DOI 10.37882/2223-2966.2025.10.19

МЕТОДЫ АВТОМАТИЗАЦИИ АУДИТА СМАРТ-КОНТРАКТОВ В СЕТИ ETHEREUM

METHODS FOR AUTOMATING THE AUDIT OF SMART CONTRACTS ON THE ETHEREUM NETWORK

A. Kleimenov

Summary. This work is dedicated to the systematic analysis and development of methods for the automated auditing of smart contracts within the Ethereum blockchain network. The study involved an analysis of key vulnerabilities inherent in smart contract code and systematized the limitations of traditional, manual auditing methods in the face of exponential growth in the complexity and number of decentralized applications. A multi-level classification of automated methods, encompassing static analysis, dynamic analysis, and formal verification, is proposed and substantiated. The developed auditing system, based on the principles of systematic analysis and integrating state-of-the-art tools, significantly enhances the efficiency, reliability, and scalability of the vulnerability detection process. Particular attention is paid to the practical integration of these methods into the full audit lifecycle, aligning with the professional standards employed by leading auditing firms for risk assessment. The research findings have practical significance for enhancing the security of decentralized financial systems and mitigating risks associated with the operation of smart contracts.

Keywords: smart contracts, Solidity, EVM, Ethereum, audit, automation, static analysis, dynamic analysis, formal verification, security, system analysis.

Клеймёнов Антон Дмитриевич
Аспирант, Российская академия народного хозяйства и государственной службы при Президенте Российской Федерации
Antonkl22@gmail.com

Аннотация. Работа посвящена системному анализу и разработке методов автоматизированного аудита смарт-контрактов в блокчейн сети Ethereum. В рамках исследования проведен анализ ключевых уязвимостей, присущих программному коду смарт-контрактов, и систематизированы ограничения традиционных, ручных методов аудита в условиях экспоненциального роста сложности и количества децентрализованных приложений. Предложена и обоснована многоуровневая классификация автоматизированных методов, включающая статический, динамический анализ и формальную верификацию. Разработанная система аудита, базирующаяся на принципах системного анализа и интегрирующая передовые инструменты, позволяет существенно повысить эффективность, надежность и масштабируемость процесса выявления уязвимостей. Особое внимание уделено практической интеграции данных методов в полный жизненный цикл аудита, что соответствует профессиональным стандартам, применяемым в ведущих аудиторских компаниях при оценке рисков. Результаты исследования имеют практическое значение для повышения безопасности децентрализованных финансовых систем и минимизации рисков, связанных с эксплуатацией смарт-контрактов.

Ключевые слова: смарт-контракты, Solidity, EVM, Ethereum, аудит, автоматизация, статический анализ, динамический анализ, формальная верификация, безопасность, системный анализ.

Системный анализ и проблематика аудита смарт-контрактов

Эволюция и системные риски смарт-контрактов

Технология блокчейн и платформа Ethereum, произвели революцию в области децентрализованных приложений и финансовых систем. Смарт-контракты, как самоисполняющиеся компьютерные программы, стали краеугольным камнем этой парадигмы, автоматизируя выполнение условий соглашений без необходимости в посредниках, полагаясь только на код. С момента их появления в 2015 году смарт-контракты перешли от простых протоколов обмена токенами к сложным, многокомпонентным системам, управляющим миллиардами долларов в активах, формируя основу для экосистемы децентрализованных финансов (DeFi) [1, 2]. В 2025 году объём рынка смарт-контрактов в сети Ethereum составил более 2 млрд долларов. По прогнозам канадской консалтинговой компании «Precedence Research», при существующей динамике развития, к 2034 году рынок смарт-контрактов вырастет до 815 млрд долларов, что говорит о потенциально крупном развитии этого направления в будущем [3].

