Поиск уязвимостей в смарт-контрактах на основе машинного обучения

В.С. Белоус[I], И.А. Тарханов[II,III]

[I] Национальный исследовательский технологический университет «МИСиС», г. Москва, Россия
[II] Федеральное государственное бюджетное учреждение высшего образования «Государственный академический университет гуманитарных наук», г. Москва, Россия
[III] Федеральный исследовательский центр «Информатика и управление» Российской академии наук», г. Москва, Россия

Аннотация. С ростом популярности блокчейн-проектов растет и количество децентрализованных приложений на их основе. Центральным звеном этих приложений являются смарт-контракты. Эта технология еще относительно молода и имеет ряд известных проблем с безопасностью. Статистика взлома смарт-контрактов свидетельствует об актуальности проблемы поиска уязвимостей в коде смарт-контрактов. В статье описано 3 модели машинного обучения для поиска уязвимостей в смарт-контрактах, написанных на языке Solidity. Особое внимание уделяется подготовке датасета для обучения и сравнения с известными анализаторами кода. Полученные по результатам обучения и апробации моделей метрики позволяют утверждать, что модель, состоящая из трех двунаправленных рекуррентных слоев BiGRU и трех сверточных слоев CNN эффективна при поиске уязвимостей смарт-контрактов.

Ключевые слова: *уязвимость, смарт-контракт, машинное обучение, BiGRU, CNN.*
DOI: 10.14357/20790279240310 **EDN: URZLYI**

Введение

Современная цифровая экономика, поддерживаемая технологией блокчейн, сталкивается с постоянным ростом интереса к смарт-контрактам – программам, автоматизирующим и обеспечивающим выполнение условий сделок на блокчейн-платформах. Смарт-контракты являются фундаментальным элементом децентрализованных приложений (далее DApps) и обеспечивают правила децентрализованных и прозрачных отношений в сетях с нулевым доверием [1]. С другой стороны распространение использования смарт-контрактов выявляет несовершенство этой технологии. Все больше появляется информации об уязвимостях [2], что поднимает вопросы безопасности в контексте цифровых децентрализованных сред.

Недостаточная проверка безопасности перед развертыванием смарт-контрактов на платформах, таких как Ethereum, приводит к уязвимостям, которые могут быть использованы злоумышленниками для проведения атак и несанкционированного доступа к финансовым средствам [3]. Потеря средств из-за уязвимостей в смарт-контрактах может иметь серьезные последствия для участников системы, подчеркивая необходимость разработки эффективных методов обнаружения и предотвращения подобных угроз. Если рассматривать финансовые потери, то в 2016 году общий урон от уязвимостей в смарт-контрактах составил 100 миллионов долларов, в 2017 – 200 миллионов, в 2018 – 1,5 миллиарда, в 2019 и 2020 – 500 миллионов, в 2021 – 3,3 миллиарда, в 2022 – 3,8 миллиарда и около 1,5 миллиардов в 2023 году [1].

Существующие методы обнаружения уязвимостей в смарт-контрактах ограничены в контексте быстро меняющейся криптографической и кибербезопасной среды. В связи с этим, использование машинного обучения представляется перспективным подходом для повышения точности и общей эффективности обнаружения уязвимостей, учитывая динамичность и сложность кода смарт-контрактов. Множество исследований на тему поиска уязвимостей в смарт-контрактах [4] в последние годы подчеркивает актуальность этой проблемы. Целью данного исследования является разработка нового метода, основанного на машинном обуче-

нии. Это не только позволит улучшить надежность обнаружения уязвимостей в смарт-контрактах, но и способствует обеспечению устойчивости и безопасности блокчейн-систем в целом.

1. Уязвимости смарт-контрактов

Смарт-контракты – это компьютерные программы, которые размещаются и выполняются на блокчейн-сети. Каждый смарт-контракт состоит из кода, определяющего заранее установленные условия, которые, когда они выполняются, выдают определенные результаты [5]. Запускаясь на децентрализованной блокчейн-сети вместо централизованного сервера, смарт-контракты позволяют нескольким сторонам приходить к общему результату точным, своевременным и устойчивым к вмешательству образом. Блокчейны с возможностью использования смарт-контрактов предоставляют разработчикам возможность написания произвольной логики приложения поверх децентрализованного консенсуса, который блокчейны предоставляют по умолчанию.

Смарт-контракт определяет, как пользователи могут с ним взаимодействовать, включая, кто может взаимодействовать со смарт-контрактом, в какие моменты времени и какие вводы приводят к каким выводам. Результатом являются многосторонние цифровые соглашения, которые эволюционируют от сегодняшнего вероятностного состояния, где они, вероятно, будут выполнены по желаемому, до нового детерминированного состояния, где они гарантированно будут выполняться в соответствии с их кодом [6].

В данном исследовании мы сфокусируемся на анализе уязвимостей смарт-контрактов на языке программирования Solidity и блокчейна Ethereum по следующим причинами:

– Широкое распространение. Ethereum является одним из самых популярных блокчейнов, используемых для развертывания смарт-контрактов.
– Множество смарт-контрактов. На Ethereum развернуто множество смарт-контрактов, предоставляющих обширный набор данных для исследования. Это обеспечивает разнообразие в типах контрактов, исследование которых может привести к обнаружению различных уязвимостей.
– Открытость и доступность сети Ethereum, которая является открытой и децентрализованной платформой. Данные о смарт-контрактах и их транзакциях доступны публично. Это обеспечивает открытость для исследователей, желающих провести анализ и оценку безопасности.
– Активное сообщество и развитие. Ethereum обладает активным сообществом разработчиков и исследователей, что обеспечивает поддержку и обмен опытом в области безопасности смарт-контрактов.

Согласно статье [7] наиболее распространенными и опасными уязвимостями в смарт-контрактах являются:

– арифметическое переполнение,
– некорректное управление доступом,
– неотмеченная отправка,
– рекурсия.

Исследователи используют различные методологии для анализа смарт-контрактов, многие из которых общедоступны под лицензиями с открытым исходным кодом. Существует пять общих методологий для анализа смарт-контрактов, таких как формальная верификация [8], символьное выполнение [9], фаззинг [10], промежуточное представление [11] и машинное обучение [12].

Машинное обучение достигает обнадеживающих результатов в области безопасности программ [4]. По сравнению с вышеупомянутыми методами и методологиями, оно объединяет в себе статический анализ и динамическое обнаружение для решения проблем высокого уровня ложных отрицательных результатов статического анализа и низкого покрытия кода динамического анализа.

2. Математическая постановка задачи поиска уязвимостей

2.1. Математическое определение уязвимости байт-кода смарт-контракта

Пусть дан байткод смарт-контракта C, представленный как последовательность инструкций $I_1, I_2, ..., I_n$, где каждая инструкция I_i может содержать операции чтения/записи в хранилище, математические операции, вызовы других контрактов и т. д. Пусть также имеется множество возможных уязвимостей V, которые могут присутствовать в контракте.

Задача состоит в определении, содержит ли байткод C какую-либо уязвимость из множества V. Это можно сформулировать следующим образом:

– определение функции $f: C \rightarrow \{0,1\}$, где $f(C) = 1$, если байткод C содержит уязвимость, и $f(C) = 0$ в противном случае;
– пусть $V = \{v_1, v_2, ..., v_m\}$ – множество из m возможных уязвимостей. Каждая уязвимость v_i может быть описана некоторым предикатом $P_i(C)$, который верен, если уязвимость v_i присутствует в байткоде C, и ложен в противном случае;

– тогда задача поиска уязвимостей в байткоде смарт-контракта сводится к проверке следующего условия:

$$f(C) = \bigvee_{i=1}^{m} P_i(C), \quad (1)$$

где V обозначает логическое ИЛИ. То есть, байткод C содержит уязвимость, если хотя бы один из предикатов $P_i(C)$ истинен.

Подход к решению задачи включает анализ каждого предиката $P_i(C)$ для определения того, верен ли он для данного байткода C или нет.

