

Применение ИИ для разработки безопасных скриптов администрирования Linux

Саяпин Никита Анатольевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Аннотация

В статье рассматривается процесс автоматизации разработки скриптов для администрирования операционных систем семейства Linux с применением больших языковых моделей на примере GPT-5.2. Оценивается способность нейросети генерировать отказоустойчивый и защищенный код для выполнения рутинных задач системного администрирования. Демонстрируется влияние уровня детализации пользовательского запроса на внедрение практик безопасного программирования, таких как валидация ввода, обработка исключительных ситуаций и безопасная работа с файловой системой.

Ключевые слова: GPT-5.2, системное администрирование, Linux, bash, Python, безопасный код, автоматизация рутины, генерация кода, DevSecOps.

Application of artificial intelligence for developing secure linux administration scripts

Sayapin Nikita Anatolyevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article explores the process of automating the development of scripts for administering Linux operating systems using large language models, exemplified by GPT-5.2. The ability of the neural network to generate fault-tolerant and secure code for performing routine system administration tasks is evaluated. The impact of the user prompt's detail level on the implementation of secure programming practices, such as input validation, exception handling, and secure file system operations, is demonstrated.

Keywords: GPT-5.2, system administration, Linux, bash, Python, secure code, routine automation, code generation, DevSecOps.

1. Введение

1.1 Актуальность

Повседневное администрирование серверных инфраструктур на базе Linux неразрывно связано с написанием скриптов на Bash или Python для автоматизации таких задач, как резервное копирование, ротация логов,

мониторинг состояния системы и управление учетными записями. Ошибки при написании подобных скриптов зачастую становятся причиной серьезных инцидентов информационной безопасности. Некорректная обработка пользовательского ввода может привести к уязвимостям внедрения команд (Command Injection), отсутствие проверок прав доступа — к повышению привилегий, а небезопасная работа с временными файлами — к атакам типа Race Condition. В условиях парадигмы DevSecOps безопасность должна внедряться на самых ранних этапах разработки внутренних инструментов. Использование больших языковых моделей, таких как GPT-5.2, представляет собой перспективный подход к решению этой проблемы. Модель может не только сгенерировать синтаксически верный код, но и интегрировать в него механизмы защиты, опираясь на лучшие практики написания безопасных сценариев. Это позволяет системным администраторам и специалистам по сетям значительно сократить время на написание и отладку инфраструктурного кода, минимизируя влияние человеческого фактора.

1.2 Обзор исследований

А.К. Ковалёв и А.И. Панов исследуют применение предобученных языковых моделей в задачах автоматизированного управления и воплощенного искусственного интеллекта, доказав их эффективность в интерпретации сложных инструкций [1]. И.Р. Бухтияров и А.Б. Шакиров доказали эффективности применения модели Unixcoder для автоматического обнаружения уязвимостей в исходном коде программ и сравнения с классическими трансформерными моделями [2]. А.В. Куртукова и А.С.Романов проводили анализ искусственно сгенерированных исходных кодов для разграничения машинного и человеческого авторства [3]. А.С.Кечеджиев и О.Л. Цветкова предлагают алгоритм разработки и отладки методики, эффективной при использовании искусственного интеллекта в рассматриваемой сфере [4]. Н.В. Евглевская и А.А. Казанцева предложили методику повышения защищенности LLM-сервисов от атак типа Prompt Injection [5].

1.3Цель исследования

Целью исследования является оценка возможностей модели GPT-5.2 в автоматической генерации безопасных и отказоустойчивых скриптов администрирования Linux по естественно-языковому описанию, а также анализ зависимости защищенности итогового кода от степени детализации входного промпта.