По мере увеличения сложности и стоимости заключаемых в них активов, они всё чаще становятся объектом атак и мошеннических действий со стороны злоумышленников. Проблема заключается в неизменяемости кода после его размещения в блокчейне: любая, даже незначительная ошибка, допущенная на этапе разработки, становится постоянным источником риска, а любая специально заложенная уязвимость остаётся в нём навсегда. Хотя технология блокчейн и позволяет увидеть весь код смарт-контракта (см. рисунок 1), понять его достаточно сложно среднестатистическому человеку, а оценить безопасность практически (без необходимых знаний и автоматизированных проверок) невозможно.

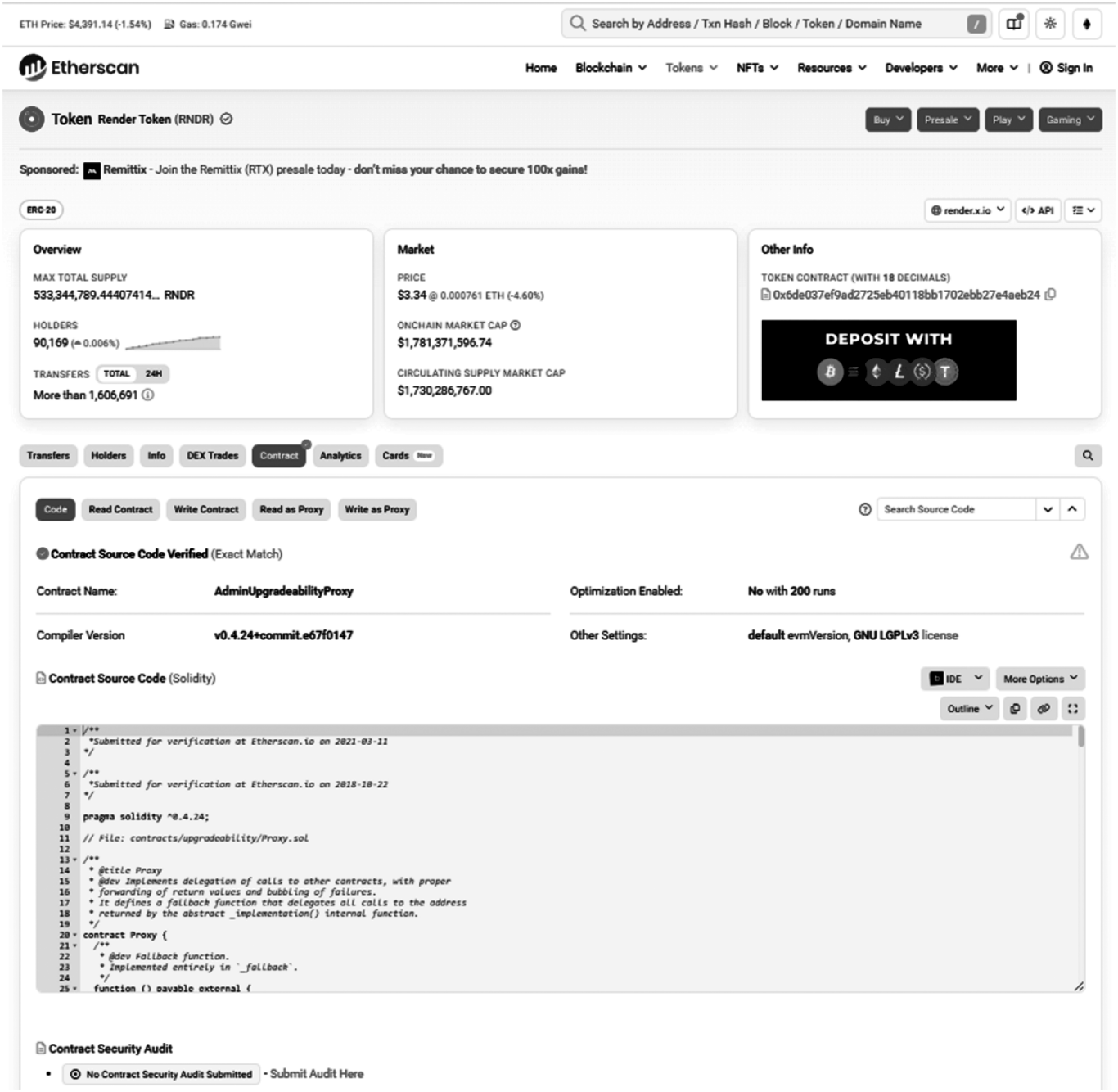

Рис. 1. Код смарт-контракта на сайте Etherscan.io

Ограничения классического аудита и необходимость автоматизации

Классический аудит смарт-контрактов, основанный на ручном анализе кода экспертом-аудитором, имеет ряд ограничений. Он является трудоемким, требует высокой квалификации и глубоких знаний не только языков программирования смарт-контрактов (Solidity или Vyper), но и особенностей виртуальной машины Ethereum (EVM — Ethereum Virtual Machine). Человеческий фактор часто приводит к ошибкам, особенно при анализе большого количества кода, что делает ручной аудит неэффективным и не масштабируемым. В условиях, когда ежедневно развертываются сотни новых контрактов, такой подход не способен обеспечить требуемый уровень покрытия. Более того, ручной аудит не может эффективно выявить сложные зависимости между смарт-контрактами или неочевидные уязвимости, которые проявляются только в определенных последовательностях транзакций.

Именно эти системные проблемы инициируют переход к автоматизации. Автоматизированные методы аудита позволяют проводить первичный анализ кода в масштабе, выявлять типовые уязвимости с высокой скоростью и точностью, а также формализовать про-

цесс проверки. Цель автоматизации — не заменить человеческого аудитора, а предоставить ему мощный инструментарий для фокусировки на наиболее сложных и критических аспектах безопасности, таких как анализ бизнес-логики, проверка архитектурных решений и оценка уникальных рисков, которые не поддаются автоматическому выявлению. Применение автоматизированных инструментов позволяет сократить время аудита, снизить его стоимость и повысить его надежность.