2.2. Постановка задачи машинного обучения

При обнаружении обозначенных ранее уязвимостей в смарт-контрактах необходимо ставить машинное обучение как задачу многозначной классификации [13].

Пусть есть множество объектов X, множество классов C и обучающая выборка представлена парой (x_i, Y_i), где x_i – объект, Y_i – множество меток, к которым принадлежит объект x_i. Таким образом, каждый объект может быть связан с несколькими метками.

Задачу многозначной классификации можно рассматривать как предсказание множества меток для новых объектов. Модель многозначной классификации обучается на обучающей выборке с использованием функции потерь, которая учитывает совпадение предсказанных и истинных меток.

Подходом к решению задачи многозначной классификации является использование бинарных классификаторов для каждой метки. Так, если у нас есть K классов, то для каждой метки c_i мы обучаем бинарный классификатор $h_i(x)$, который предсказывает, принадлежит ли объект x метке c_i или нет. Функция предсказания для многозначной классификации определена как

$$h(x) = \{c_i | h_i(x) = 1, i = 1,2, \dots, K\}, \quad (2)$$

где $h_i(x)$ – предсказание бинарного классификатора для метки c_i, а i принадлежит диапазону от 1 до 4, включительно.

Для бинарной классификации одной метки будем использовать логистическую функцию потерь [15], которая имеет следующую формулу:

$$L_i(y_i, \hat{y}_i) = -\frac{1}{N}\sum_{j=1}^{N}(y_{ij} * \log(\widehat{y_{ij}}) + (1 - y_{ij}) * \log(1 - \widehat{y_{ij}})) \quad (3)$$

где N – количество примеров в обучающем наборе, y_{ij} – истинное значение i-ой метки для j-го примера (0 или 1), $\widehat{y_{ij}}$ – предсказанная вероятность i-ой метки для j-го примера.

Следующие критерии указывают на то, что разрабатываемая модель справляется с поставленной задачей поиска уязвимостей эффективно [13]:

– должно быть превышающее процентное соотношение истинно положительный и истинно отрицательных исходов на матрице ошибок;
– значение метрик *accuracy, precision, recall* и *F1-score* должно быть не ниже 0,9;
– значение площади под ROC-кривой не ниже 0,8,
– значение общего времени поиска уязвимостей должно быть минимальным.

В ходе эксперимента мы рассмотрим 3 модели, в основании архитектур которых лежали следующие слои:

1) два рекуррентных слоя GRU [14],
2) два рекуррентных слоя GRU и два сверточных слоя CNN [15],
3) три двунаправленных рекуррентных слоя BiGRU и три сверточных слоя CNN.

Первый вариант является одним из самых распространенных в существующих исследованиях [16]. Два последующих проверяют стратегию объединения скорости и легкости сверточных сетей с чувствительностью к порядку рекуррентных сетей. Основная его идея заключается в использовании одномерной сверточной сети для предварительной обработки данных перед передачей их в рекуррентную сеть. Этот прием оказывается особенно выгодным, когда имеющиеся последовательности настолько длинны (с несколькими тысячами интервалов и больше), что их нереально обработать с помощью рекуррентной сети.

3. Результаты подготовки датасета и обучения модели

3.1. Подготовка датасета для обучения

Согласно статье [8], только лишь 1% всех смарт-контрактов в блокчейне Ethereum имеют открытый исходный код. Это может быть обусловлено тем, что исходные коды могут содержать конфиденциальную информацию или коммерческие секреты. Поэтому было принято решение использовать байт-код смарт-контрактов, который находится в публичном доступе на платформе блокчейн.

Формирование датасета осуществлялось из большого датасета crypto_ethereum Google BigQuery [17], содержащего информацию с блокчейна Ethereum. Он был создан для аналитики и исследований в области смарт-контрактов и транзакций Ethereum. Общая схема подготовки датасета представлена на рис. 1.

Сам байт-код смарт-контрактов был взят из таблицы contracts (рис. 2). Для выгрузки понадобились только поля bytecode и block_timestamp. Последнее содержит временную метку блока, в

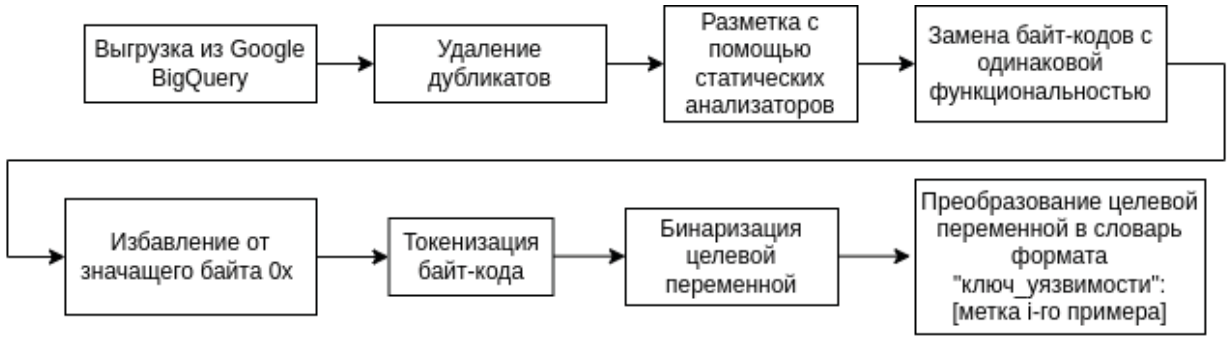

Рис. 1. Общая схема преобразования данных

котором был создан контракт. Метка понадобится для формирования свежих данных.

Далее был создан запрос для получения данных и их последующей выгрузки в несколько таблиц, с которыми можно непосредственно работать. Это делается по причине ограниченности ресурсов платформы Google BigQuery для выполнения запросов. Количество кортежей в проектных таблицах будет 200 тысяч, а всего таких таблиц 15 штук. Общий объем выгруженных данных составит 3 миллиона кортежей.

Перед разметкой данных была выполнена их предварительная обработка по удалению дубликатов. После предварительной обработки количество кортежей уменьшилось почти в 30 раз и составило 106 тысяч.

Далее были сопоставлены индексы из исходного датасета с индексами из размеченного текстового файла с помощью статического анализатора Conkas и были добавлены метки в новый столбец. Значения в столбце vuls ранжируются 0 до 4, где с 0 по 3 метки наши уязвимости, а метка 4 обозначает, что байт-код смарт-контракта безопасен (рис. 3).

Разделение датасета на обучающую, тестовую и валидационную выборки будем осуществлять в процентном соотношении 80/10/10. Общее количество кортежей на тестовой выборке составило 85179, на валидационной и тестовой – 10647.

Распределение каждой метки класса в разделах обучения, тестирования и проверки представлено на рис. 4. Для этого сделаем разбиение количества классов уязвимостей по одному для каждого байт-кода. В результате будет несколько одинаковых байт-кодов с разными уязвимостями.

Анализ графика показывает, что классы довольно несбалансированные. Чтобы бороться с этой проблемой далее используем метрики оценки, которые учитывают дисбаланс классов.

Согласно статье [18] наиболее эффективными методами обработки байткода смарт-контрактов являются:

– замена байт-кодов, имеющих одинаковую функциональность в одну общую операцию. Примером могут послужить опкоды PUSH1 – PUSH32, которым соответствуют байт-коды 0x60 – 0x7f;
– избавление от значащего байта 0x во всех байт-кодах.