2 Материалы и методы

В ходе исследования применяется языковая модель GPT-5.2 для генерации bash- и Python-скриптов, решающих типовые задачи системного администрирования. Рассматриваются три сценария создания инструментов: резервное копирование критичных данных, мониторинг системных журналов на предмет подозрительной активности и управление привилегированными

пользователями. Для каждого сценария формируются текстовые запросы (промпты) различной степени детализации. Минимальные запросы содержат лишь указание конечной цели. Средние запросы дополняются базовыми требованиями к логированию. Высоко детализированные запросы включают строгие инструкции по валидации входных параметров, обработке системных сигналов, использованию безопасных временных файлов и обработке кодов возврата утилит. Сгенерированный код проходит проверку на наличие потенциальных уязвимостей с помощью статического анализатора ShellCheck и тестировался на работоспособность в изолированной виртуальной машине под управлением операционной системы Ubuntu 24.04.

3. Результаты

3.1 Генерация скрипта резервного копирования по минимальному описанию

На начальном этапе исследуется способность нейросети генерировать код по общим запросам, имитируя нехватку времени у администратора на подробное техническое задание. Промпт звучал так: «Напиши bash-скрипт для бэкапа папки /var/www в директорию /backup». Модель предоставила синтаксически корректный скрипт, использующий утилиту tar с подстановкой текущей даты в имя архива. Однако анализ показал, что код не содержит проверок на существование исходной директории, не проверяет наличие свободного места на целевом диске и не имеет защиты от прерываний. При возникновении ошибки на этапе архивации скрипт завершает работу без уведомления пользователя, что в условиях реальной эксплуатации недопустимо и может привести к потере данных (Рис. 1).

```
#!/bin/bash
tar -czf /backup/www_backup_$(date +%F).tar.gz /var/www
echo "Backup completed"
```

Рисунок 1 – Результат генерации бэкап-скрипта по минимальному промпту

3.2 Генерация скрипта мониторинга со средней детализацией промпта

На втором этапе в запрос добавлены требования безопасности и логирования. Промпт: «Напиши bash-скрипт, который парсит /var/log/auth.log на наличие неудачных попыток SSH-входа. Скрипт должен отправлять алерт, использовать безопасное временное хранилище и логировать свои ошибки». Модель GPT-5.2 значительно усложнила структуру кода. Добавлена директива set -e для немедленного завершения скрипта при ошибках, а также проверка на запуск от имени суперпользователя. Для хранения промежуточных данных нейросеть применила команду mktemp, что предотвращает атаки через предсказуемые имена временных файлов. Код стал более устойчивым к

внешним воздействиям и пригодным для использования в базовых инфраструктурах (Рис. 2).

```
#!/bin/bash
set -euo pipefail

if [[ "${EUID}" -ne 0 ]]; then
    echo "Error: This script must be run as root" >&2
    exit 1
fi

TMP_FILE=$(mktemp /tmp/ssh_monitor.XXXXXX)
trap 'rm -f "$TMP_FILE"' EXIT

grep "Failed password for" /var/log/auth.log > "$TMP_FILE" || true
if [[ -s "$TMP_FILE" ]]; then
    echo "Alert: Found failed login attempts!"
    cat "$TMP_FILE"
fi
```

Рисунок 2 – Скрипт мониторинга с базовыми элементами безопасности

3.3 Разработка безопасного инструмента управления по детальному промпту

На третьем этапе сформулирован максимально подробный запрос, учитывающий стандарты DevSecOps. Промпт: «Разработай bash-скрипт для добавления нового пользователя в систему Linux. Скрипт принимает имя пользователя как аргумент. Обеспечь строгую валидацию ввода (только латинские буквы и цифры), проверку существования пользователя, генерацию безопасного случайного пароля (не менее 16 символов), настройку прав доступа на домашнюю директорию (700) и принудительную смену пароля при первом входе. Все действия должны логироваться в syslog». Модель сгенерировала код корпоративного уровня. В скрипте реализованы регулярные выражения для валидации входных параметров, предотвращающие инъекции команд. Используется утилита logger для отправки сообщений в системный журнал. Применен инструмент openssl rand для создания криптографически стойкого пароля (Рис. 3).

```
#!/bin/bash
set -euo pipefail

USERNAME="${1:-}"

if [[ ! "$USERNAME" =~ ^[a-z_][a-z0-9_-]{0,31}$ ]]; then
    logger -t UserAddScript "Invalid username format attempted: $USERNAME"
    echo "Error: Invalid username." >&2
    exit 1
fi

if id "$USERNAME" &>/dev/null; then
    echo "User $USERNAME already exists." >&2
    exit 1
fi

PASSWORD=$(openssl rand -base64 16)
useradd -m -s /bin/bash "$USERNAME"
echo "$USERNAME:$PASSWORD" | chpasswd
chage -d 0 "$USERNAME"
chmod 700 "/home/$USERNAME"

logger -t UserAddScript "User $USERNAME successfully created with secure policies."
echo "User $USERNAME created. Temporary password: $PASSWORD"
```