Цели и задачи исследования

Настоящее исследование ставит своей целью разработку и обоснование системы проведения автоматизированного аудита смарт-контрактов, которая позволит нивелировать риски, присущие ручному анализу, и значительно повысить общий уровень безопасности децентрализованных приложений.

Для решения задачи необходимо:
- Провести детальный системный анализ и классификацию уязвимостей смарт-контрактов Ethereum.
- Систематизировать существующие методы автоматизированного анализа: статический, динамический, формальную верификацию и оценить их сильные и слабые стороны.
- Разработать поэтапную методологию, интегрирующую различные автоматизированные методы в единый, последовательный процесс аудита.
- Оценить эффективность предложенной методологии на основе анализа реальных кейсов и сравнения с традиционными подходами.
- Сформулировать практические рекомендации для аудиторских компаний и разработчиков по внедрению автоматизированных процессов.

Системная классификация уязвимостей смарт-контрактов Ethereum

Для эффективного аудита, как ручного, так и автоматизированного, необходимо иметь классификацию угроз. Ввиду опыта разработки смарт-контрактов на языке Solidity, автором предлагается классификация уязвимостей, написанных на этом языке. Представленная ниже классификация основана на результатах анализа более чем 100 аудиторских отчетов (в том числе открытые отчёты крупнейшего аудитора Certik до 2023 года включительно) и выявленных угрозах самостоятельно [3].

Уязвимости, связанные с особенностями EVM

Эти уязвимости проистекают из-за специфики работы виртуальной машины Ethereum и являются одними из наиболее распространенных и критических [4].

Reentrancy (Повторный вход). Это одна из самых известных уязвимостей, использованная в атаке на The DAO. Она возникает, когда внешний вызов (например, call.value ()) делается к другому контракту, который затем рекурсивно вызывает исходный контракт, не дожидаясь, пока состояние последнего будет обновлено. Например, злоумышленник может многократно вызывать функцию вывода средств, пока баланс на контракте еще не обнулился. Современные инструменты автоматизации могут обнаруживать потенциальные reentrancy путем анализа графа вызовов [5].