Далее, необходимо провести токенизацию [19]. Иными словами разбить байткоды на серию байтов, каждый из которых содержит два символа

Field name	Type	Mode
address	STRING	REQUIRED
bytecode	STRING	NULLABLE
function_sighashes	STRING	REPEATED
is_erc20	BOOLEAN	NULLABLE
is_erc721	BOOLEAN	NULLABLE
block_timestamp	TIMESTAMP	REQUIRED
block_number	INTEGER	REQUIRED
block_hash	STRING	REQUIRED

Рис. 2. Поля таблицы с данными о смарт-контрактах

	bytecode_c	vuls
0	0x6080604052600436106100e6576000357c0100000000...	['4']
1	0x6080604052348015610010576000800fd5b5060043610...	['4']
2	0x60806040526004361061017457600035 7c0100000000...	['1']
3	0x6080604052348015610010576000800fd5b5060043610...	['4']
4	0x6080604052600436106100d0576000357c0100000000...	['4']
...	...	...
85174	0x6080604052348015610010576000800fd5b5060043610...	['3']
85175	0x6080604052600436106100 8a5760003560e01c80638b...	['4']
85176	0x6080604052348015610010576000800fd5b5060043610...	['2', '3']
85177	0x6080604052348015610010576000800fd5b5060043610...	['1', '2']
85178	0x6080604052600436106101d05760003560e01c80637d...	['2']

Рис. 3. Пример кортежей обучающей выборки

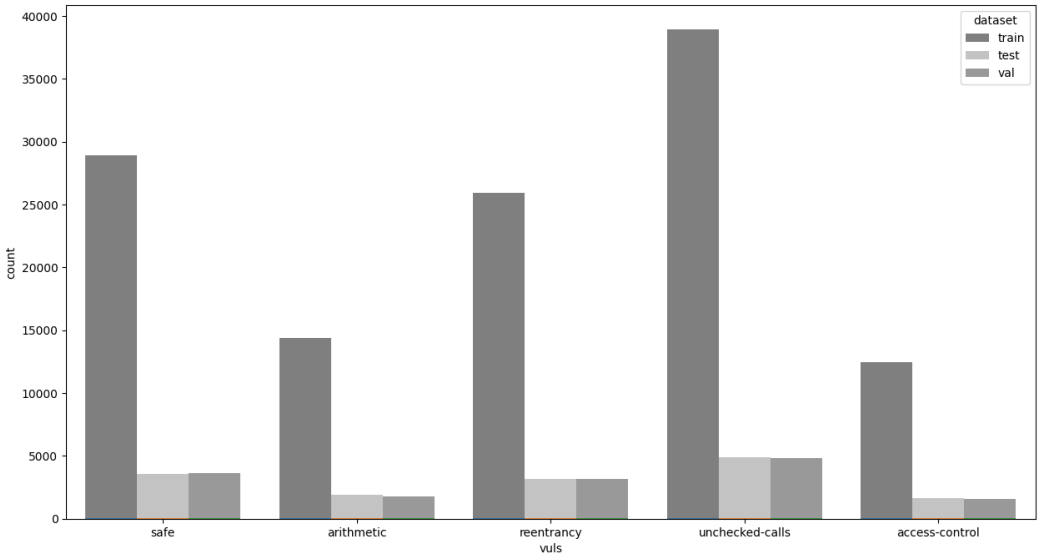

Рис. 4. Классовое распределение данных

и поможет раскрыть особенности смарт-контрактов. Токенизировав байт-код, определим длину выходной последовательности, которая покроет 95% данных. Выбор 95-перцентиля означает, что мы игнорируем 5% данных с наибольшими длинами, что может оказаться не таким критичным для обучения модели, но позволяет избежать слишком больших значений выходной последовательности, которые могли бы привести к избыточным затратам по памяти и вычислительным ресурсам.

Далее метод text_vectorizer из библиотеки TensorFlow выполнит задачу векторизации, преобразуя текстовые данные в числовые последовательности [20]. Следующим шагом будет преобразование обычных векторов в векторы плотных эмбедингов. За это будет отвечать соответствующий слой представленных далее моделей [21].

3.2. Обучение первой модели

Схема архитектуры данной модели представлена на рис. 5. О первых четырех блоках слоев было рассказано выше. Теперь поговорим о роли слоев Dropout и Dense. Слой прореживания или Dropout – один из наиболее эффективных и распространенных приемов регуляризации для нейронных сетей [22]. Прореживание, которое применяется

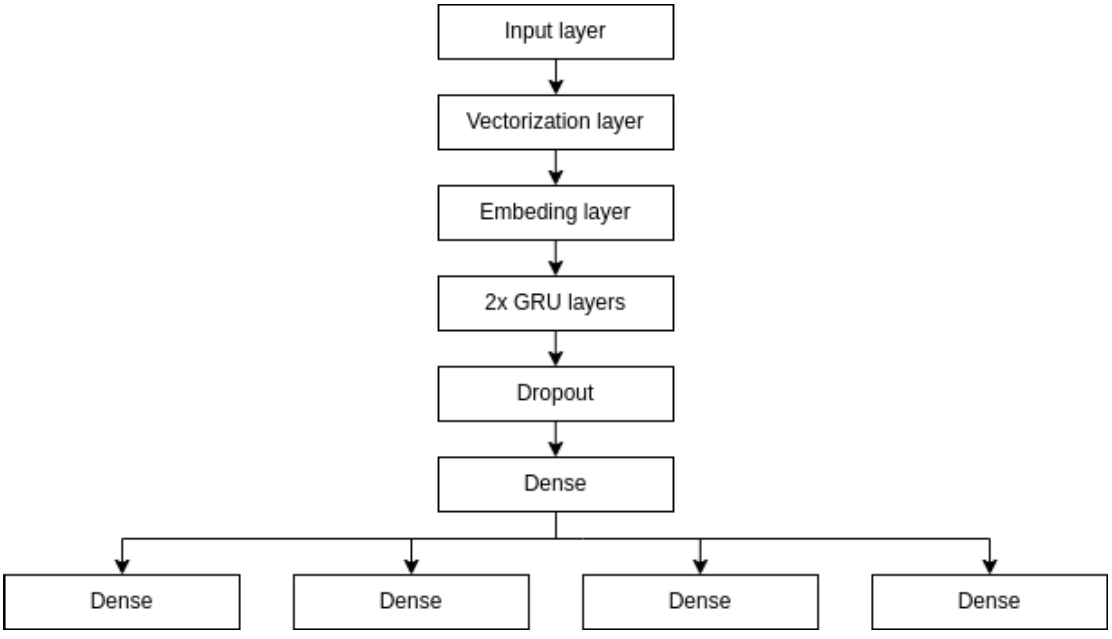

Рис. 5. Архитектура первой модели

к слою, заключается в удалении (присваивании нуля) случайно выбираемым признакам на этапе обучения.

Dense-слои, также известные как fully connected (полносвязные) слои, являются основными строительными блоками нейронных сетей. Они соединяют каждый нейрон из предыдущего слоя с каждым нейроном следующего слоя. Это позволяет модели обучаться на более сложных комбинациях признаков, которые могут быть выявлены в предыдущих слоях. В нашей архитектуре он принимает на вход выходные данные последнего слоя GRU и преобразует их в пространство той же размерности с применением функции активации ReLU [23]. Следующие за этим четыре Dense слоя указывают, что они будут выдавать значение в диапазоне от 0 до 1, что подходит для задачи бинарной классификации.

Для минимизации функции потерь в процессе обучения модели использовался оптимизатор Adam [24]. В нашем случае, данный параметр будет уменьшаться каждые пять эпох на одну десятую, если модель покажет результаты хуже, чем раньше.

Еще одним важным параметром является количество эпох обучения. Эпоха в обучении моделей представляет собой один проход через весь тренировочный набор данных. Количество эпох определяет, сколько раз весь тренировочный набор будет использоваться для обучения модели. На каждой эпохе модель проходит через все примеры в тренировочном наборе данных, вычисляет потери и обновляет веса сети в соответствии с выбранным оптимизатором и функцией потерь. Также нельзя не отметить такой параметр как размер батча. Он определяет количество образцов данных, которые обрабатываются одновременно перед выполнением одного шага обновления весов модели.

Модель будет обучена со следующими параметрами:

– количество эпох – 20,
– размер батча – 352,
– оптимизатор: Adam,
– коэффициент скорости обучения: на начальном этапе 0.001,
– коэффициент прореживания на слое Dropout – 0.2.

