования и репутацию до установки, встраиванию инструментов сканирования секретов и композиционного анализа в конвейер Continuous Integration / Continuous Delivery / Deployment (CI/CD), следованию правилам безопасного кодирования[8] и многому другому. Важным элементом цикла безопасной разработки ПО остаётся и динамический анализ кода, в том числе фаззинг-тестирование, позволяющее выявлять уязвимости, недоступные статическим методам[9].

Со стороны проверяющего адаптация методики сводится к нескольким направлениям. Во-первых, это регулярное расширение набора используемого инструментария, а также сигнатур детекторов под новые типы потенциальных проблем безопасности. Во-вторых, дополнение методики композиционного анализа шагом верификации существования и репутации зависимостей. В-третьих, приоритизация правил статического анализа с учётом предсказуемо повторяющихся классов уязвимостей и обоснованный выбор инструментов под конкретную задачу. Также немаловажно закладывать дополнительное время триажа с поправкой на возросший объём находок.

В перспективе доля ИИ-сгенерированного кода будет расти, а вместе с ней и нагрузка на анализ уязвимостей, и значимость выявленных методических пробелов. Без адаптации инструментов и методик это ведёт к увеличению доли пропущенных проблем при неизменном бюджете проверки и расширению поверхности атак на цепочку поставок.

ЗАКЛЮЧЕНИЕ

Использование искусственного интеллекта разработчиком отражается на всех трёх рассмотренных видах проверок, но не одинаковым образом. Общий вывод состоит в том, что рост доли ИИ-сгенерированного кода не снижает применимость инструментария испытательной лаборатории, но существенно увеличивает нагрузку на неё и обнажает пробелы в методике, изначально рассчитанной на код, написанный человеком. Дальнейшая работа в этом направлении может быть связана с обновлением типовых программ и методик испытаний с явным учётом рассмотренных рисков.

[8] Как избежать ошибок при безопасной разработке программного обеспечения / В. В. Вареница, А. С. Марков, В. Л. Цирлов и др.— М.: Квантмедиа, 2025.— 344 с. EDN: ZFHEHG

[9] Арустамян С. С., Антипов И. С. Интеллектуальные методы фаззинг-тестирования в рамках цикла безопасной разработки программ.