Integer Overflow/Underflow (Переполнение/недополнение целых чисел). Эта уязвимость возникает, когда результат арифметической операции выходит за пределы диапазона, поддерживаемого типом данных. В Solidity, например, uint256 может хранить значения от 0 до 2256–1. Если к 2256–1 прибавить 1, произойдет переполнение, и результат станет 0. Аналогично, вычитание 1 из 0 приведет к недополнению и получению 2256–1. Это может быть использовано для манипуляции балансами, ценами или правами доступа. Эта уязвимость была широко распространена в более старых версиях Solidity (до версии 0.8.0, в которой внедрили автоматическую проверку [6]) однако она все еще может возникать в более сложных сценариях.

Timestamp Dependency (Зависимость от метки времени). Использование block.timestamp или block.number для выполнения критически важных операций, таких как распределение наград или случайная генерация, является небезопасным, поскольку могут незначительно манипулировать меткой времени в своих интересах. Злоумышленник может использовать это для предсказания исхода или для получения преимущества в азартных играх на блокчейне.

Gas Limit and Loops (Лимит газа и циклы). Транзакции в Ethereum ограничены лимитом газа. Если в контракте есть циклы с неопределенным числом итераций, существует риск, что транзакция исчерпает газ, не завершив свою работу. Это может привести к DoS-атакам (отказ в обслуживании) или «заморозке» контракта.

Уязвимости логики и бизнес-процессов

Эти уязвимости не связаны напрямую с особенностями EVM, а скорее являются следствием ошибок в проектировании бизнес-логики контракта.

Access Control (Контроль доступа). Проблема возникает, когда критически важные функции контракта недостаточно защищены. Классический пример — функция withdrawAllFunds(), которая позволяет вывести все средства, но не проверяет, имеет ли отправитель транзакции (msg.sender) на это право. В таком случае любой поль-

зователь может ею воспользоваться. Эта уязвимость, по сути, нарушает базовый принцип информационной безопасности — принцип наименьших привилегий, и к сожалению, до сих пор встречается очень часто.

Front-running (Опережение). Эта атака возможна из-за прозрачности блокчейна. Злоумышленник видит выгодную, но ещё не исполненную транзакцию в мемпуле (очереди транзакций) и «подрезает» её. Для этого он отправляет идентичную транзакцию, но предлагает майнерам более высокую комиссию (gas price), чтобы его версия была обработана первой. Такая тактика особенно эффективна против децентрализованных бирж (DEX) и смарт-контрактов аукционов.

Logic Errors (Ошибки логики). Это обширная категория уязвимостей, связанная не с безопасностью доступа, а с внутренней flawed логикой работы контракта. Сюда входят ошибки в математических вычислениях, некорректное изменение внутренних состояний контракта, неверная реализация механизмов мультисигнатур или любые другие баги, которые искажают изначальный замысел разработчика и ведут к непредсказуемому поведению системы.

Статический анализ как фундамент автоматизации аудита

Теоретические основы статического анализа

Статический анализ — это процесс исследования программного кода без его фактического выполнения. В контексте смарт-контрактов он направлен на выявление потенциальных уязвимостей, ошибок и нарушений стандартов кодирования на этапе разработки, до развертывания контракта в блокчейн. Статический анализ является первой и самой эффективной «линией обороны», так как позволяет обнаружить большинство типовых проблем с высокой скоростью и без дополнительных затрат на газ или развертывание тестовой сети. Ключевые методы, используемые в статическом анализе смарт-контрактов:

Анализ синтаксического дерева (AST). Это древовидное представление структуры кода, которое позволяет анализировать его семантику. Инструменты парсят исходный код в AST и затем обходят его, обнаруживая определенные шаблоны, которые могут указывать на уязвимость. Например, поиск call.value() без соответствующей проверки может свидетельствовать о потенциальной уязвимости reentrancy.