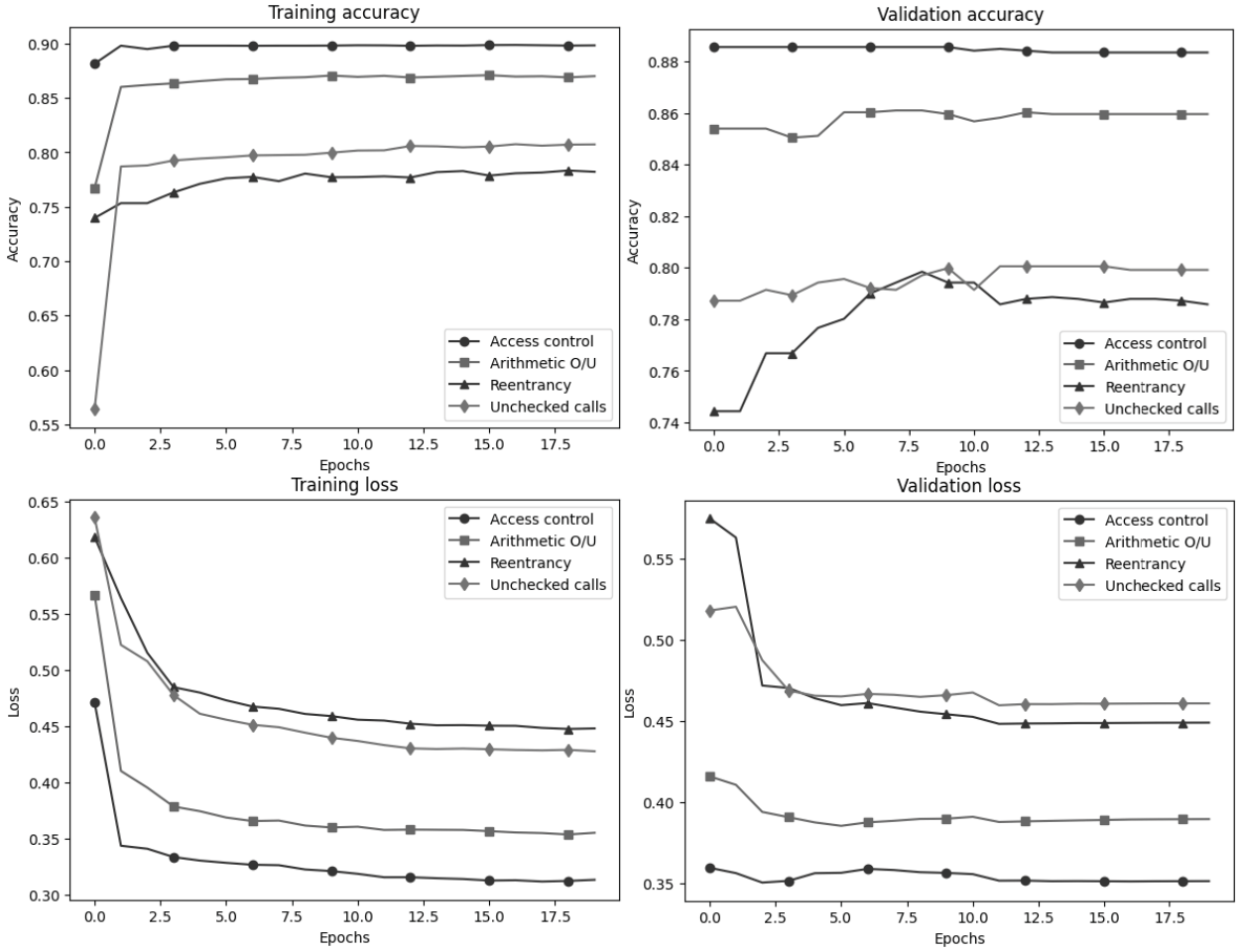

Рис. 6. Результаты обучения первой модели

Табл. 1

Метрики качества первой модели

Метрика/ Уязвимость	Access control	Arithmetic O/U	Reentrancy	Unchecked calls
Accuracy	0.90	0.87	0.77	0.79
Precision	0.81	0.87	0.75	0.75
Recall	0.90	0.87	0.77	0.79
F1-score	0.85	0.82	0.75	0.75

Результаты обучения показаны на рис. 6. Общее время обучения составило 5 дней. На одну эпоху примерно 6 часов. На тестовой выборке модель показала значения, представленные в табл. 1.

Как видно, результаты обучения низкие, поэтому было принято решение адаптировать архитектуру внедрением одномерных сверточных слоев.

3.3. Обучение второй модели

Схема архитектуры данной модели представлена на рис. 7.

Увеличение емкости сети обычно осуществляется за счет увеличения числа параметров слоя или добавления дополнительных слоев. Наложение рекуррентных слоев друг на друга – классический способ конструирования более мощных рекуррентных сетей. Например, в настоящее время алгоритм Google Translate представляет собой стек из семи больших слоев LSTM – это огромная сеть.

Между сверточными и рекуррентными слоями был добавлен слой Reshape. Он преобразует выходные данные пулинга к форме, подходящей для входа в рекуррентные слои.

По сравнению с первой моделью было изменено только количество эпох обучения, оно стало равно 60. Результаты обучения показаны на рис. 8.

Общее время обучения данной модели составило примерно 2 дня. На одну эпоху уходило 45 минут. На тестовой выборке модель показала значения, представленные в табл. 2.

