Исследование RAG-подходов для обнаружения уязвимостей с использованием больших языковых моделей

Михайлишин А. С.
ФГАОУ ВО «Южный федеральный университет»,
Институт математики, механики и компьютерных наук им. И. И. Воровича,
г. Ростов-на-Дону
E-mail: mikhailishin@sfedu.ru

Большие языковые модели (LLM) стали неотъемлемыми инструментами в разработке программного обеспечения, демонстрируя высокие возможности в автоматической генерации кода и отладке. Модели генерации кода, такие как ChatGPT, Claude, Gemini, Grok и т.д., сегодня широко используются разработчиками. По Stack Overflow Developer Survey 2025 [1] 84% опрошенных уже используют ИИ-инструменты в разработке или планируют использовать их. Среди профессиональных разработчиков 50.6% используют ИИ ежедневно. По GitHub Blog [2] в октябре 2025 GitHub писал о 20 млн пользователей в экосистеме Copilot, о 3+ млрд принятых подсказок кода, участие агента в 1,2 млн pull-запросах и о том, что Copilot Autofix помог исправить более миллиона ошибок и уязвимостей за 2025 год.

Вместе с ускорением разработки программного обеспечения, распространенность программных уязвимостей одновременно выросла беспрецедентными темпами. Резкий рост зарегистрированных уязвимостей [3] в 2024 - более 40 тысяч (+38% относительно 2023 года) продолжился в 2025 и превысил 48 тысяч (+20% относительно 2024 года). До широкого распространения ИИ-инструментов статистика регистраций составляла около 20 тысяч за 2021 год со средним ростом +8,3% в год. Однако и проблема эксплуатации давно известных уязвимостей не теряет актуальности. По данным GreyNoise [4], 40% CVE, эксплуатирующихся в 2024 году, были раскрыты не позднее 2020 года, а Cisco Talos отмечает [5], что в 2025 году 39% уязвимостей, добавленных в KEV как активно эксплуатируемые, относились к CVE-2024 и более ранних лет.

LLM не получают автоматически знания об уязвимостях, обнаруженных после даты отсечения их знаний — точки, после которой в обучение больше не включаются новые данные. Следовательно, они могут непреднамеренно предлагать код, содержащий известные уязвимости безопасности, поскольку соответствующие CVE записи отсутствовали в обучающих данных.

На практике для поиска уязвимостей сегодня применяются несколько основных подходов. Первый класс методов ориентирован на поиск уже известных проблем в зависимостях и компонентах проекта, а именно сканирование зависимостей, контейнеров и образов. Такие методы хорошо работают, когда уязвимость уже зарегистрирована в базах CVE, GHSA, OSV и может быть сопоставлена с конкретной версией библиотеки или пакета в проекте, однако она не решает вопрос уязвимостей в коде проекта.

Второй класс методов связан со статическим анализом исходного кода: это SAST, поиск по шаблонам, taint analysis, data-flow analysis, межпроцедурный анализ и символьное исполнение. Эти подходы позволяют выявлять не только известные CVE, но и целые классы уязвимостей, например SQL injection, XSS, SSRF, path traversal, небезопасную десериализацию. Но на больших репозиториях у них возникают ограничения: высокий уровень ложноположительных срабатываний, слабый учет контекста проекта и сложность анализа цепочек, проходящих через несколько файлов, модулей и уровней абстракции.

Здесь возникает интерес к применению больших языковых моделей. LLM способны интерпретировать код семантически, связывать разрозненные участки программы, объяснять подозрительные фрагменты и помогать в анализе сложной логики. Одним из наиболее перспективных подходов здесь является RAG - retrieval-augmented generation, то есть генерация с опорой на внешний контекст, который извлекается из самого проекта. Для задач программного кода это особенно важно, потому что уязвимость редко локализуется в одном фрагменте: часто необходимо связать точку ввода данных, промежуточную обработку, вызовы библиотек и потенциально опасную операцию.

Сегодня для работы с репозиториями используются несколько классов RAG-подходов. Во-первых, это векторный поиск по эмбеддингам, когда кодовые фрагменты индексируются и затем извлекаются по семантическому сходству. Во-вторых, гибридные подходы, которые сочетают семантический поиск с ключевыми словами, символами, импортами и структурой проекта. В-третьих, подходы с учетом графовой структуры репозитория: графов вызовов, импортов, зависимостей между модулями и других связей. Наконец, появляются агентные системы repo-level understanding, в которых модель не просто получает фрагменты кода, а последовательно исследует проект, уточняет гипотезы и собирает контекст по шагам. Примерами таких работ являются RepoMaster, использующий графы вызовов, модульных зависимостей и иерархии кода [6], GraphCodeAgent, опирающийся на двойное графовое представление репозитория [7], а также Repository Intelligence Graph, где для навигации по проекту строится детерминированная архитектурная карта [8]. Отдельно можно отметить ELITE, предлагающий итеративное извлечение контекста без классического embedding-based retrieval [9].