Анализ потока управления (Control Flow Graph, CFG). CFG представляет собой граф, где узлы — это базовые блоки кода, а ребра — возможные пути исполнения. Анализируя CFG, можно обнаружить недостижимый код, бесконечные циклы или сложные пути, которые могут скрывать логические ошибки.

Анализ потока данных (Data Flow Analysis, DFA). DFA отслеживает, как данные перемещаются и изменяются в процессе выполнения программы. В контексте аудита это позволяет выявить неинициализированные переменные, некорректное использование данных или утечку конфиденциальной информации.

Обзор и сравнительный анализ инструментария

На рынке существует широкий спектр инструментов для статического анализа, каждый из которых имеет свои особенности и специализацию. Ниже представлен обзор наиболее распространенных.

Статический анализ — это краеугольный камень автоматизированного аудита, но он не лишен недостатков. Он может давать ложные срабатывания (false positives), поскольку не учитывает контекст выполнения кода и реальные входные данные. Кроме того, статический анализ не может выявить уязвимости, которые зависят от взаимодействия с внешними контрактами или внешними событиями, такими как цена газа или динамическое поведение оракулов. Эти ограничения обуславливают необходимость в применении более продвинутых методов.

Slither — один из самых популярных и мощных инструментов [6]. Slither написан на Python и использует анализ потока данных для выявления широкого спектра уязвимостей, включая reentrancy, проблемы с контролем доступа, небезопасные call() и другие [7]. Slither также предоставляет возможность написания кастомных детекторов, что делает его гибким инструментом для выявления специфических уязвимостей. В моей практике аудитора Slither является обязательным инструментом на первом этапе проверки, позволяя быстро получить список потенциальных рисков для дальнейшей верификации.

Mythril — инструмент, сочетающий статический и символический анализ [8, 9]. Он может анализировать байт-код контракта, что позволяет ему находить уязвимости даже при отсутствии исходного кода. Mythril хорошо подходит для обнаружения integer overflow/underflow, reentrancy и других низкоуровневых проблем.

Динамический анализ и формальная верификация: преодоление ограничений

Динамический анализ — это процесс исследования кода во время его исполнения. В контексте смарт-контрактов это означает развертывание контракта в тестовой среде и подачу на него различных транзакций для наблюдения за его поведением. Динамический

анализ способен выявить уязвимости, которые не могут быть найдены статическими анализаторами, так как он работает с реальными данными и учитывает контекст выполнения.

Ключевые методы динамического анализа:

— Fuzzing (Фаззинг). Этот метод заключается в подаче на вход контракта большого количества случайных или полу случайных данных (входящих транзакций) с целью вызвать непредвиденное поведение или сбой. Фаззинг эффективно находит ошибки, которые проявляются только при определенных комбинациях входных параметров.
— Symbolic Execution (Символьное исполнение). Этот более сложный метод анализирует выполнение программы с символическими (неконкретными) значениями входных данных. Вместо выполнения одной ветки кода, символьное исполнение строит дерево всех возможных путей выполнения. Это позволяет находить уязвимости, которые требуют очень специфических условий для срабатывания.

Обзор инструментов динамического анализа:

— Echidna. Инструмент, разработанный компанией Trail of Bits. Echidna — это фаззер, который работает с Solidity-контрактами. Он позволяет аудитору писать «свойства» — инварианты, которые должны всегда оставаться истинными (например, «баланс контракта никогда не должен быть отрицательным»), и затем ищет последовательность транзакций, которая нарушит это свойство.
— Manticore. Фреймворк для символьного исполнения, который анализирует байт-код EVM. Manticore может автоматически находить пути выполнения, которые приводят к критическим состояниям, таким как reentrancy, integer overflow/underflow или несанкционированный доступ. Его основное ограничение — это проблема «взрыва путей» (path explosion), когда количество возможных путей становится экспоненциально большим для сложных контрактов.