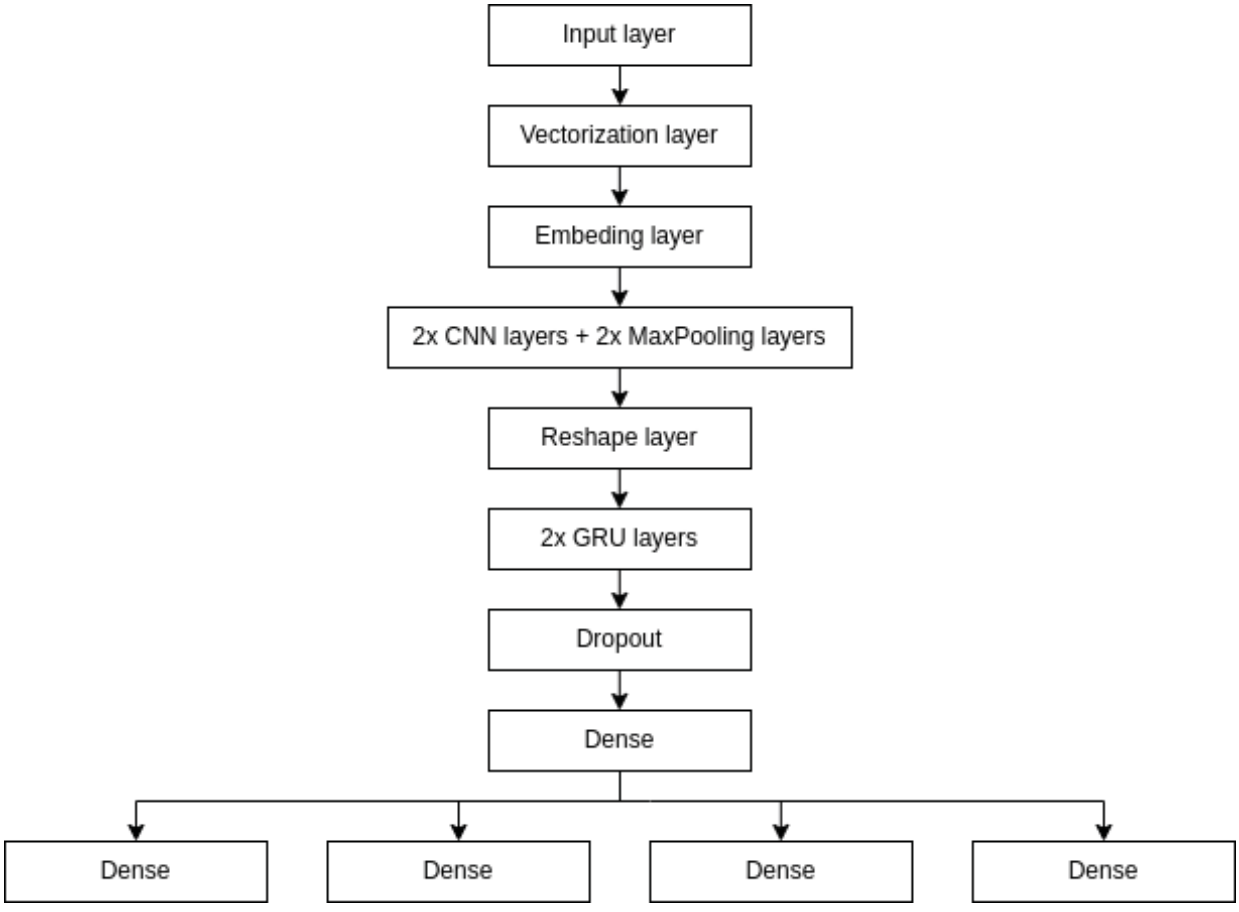

Рис. 7. Архитектура второй модели

Табл. 2

Метрики качества второй модели

Метрика/ Уязвимость	Access control	Arithmetic O/U	Reentrancy	Unchecked calls
Accuracy	0.88	0.87	0.86	0.86
Precision	0.88	0.85	0.86	0.85
Recall	0.89	0.87	0.86	0.86
F1-score	0.88	0.85	0.87	0.86

Качество модели значительно улучшилось по всем метрикам уязвимостям рекурсии и неотмеченной отправки. По арифметическому переполнению и управлению доступом улучшилась F1-мера. Но при этом, точность обучения в уязвимости управления доступом упала на 2 сотых. Это не критично, так как с несбалансированной выборкой как у нас, лучше всего ориентироваться на последние 3 метрики.

3.4. Обучение финальной модели

Схема архитектуры финальной модели представлена на рис. 9.

В финальной модели было принято решение «утяжелить» нашу модель, добавив по одному одномерному сверточному и рекуррентному слою, а также сделать рекуррентные сети двунаправленными. Обрабатывая последовательность в двух направлениях, двунаправленная рекуррентная сеть способна выявить шаблоны, незаметные для однонаправленной сети. Основные параметры модели не отличались от предыдущей. Результаты обучения показаны на рис. 10. На тестовой выборке модель показала наилучшие результаты. Они представлены в табл. 3.

Теперь посмотрим, насколько модель ошибалась в классификации по всем уязвимостям (рис. 11). Сделать это можно с помощью матрицы ошибок.

В качестве инструментов для сравнения были выбраны два популярных анализатора, один статический Mythirl[25], другой динамический sFuzz[26], а