Рисунок 3 – Фрагмент безопасного скрипта администрирования

3.4 Сравнительный анализ результатов

Для структуризации полученных данных проведено сравнение характеристик безопасности сгенерированного кода в зависимости от уровня детализации исходного запроса к модели GPT-5.2. Оценивалось наличие критически важных механизмов защиты в итоговых bash-скриптах (Табл. 1).

Таблица 1 – Влияние уровня детализации промпта на безопасность скриптов

Критерий	Минимальный промпт	Средний промпт	Высокий промпт
Директивы безопасного выполнения (set -e)	Отсутствуют	Присутствуют	Присутствуют (полный набор)
Валидация пользовательского ввода	Отсутствует	Отсутствует	Строгая (RegEx)
Проверка прав суперпользователя	Отсутствует	Присутствует	Присутствует
Безопасная работа с временными файлами	Не используется	mktemp + trap	Полностью изолированно
Обработка исключений и ошибок	Игнорируется	Базовая	Детализированная
Интеграция с системным журналом	Консольный вывод	Перенаправление потоков	syslog (logger)
Общий уровень надежности кода	Низкий	Средний	Высокий

Представленные в таблице данные наглядно показывают, что без явных указаний нейросеть склонна генерировать код, решающий задачу наиболее прямолинейным, но небезопасным способом. По мере усложнения требований и внедрения в промпт концепций безопасности итоговый результат трансформируется из простого набора команд в полноценный отказоустойчивый программный инструмент.

4 Выводы

Результаты исследования подтверждают высокую практическую ценность больших языковых моделей, в частности GPT-5.2, для задач системного администрирования Linux. Модель способна не только писать код, но и внедрять в него механизмы защиты операционной системы, предотвращая типичные уязвимости скриптовых языков. Ключевым фактором успешного применения нейросетей в парадигме DevSecOps является качество формируемого запроса. При подробном описании требований к валидации данных, логированию и отказоустойчивости GPT-5.2 генерирует скрипты, полностью соответствующие профессиональным стандартам безопасности, что позволяет минимизировать риски компрометации инфраструктуры на этапе автоматизации рутинных процессов.

Библиографический список

1. Ковалев А.К., Панов А.И. Применение предобученных больших языковых моделей в задачах воплощенного искусственного интеллекта // Доклады Российской академии наук. Математика, информатика, процессы управления. 2022. Т. 508, № 1. С. 94-99.
2. Бухтияров И. Р., Мусаева Я., Шакиров А. Б. Применение модели Unixcoder для поиска уязвимостей в исходном коде // Наука и бизнес: пути развития. 2025. № 2(164). С. 8-11.
3. Куртукова А.В., Романов А.С. Проблема искусственно сгенерированных исходных кодов в задаче идентификации автора программы // Сборник избранных статей научной сессии ТУСУР. 2022. № 1-2. С. 131-134.
4. Кечеджиев А. С., Цветкова О.Л. Искусственный интеллект в решении задач кибербезопасности // Молодой исследователь Дона. 2023. Т. 8, № 2(41). С. 23-27.
5. Евглевская Н. В., Казанцев А. А. Методика повышения защищенности LLM-сервисов с применением LLM security proxy // Экономика и качество систем связи. 2026. № 1(39). С. 185-193.