Формальная верификация: математическая строгость

Формальная верификация — это наиболее строгий метод анализа, который использует математические методы для доказательства корректности программы. Вместо поиска ошибок, формальная верификация доказывает их отсутствие, основываясь на спецификациях и свойствах, которые должен удовлетворять контракт. Она идеально подходит для аудита критически важных компонентов, таких как протоколы управления активами или логика токенов, где даже одна ошибка может привести к катастрофическим потерям.

S-LVM (векторизация и верификация). Это специализированный метод для верификации смарт-контрактов. Он позволяет доказать, что определенные свойства, сформулированные в формальном языке, никогда не будут нарушены, вне зависимости от последовательности входных данных.

Обзор инструментов формальной верификации [10]:

— Certora Prover. Коммерческий инструмент, который использует исчерпывающий анализ для верификации смарт-контрактов. Он позволяет разработчикам и аудиторам писать формальные спецификации и доказывать, что контракт им соответствует.
— K-Framework. Инструмент с открытым исходным кодом, который позволяет создавать формальные модели для различных языков программирования, включая EVM. K-Framework используется для проверки свойств безопасности и корректности.

Формальная верификация — это очень мощный, но трудоемкий метод. Написание спецификаций требует высокой квалификации, а процесс верификации может быть вычислительно сложным и длительным. По этой причине, формальная верификация обычно применяется только к наиболее критическим и сложным частям кода.

Комплексная методология автоматизированного аудита

Автор предлагает комплексную, многоуровневую автоматизированную систему, которая интегрирует статический, динамический и формальную верификацию в единый процесс. Эта методология предназначена для максимизации эффективности и минимизации рисков.

Рис. 2. Схема проведения автоматизированной поверки безопасности смарт-контракта

Этап 1. Подготовительный анализ и сканирование (L1). На этом этапе используется набор быстрых, легко интегрируемых инструментов статического анализа (например, Solhint, Slither). Цель — выявить типовые уязвимости, нарушения стандартов кодирования и подготовить список потенциальных проблем для дальнейшего анализа. Этот этап позволяет быстро отсеять 80 % типовых ошибок, не требующих глубокого анализа.

Этап 2. Углубленный статический и символический анализ (L2). На этом этапе применяются более мощные инструменты (например, Mythril), которые анализируют байт-код и проводят символическое исполнение для выявления более сложных уязвимостей, которые не были найдены на первом этапе.

Этап 3. Динамический анализ и фаззинг (L3). На этом этапе контракт разворачивается в тестовой среде, и на него подаются фаззинговые транзакции с помощью инструментов типа Echidna. Цель — выявить уязвимости, которые проявляются только в динамическом контексте, при определенных последовательностях вызовов. Этот этап особенно важен для тестирования межконтрактных взаимодействий.

Этап 4. Формальная верификация критических компонентов (L4). На этом этапе выбираются наиболее критические части кода, отвечающие за управление активами, правами доступа или сложную бизнес-логику, и для них пишутся формальные спецификации. Затем используется формальный верификатор (например, Certora Prover) для доказательства отсутствия уязвимостей.

Этап 5. Ручной аудит и анализ бизнес-логики (L5). Несмотря на широкое использование автоматизации, ручной аудит остается критически важным. Аудитор, используя результаты, полученные на предыдущих этапах, фокусируется на высокоуровневых проблемах: анализе бизнес-логики, оценке архитектуры, проверке соответствия контракта документации и выявлению атак, которые не поддаются автоматическому обнаружению, таких как атаки на оракулы или front-running.