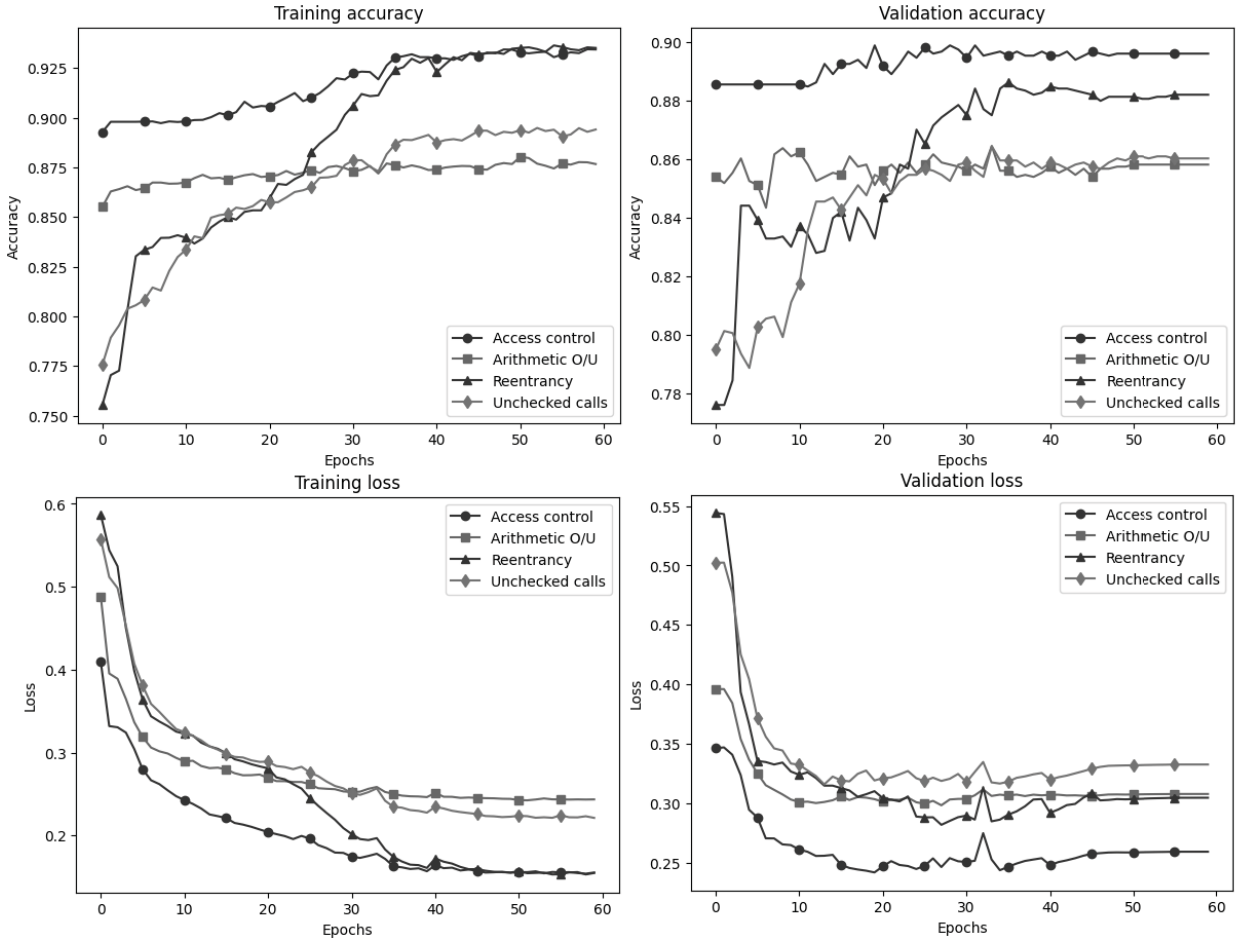

Рис. 8. Результаты обучения второй модели

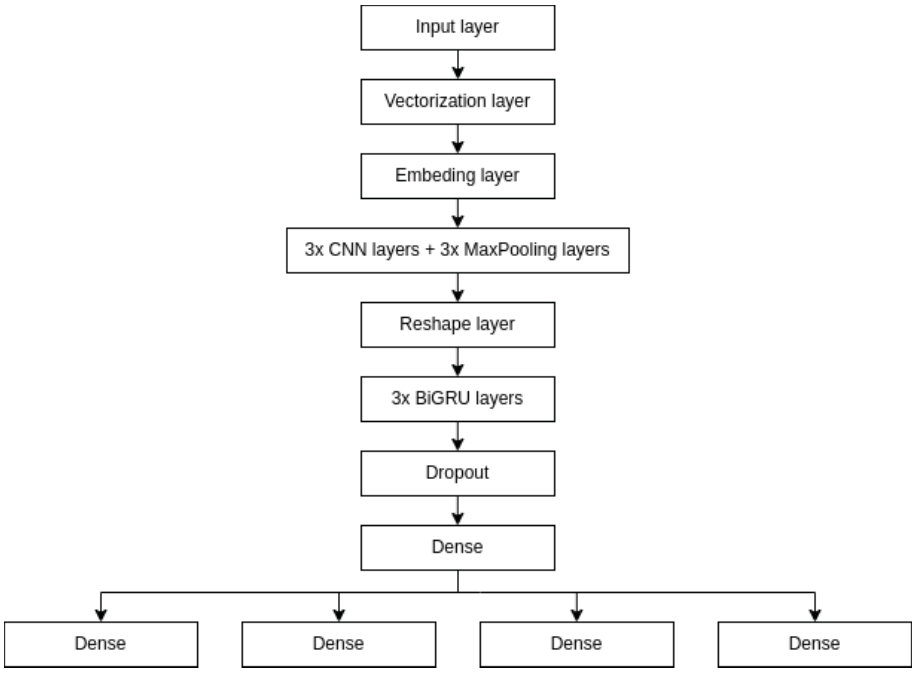

Рис. 9. Архитектура финальной модели

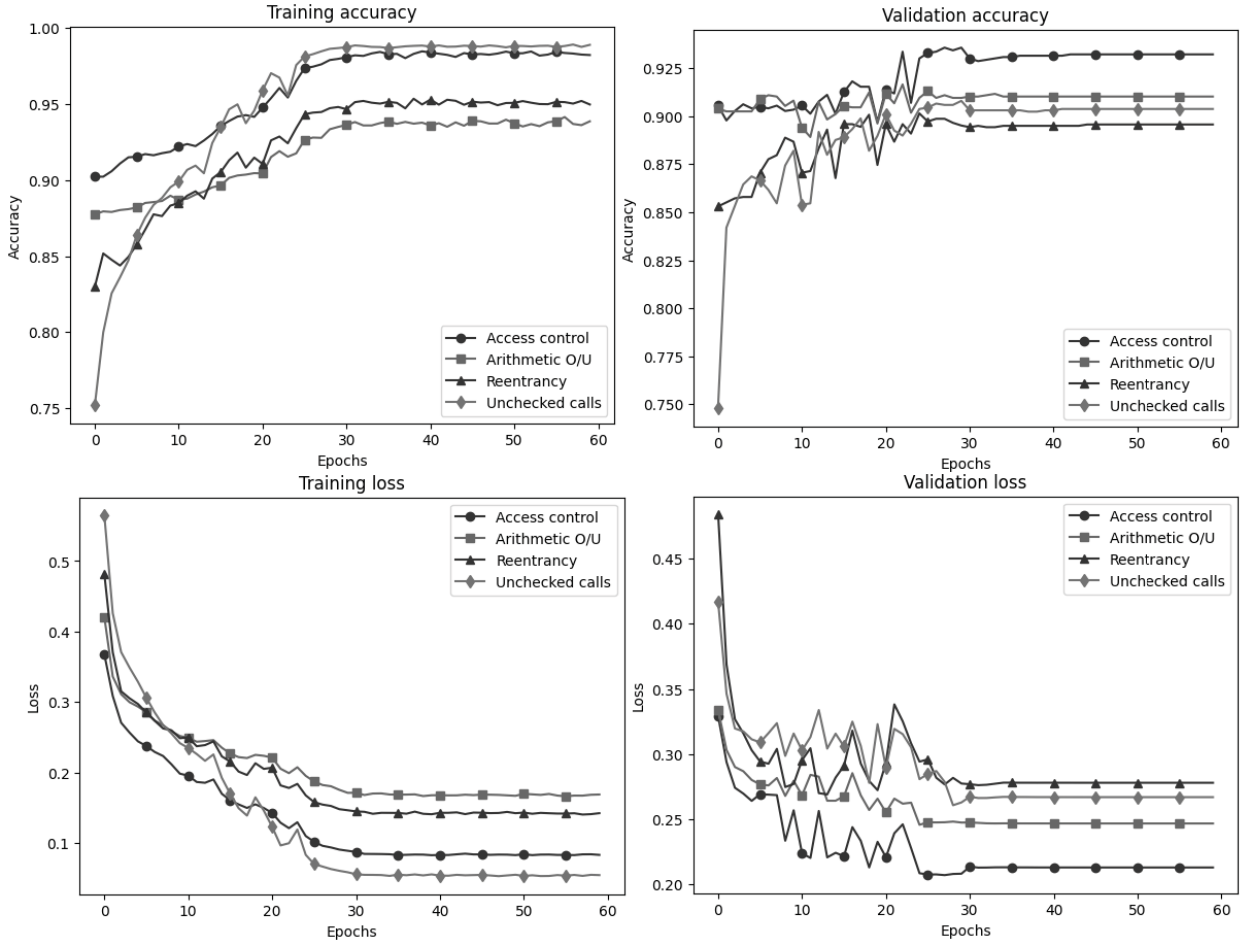

Рис. 10. Результаты обучения финальной модели

Табл. 3

Метрики качества финальной модели

Метрика/ Уязвимость	Access control	Arithmetic O/U	Reentrancy	Unchecked calls
Accuracy	0.937	0.914	0.908	0.902
Precision	0.936	0.911	0.907	0.902
Recall	0.937	0.914	0.908	0.902
F1-score	0.936	0.912	0.908	0.902

также их метрики качества из соответствующей статьи[27]. Результаты сравнения представлены в табл. 4.

Практически по всем показателем разработанный инструмент превосходит уже зарекомендовавшие себя анализаторы уязвимостей.

Заключение

В представленном исследовании был разработан метод поиска уязвимостей в смарт-контрактах на основе машинного обучения и проведен ряд экспериментов, который использовал начальный датасет crypto_ethereum.

Для использования датасета для обучения был проведен этап подготовки данных. В результате на подготовленном датасете было обучено 3 модели. После обучения каждой модели последовательно проводилось их «утяжеление», пока не были достигнуты изначально установленные критерии эффективности ее применения.