Однако в security-сценарии даже этого часто недостаточно. Обычный RAG хорошо находит похожие по смыслу участки кода, но не всегда позволяет надежно восстановить цепочку распространения данных от source к sink. Для поиска уязвимостей важно не только найти похожий фрагмент, но и понять, откуда пришли пользовательские данные, проходили ли они санитизацию, через какие функции и модули они распространялись и в какую опасную точку в итоге попали. Поэтому для задач обнаружения уязвимостей RAG-подсистему необходимо расширять за счет структурного и security-ориентированного контекста.

В своем исследовании я собираюсь исследовать именно такие расширения. Необходимо дополнить базовый семантический поиск многоуровневой индексацией проекта:

- индексация на уровне файлов, функций, классов, импортов, внешних вызовов и графа зависимостей
- трассировка движения пользовательских данных в проекте от входа до места применения действий связанных с ними

- дополнительный индекс маркеров-безопасности для функций - обращения к exec, файловые операции, сетевые запросы и другие потенциально опасные паттерны.

Далее предполагается сравнить такую систему с традиционными вариантами RAG на специализированных бенчмарках. В качестве основы для оценки могут использоваться SEC-bench, ориентированный на реальные задачи безопасности для LLM-агентов [10], JitVul, предназначенный для практического обнаружения уязвимостей в репозиториях [11], а также CVE-Bench, полезный как источник реалистичных security-сценариев [12]. Сравнение можно проводить по полноте найденного контекста, точности локализации уязвимости, доле нерелевантных фрагментов, устойчивости на больших репозиториях и вычислительным затратам. Это позволит определить, какие механизмы retrieval дают наибольший прирост именно для security-анализа и для каких классов уязвимостей они наиболее полезны.

Литература

1. «AI | 2025 Stack Overflow Developer Survey». б. д. Просмотрено 16 март 2026 г. https://survey.stackoverflow.co/2025/ai/.
2. «Copilot: Faster, smarter, and built for how you work now - The GitHub Blog». б. д. Просмотрено 16 март 2026 г. https://github.blog/ai-and-ml/github-copilot/copilot-faster-smarter-and-built-for-how-you-work-now/.
3. «Browse CVE Vulnerabilities by Date». б. д. Просмотрено 16 март 2026 г. https://www.cvedetails.com/browse-by-date.php.
4. «GreyNoise 2025 Mass Internet Exploitation Report». б. д. Просмотрено 16 март 2026 г. https://www.greynoise.io/resources/2025-mass-internet-exploitation-report.
5. Cisco Talos Blog. 2026. «Patch, Track, Repeat: The 2025 CVE Retrospective». март 5.

https://blog.talosintelligence.com/patch-track-repeat-the-2025-cve-retrospective/.

6. Wang, Huacan, Ziyi Ni, Shuo Zhang, и др. 2025. «RepoMaster: Autonomous Exploration and Understanding of GitHub Repositories for Complex Task Solving». arXiv:2505.21577. Preprint, arXiv, август 25. https://doi.org/10.48550/arXiv.2505.21577.
7. Li, Jia, Xianjie Shi, Kechi Zhang, и др. 2025. «GraphCodeAgent: Dual Graph-Guided LLM Agent for Retrieval-Augmented Repo-Level Code Generation». arXiv:2504.10046. Preprint, arXiv, ноябрь 18. https://doi.org/10.48550/arXiv.2504.10046.
8. Cherny-Shahar, Tsvi, и Amiram Yehudai. 2026. «Repository Intelligence Graph: Deterministic Architectural Map for LLM Code Assistants». arXiv:2601.10112. Preprint, arXiv, январь 15. https://doi.org/10.48550/arXiv.2601.10112.
9. Wang, Zhangyu, Siyuan Gao, Rong Zhou, Hao Wang, и Li Ning. 2025. «ELITE: Embedding-Less retrieval with Iterative Text Exploration». arXiv:2505.11908. Preprint, arXiv, май 17. https://doi.org/10.48550/arXiv.2505.11908.
10. Lee, Hwiwon, Ziqi Zhang, Hanxiao Lu, и Lingming Zhang. 2025. «SEC-bench: Automated Benchmarking of LLM Agents on Real-World Software Security Tasks». arXiv:2506.11791. Preprint, arXiv, октябрь 22. https://doi.org/10.48550/arXiv.2506.11791.
11. Yildiz, Alperen, Sin G. Teo, Yiling Lou, Yebo Feng, Chong Wang, и Dinil M. Divakaran. 2025. «Benchmarking LLMs and LLM-based Agents in Practical Vulnerability Detection for Code Repositories». arXiv:2503.03586. Preprint, arXiv, март 18. https://doi.org/10.48550/arXiv.2503.03586.
12. Zhu, Yuxuan, Antony Kellermann, Dylan Bowman, и др. 2025. «CVE-Bench: A Benchmark for AI Agents' Ability to Exploit Real-World Web Application Vulnerabilities». arXiv:2503.17332. Preprint, arXiv, июнь 24. https://doi.org/10.48550/arXiv.2503.17332.