Оценка эффективности и масштабируемости

Предложенная методология значительно повышает эффективность и масштабируемость аудиторского процесса. Она позволяет сократить время на ручной анализ, автоматизируя рутинные задачи, и позволяет аудитору сосредоточиться на высокоуровневых рисках. Интеграция автоматизированных инструментов в процессы разработки dApps обеспечивает непрерывный мониторинг безопасности, что является ключевым требованием для корпоративных клиентов. Внедрение данной методологии позволит аудиторским компаниям предложить более надежный и всесторонний продукт, соответствующий лучшим практикам в области IT-аудита и кибербезопасности.

Заключение

Проведенное исследование позволило выработать и обосновать комплексную методологию автоматизированного аудита смарт-контрактов Ethereum. Системный анализ архитектуры, уязвимостей и методов их выявления позволил сформировать целостную картину, которая выходит за рамки узкоспециализированного анализа кода. Предложенная многоуровневая методология, интегрирующая статический и динамический анализ с формальной верификацией, является гибким и масштабируемым инструментом для повышения безопасности децентрализованных приложений.

Разработанная методология не только повышает эффективность аудита, но и позволяет минимизировать системные риски, связанные с эксплуатацией смарт-контрактов, что имеет важное значение для защиты активов пользователей и укрепления доверия к технологии блокчейна. Внедрение этой методологии в практику аудиторских компаний, особенно на фоне возрастающих требований к кибербезопасности в финансовом секторе, является критически важным шагом.

Дальнейшие исследования могут быть направлены на разработку гибридных инструментов, которые сочетают в себе элементы статического и динамического анализа, а также на создание более интуитивно понятных инструментов формальной верификации, что позволит сделать их доступными для более широкого круга разработчиков и аудиторов.

ЛИТЕРАТУРА

1. Ethereum.org. Документация по смарт-контрактам [Электронный ресурс]. URL: https://ethereum.org/ru/developers/docs/smart-contracts/ (дата обращения: 22.08.2025).
2. Wood G. Ethereum: A Secure Decentralized General Ledger // Ethereum Yellow Paper. — 2014. — URL: https://ethereum.github.io/yellowpaper/paper.pdf (дата обращения: 29.08.2025).
3. Precedence Research. Smart Contracts Market Size, Share, and Growth Report [Электронный ресурс]. URL: https://www.precedenceresearch.com/smart-contracts-market (дата обращения: 27.08.2025).
4. Solidity. Ethereum Smart Contracts [Электронный ресурс]. URL: https://solidity.readthedocs.io/en/latest/ (дата обращения: 28.08.2025).
5. Luu L., Chu D.-H., Olickel H. et al. Making Smart Contracts Smarter // Proceedings of the 23rd ACM Conference on Computer and Communications Security. — 2016. — URL: https://eprint.iacr.org/2016/633.pdf (дата обращения: 28.08.2025).
6. Solidity Documentation. Breaking Changes in Solidity 0.8.0 [Электронный ресурс]. URL: https://docs.soliditylang.org/en/latest/080-breaking-changes.html (дата обращения: 28.08.2025).
7. Slither: Static Analyzer for Solidity [Электронный ресурс] // GitHub. — URL: https://github.com/crytic/slither?tab=readme-ov-file#using-pip (дата обращения: 28.08.2025).
8. Tsankov P., Dan A., Cohen B. et al. Securify: Automated Security Analysis of Smart Contracts // Proceedings of the 2018 IEEE Symposium on Security and Privacy. — 2018. — URL: https://ieeexplore.ieee.org/document/8418653 (дата обращения: 28.08.2025).
9. Mythril: Security Analysis Tool for EVM Bytecode [Электронный ресурс] // GitHub. — URL: https://github.com/ConsenSysDiligence/mythril (дата обращения: 29.08.2025).
10. Bhargavan K., Delignat-Lavaud A., Fournet C. et al. Formal Verification of Smart Contracts // Proceedings of the 2016 ACM Workshop on Blockchain, Cryptocurrencies, and Contracts. — 2016. — URL: https://dl.acm.org/doi/10.1145/2996911.2996918 (дата обращения: 29.08.2025).

Журнал «Современная наука: актуальные проблемы теории и практики»