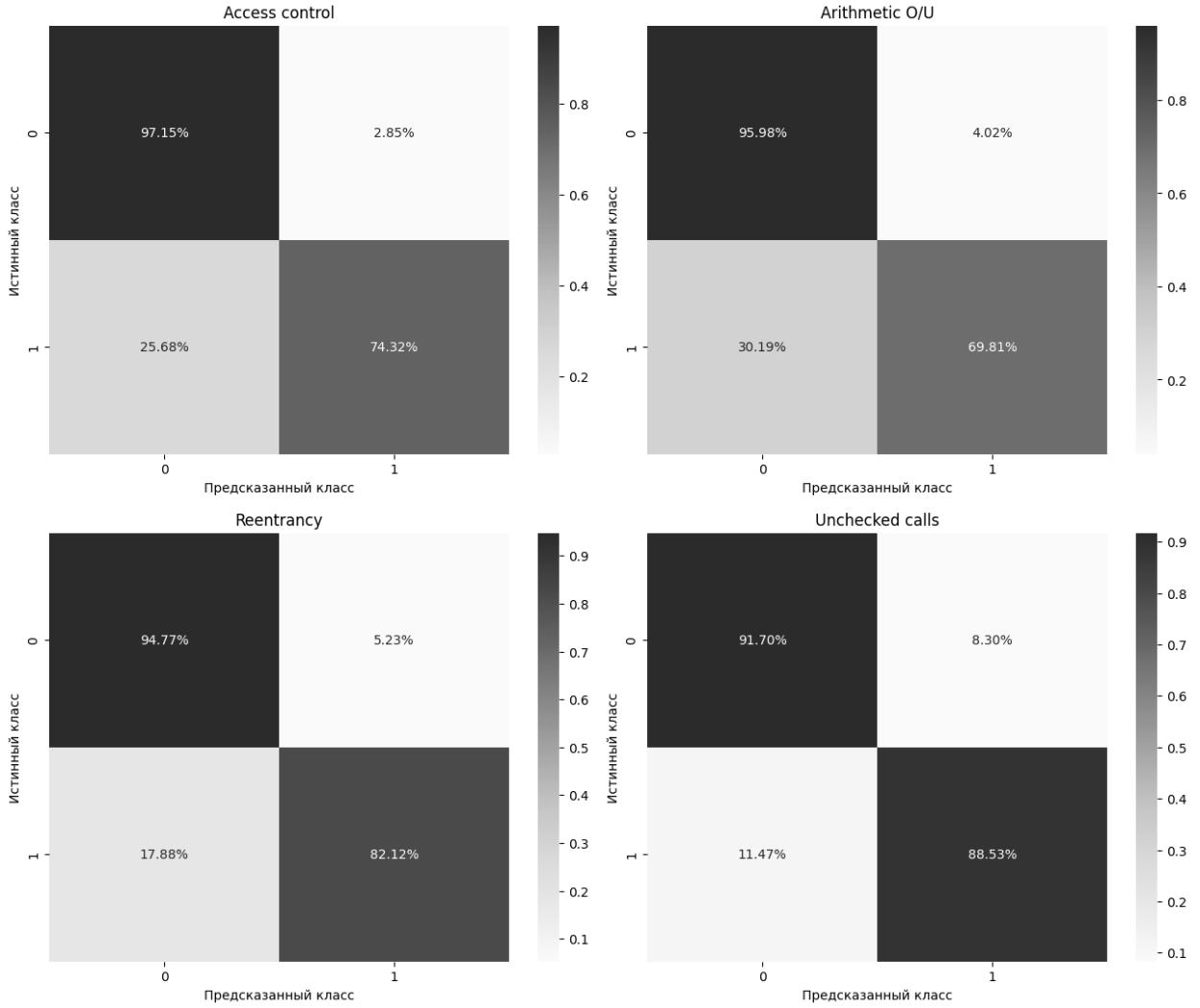

Рис. 11. Матрица ошибок финальной модели

Табл. 4

Сравнение разработанной модели с статическими инструментами

Уязвимость	Метрика/Критерий	Инструмент		
		Mythirl	sFuzz	Разработанная модель
1	2	3	4	5
Reentrancy	Accuracy	70.4%	34.7%	90.84%
	Precision	100%	100%	90.75%
	Recall	67.2%	21.1%	90.84%
	F1-score	78.2%	34.7%	90.76%
Unchecked calls	Accuracy	87%	74%	90.22%
	Precision	100%	100%	90.22%
	Recall	82.7%	65.4%	90.22%
	F1-score	90.5%	79.1%	90.22%
Arithmetic O/U	Accuracy	58.5%	47.6%	91.45%
	Precision	100%	100%	91.15%
	Recall	47.7%	33.8%	91.45%
	F1-score	64.6%	50.6%	91.25%
Access control	Accuracy	77.8%	N/A%	93.75%
	Precision	100%	N/A%	93.56%
	Recall	40%	N/A %	93.75%
	F1-score	57.1%	N/A %	93.62%
Среднее время обнаружения уязвимостей		609 секунд	289 секунд	0.22 секунды

Сравнительный анализ с другими известными анализаторами уязвимостей показал эффективность машинного обучения в решении задачи классификации уязвимостей смарт-контрактов. Важным преимуществом описанного подхода является время обнаружения уязвимостей в процессе работы, что доказано экспериментально.

В будущем планируется провести подробный сравнительный анализ разработанных моделей обучения с последними исследованиями в этой области, а также апробировать его применение на реальных данных смарт-контрактов основной сети Ethereum.

Литература

1. *Ray P.P.* Web3: A comprehensive review on background, technologies, applications, zero-trust architectures, challenges and future directions // Internet of Things and Cyber-Physical Systems. 2023.
2. Crypto Hacks 2023: Full List of Scams and Exploits as Millions Go Missing. // ccn.com. 2024. URL: https://www.ccn.com/education/crypto-hacks-2023-full-list-of-scams-and-exploits-as-millions-go-missing
3. *Huang Y. et al.* Smart contract security: A software lifecycle perspective //IEEE Access. 2019. Vol. 7. P. 150184-150202.
4. *Kiani R., Sheng V.S.* Ethereum Smart Contract Vulnerability Detection and Machine Learning-Driven Solutions: A Systematic Literature Review //Electronics. 2024. Vol. 13. No 12. P. 2295.
5. *Mukhopadhyay M.* Ethereum Smart Contract Development: Build blockchain-based decentralized applications using solidity. Packt Publishing Ltd. 2018.
6. What Are Smart Contracts and How Do They Work? // chain.link. 2023. URL: https://chain.link/education/smart-contracts (дата обращения: 17.11.2023)
7. *Wei Z., Sun J., Zhang Z., Zhang X., Yand X., Zhu L.* Survey on Quality Assurance of Smart Contracts. // ACM Comput. Surv. 2023. URL: https://arxiv.org/pdf/2311.00270.pdf (дата обращения: 17.11.2023)
8. *Harz D., Knottenbelt W.* Towards Safer Smart Contracts: A Survey of Languages and Verification Methods. // arXiv:1809.09805. 2018. URL: https://arxiv.org/pdf/1809.09805.pdf (дата обращения: 17.11.2023)
9. *Brousmiche K., Abdellatif T.* Formal Verification of Smart Contracts Based on Users and Blockchain Behaviors Models. // 9th IFIP International Conference on New Technologies, Mobility and Security. 2018. URL: https://www.researchgate.net/publication/324175498_Formal_Verification_of_Smart_Contracts_Based_on_Users_and_Blockchain_Behaviors_Models (дата обращения: 17.11.2023)
10. *He J., Balunovic M., Ambroladze N., Tsankov P., Martin T.* Learning to Fuzz from Symbolic

Execution with Application to Smart Contracts. // In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019. URL: https://dl.acm.org/doi/10.1145/3319535.3363230 (дата обращения: 17.11.2023)
11. *Click C., Paleczny M.* A simple graph-based intermediate representation //ACM Sigplan Notices. 1995. Vol. 30. No 3. P. 35-49.
12. *Chakraborty S., Krishna R., Ding Y., Ray B.* Deep Learning Based Vulnerability Detection: Are We There Yet? // IEEE Transactions on Software Engineering. 2022. URL: https://www.researchgate.net/publication/352279734_Deep_Learning_based_Vulnerability_Detection_Are_We_There_Yet (дата обращения: 17.11.2023)
13. Multilabel Classification: An Introduction with Python's Scikit-Learn. // KDnuggets. 2023. URL: https://www.kdnuggets.com/2023/08/multilabel-classification-introduction-python-scikitlearn.html (дата обращения: 17.11.2023)
14. Gated Recurrent Unit Networks. // geeksforgeeks. 2023. URL: https://www.geeksforgeeks.org/gated-recurrent-unit-networks/ (дата обращения: 12.01.2024)
15. Crash Course in Convolutional Neural Networks for Machine Learning // machinelearningmastery. 2023. URL: https://machinelearningmastery.com/crash-course-convolutional-neural-networks/ (дата обращения: 12.01.2024)
16. *Zhuang Y. et al.* Smart contract vulnerability detection using graph neural networks // Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence. 2021.P. 3283-3290.
17. From cloud data warehouse to an AI-ready data platform. // Google BigQuery. 2024. URL: https://cloud.google.com/bigquery?hl=ru (дата обращения: 12.01.2024)
18. *Sendner C., Chen H., Fereidooni H., Petzi L., König J., Stang J., Dmitrienko A.* Smarter Contracts: Detecting Vulnerabilities in Smart Contracts with Deep Transfer Learning // 2023. URL: https://www.ndss-symposium.org/wp-content/uploads/2023/02/ndss2023_s263_paper.pdf (дата обращения: 17.11.2023)
19. *Zouhar V., Meister C., Gastaldi J., Du L., Sachan M., Cotterell R.* Tokenization and the Noiseless Channel. // 2023.acl-long.284. 2023. URL: https://aclanthology.org/2023.acl-long.284 (дата обращения: 17.11.2023)
20. Text Vectorization layer. // TensorFlow. 2023. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization (дата обращения: 17.11.2023)
21. Embedding layer. // TensorFlow. 2023. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding (дата обращения: 17.11.2023)
22. *Шолле Ф.* Глубокое обучение на Python. СПб.: Питер. 2018. 400 с.
23. *Жерон О.* Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем. СПб.: ООО «Диалектика». 2020. 1040 с.
24. Optimizers Adam. // TensorFlow. 2024. URL: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam (дата обращения: 30.03.2024).
25. Mythril. Security analysis tool for EVM bytecode // GitHub. 2024. URL: https://github.com/Consensys/mythril (дата обращения: 30.03.2024)
26. sFuzz. // GitHub. 2024. URL: https://github.com/duytai/sFuzz (дата обращения: 30.03.2024)
27. *Wei Z., Sun J., Zhang Z., Zhang X., Li M., Zhu L.* A Comparative Evaluation of Automated Analysis Tools for Solidity Smart Contracts. // arXiv:2310.20212v. 2023. URL: https://arxiv.org/pdf/2310.20212 (дата обращения: 17.11.2023)

Тарханов Иван Александрович. Федеральный исследовательский центр «Информатика и управление» Российской академии наук, г. Москва, Россия. Старший научный сотрудник. Кандидат технических наук, доцент. Область научных интересов: электронный документооборот, блокчейн, информационная безопасность. E-mail: tarkhanov@isa.ru (Ответственный за переписку).

Белоус Вадим Сергеевич. Национальный исследовательский технологический университет «МИСиС», г. Москва, Россия. Магистр. Область научных интересов: машинное обучение, блокчейн.
E-mail: belous.vadim@inbox.ru

Search for vulnerabilities in smart contracts based on machine learning

V.S Belous[I], I.A. Tarkhanov[II,III]

[I] National University of Science and Technology "MISiS", Moscow, Russia

[II] State Academic University for Humanities, Moscow, Russia

[III] Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, Moscow, Russia

Abstract. With the rising popularity of blockchain projects, the number of decentralized applications based on them is also growing. The central element of these applications are smart contracts. This technology is still relatively new and has a number of security issues. Statistics of smart contract hacking indicate the relevance of finding vulnerabilities in smart contract code problem. The article describes 3 machine learning models for searching for vulnerabilities in smart contracts written in the Solidity language. Particular attention is paid to preparing the dataset for training and comparing it with well-known code analyzers. The metrics obtained from the results of training and testing the models suggest that the model consisting of three bidirectional recurrent BiGRU layers and three convolutional CNN layers is effective in the task of searching for smart contract vulnerabilities.

Keyword: *vulnerability, smart contract, machine learning.*
DOI: 10.14357/20790279240310 **EDN: URZLYI**

References

1. *Ray P.P.* Web3: A comprehensive review on background, technologies, applications, zero-trust architectures, challenges and future directions // Internet of Things and Cyber-Physical Systems. 2023.
2. Crypto Hacks 2023: Full List of Scams and Exploits as Millions Go Missing. // ccn.com. 2024. URL: https://www.ccn.com/education/crypto-hacks-2023-full-list-of-scams-and-exploits-as-millions-go-missing
3. *Huang Y. et al.* Smart contract security: A software lifecycle perspective //IEEE Access. 2019. V. 7. P. 150184-150202.
4. *Kiani R., Sheng V.S.* Ethereum Smart Contract Vulnerability Detection and Machine Learning-Driven Solutions: A Systematic Literature Review //Electronics. 2024. V. 13. No. 12. P. 2295.
5. *Mukhopadhyay M.* Ethereum Smart Contract Development: Build blockchain-based decentralized applications using solidity. Packt Publishing Ltd, 2018.
6. What Are Smart Contracts and How Do They Work? // chain.link. 2023. URL: https://chain.link/education/smart-contracts (дата обращения: 17.11.2023)
7. *Wei Z., Sun J., Zhang Z., Zhang X., Yand X., Zhu L.* Survey on Quality Assurance of Smart Contracts. // ACM Comput. Surv. 2023. URL: https://arxiv.org/pdf/2311.00270.pdf (дата обращения: 17.11.2023)
8. *Harz D., Knottenbelt W.* Towards Safer Smart Contracts: A Survey of Languages and Verification Methods. // arXiv:1809.09805. 2018. URL: https://arxiv.org/pdf/1809.09805.pdf (дата обращения: 17.11.2023)
9. *Brousmiche K., Abdellatif T.* Formal Verification of Smart Contracts Based on Users and Blockchain Behaviors Models. // 9th IFIP International Conference on New Technologies, Mobility and Security. 2018. URL: https://www.researchgate.net/publication/324175498_Formal_Verification_of_Smart_Contracts_Based_on_Users_and_Blockchain_Behaviors_Models (дата обращения: 17.11.2023)
10. *He J., Balunovic M., Ambroladze N., Tsankov P., Martin T.* Learning to Fuzz from Symbolic Execution with Application to Smart Contracts. // In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019. URL: https://dl.acm.org/doi/10.1145/3319535.3363230 (дата обращения: 17.11.2023)
11. *Click C., Paleczny M.* A simple graph-based intermediate representation //ACM Sigplan Notices. 1995. V. 30. No. 3. P. 35-49.
12. *Chakraborty S., Krishna R., Ding Y., Ray B.* Deep Learning Based Vulnerability Detection: Are We There Yet? // IEEE Transactions on Software Engineering. 2022. URL: https://www.researchgate.net/publication/352279734_Deep_Learning_based_Vulnerability_Detection_Are_We_There_Yet (дата обращения: 17.11.2023)
13. Multilabel Classification: An Introduction with Python's Scikit-Learn. // KDnuggets. 2023. URL: https://www.kdnuggets.com/2023/08/multilabel-classification-introduction-python-scikitlearn.html (дата обращения: 17.11.2023)
14. Gated Recurrent Unit Networks. // geeksforgeeks. 2023. URL: https://www.geeksforgeeks.org/

gated-recurrent-unit-networks/ (дата обращения: 12.01.2024)
15. Crash Course in Convolutional Neural Networks for Machine Learning // machinelearningmastery. 2023. URL: https://machinelearningmastery.com/crash-course-convolutional-neural-networks/ (дата обращения: 12.01.2024)
16. *Zhuang Y. et al.* Smart contract vulnerability detection using graph neural networks // Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence. 2021. P. 3283-3290.
17. From cloud data warehouse to an AI-ready data platform. // Google BigQuery. 2024. URL: https://cloud.google.com/bigquery?hl=ru (дата обращения: 12.01.2024)
18. *Sendner C., Chen H., Fereidooni H., Petzi L., König J., Stang J., Dmitrienko A.* Smarter Contracts: Detecting Vulnerabilities in Smart Contracts with Deep Transfer Learning // ISBN 1-891562-83-5. 2023. URL: https://www.ndss-symposium.org/wp-content/uploads/2023/02/ndss2023_s263_paper.pdf (дата обращения: 17.11.2023)
19. *Zouhar V., Meister C., Gastaldi J., Du L., Sachan M., Cotterell R.* Tokenization and the Noiseless Channel. // 2023.acl-long.284. 2023. URL: https://aclanthology.org/2023.acl-long.284 (дата обращения: 17.11.2023)
20. Text Vectorization layer. // TensorFlow. 2023. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization (дата обращения: 17.11.2023)
21. Embedding layer // TensorFlow. 2023. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding (дата обращения: 17.11.2023)
22. *Sholle F.* Glubokoe obuchenie na Python. SPb.: Piter. 2018. 400 p.
23. *Zheron O.* Prikladnoe mashinnoe obuchenie s pomoshh'ju Scikit-Learn, Keras i TensorFlow: koncepcii, instrumenty i tehniki dlja sozdanija intellektual'nyh sistem. SPb.: OOO «Dialektika», 2020.
24. Optimizers Adam. // TensorFlow. 2024. URL: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam (дата обращения: 30.03.2024).
25. Mythril. Security analysis tool for EVM bytecode // GitHub. 2024. URL: https://github.com/Consensys/mythril (дата обращения: 30.03.2024)
26. sFuzz. // GitHub. 2024. URL: https://github.com/duytai/sFuzz (дата обращения: 30.03.2024)
27. *Wei Z., Sun J., Zhang Z., Zhang X., Li M., Zhu L.* A Comparative Evaluation of Automated Analysis Tools for Solidity Smart Contracts. // arXiv:2310.20212v. 2023. URL: https://arxiv.org/pdf/2310.20212 (дата обращения: 17.11.2023)

Tarkhanov I.A. PhD, State Academic University for Humanities, Moscow, Russia, e-mail: tarkhanov@isa.ru

Belous V.S. Student, National University of Science and Technology "MISiS", Moscow, Russia, e-mail: belous.vadim@inbox.ru