****МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

**Институт № 4 «Радиоэлектроника, инфокоммуникации и информационная безопасность» Кафедра 402 Группа М4О-412Б-20**

**Направление подготовки 10.03.01**

**Профиль Информационная безопасность**

**Квалификация бакалавр**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**БАКАЛАВРА**

На тему: Разработка статического анализатора уязвимостей смарт-контрактов для обнаружения и парирования потенциальных уязвимостей в системе передачи данных на основе блокчейна Ethereum

Автор ВКРБ \_\_ \_ Буров Тимур Альбертович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество полностью) (подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_Карпухин Евгений Олегович\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество полностью) (подпись)

Консультант\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_Джум Владимир Сергеевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество полностью) (подпись)

**К защите допустить**

Заведующий кафедрой 402\_\_ \_\_\_Мазепа Роман Богданович\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

(№ каф.) (фамилия, имя, отчество полностью) (подпись)

\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_г.

Москва 2024

Оглавление

[Введение 4](#_Toc166692363)

[1. Анализ актуальных уязвимостей смарт-контрактов в системе передачи данных на основе блокчейна Ethereum 6](#_Toc166692364)

[1.1 Терминология blockchain Ethereum 6](#_Toc166692365)

[1.2 Принцип работы blockchain 7](#_Toc166692366)

[1.3 Отличительная особенность blockchain Ethereum 13](#_Toc166692367)

[1.4 Преимущества и недостатки смарт-контрактов 23](#_Toc166692368)

[1.5 Анализ актуальных уязвимостей смарт-контрактов в системе передачи данных блокчейна Ethereum 25](#_Toc166692369)

[1.6 Меры профилактики некоторых уязвимостей смарт-контрактов 29](#_Toc166692370)

[1.7 Вывод 32](#_Toc166692371)

[2. Оценка эффективности использования существующих средств анализа уязвимостей смарт-контрактов 33](#_Toc166692372)

[2.1 Существующие средства анализа уязвимостей смарт-контрактов 33](#_Toc166692373)

[2.2 Оценка эффективности использования анализаторов; 36](#_Toc166692374)

[2.3 Вывод 41](#_Toc166692375)

[3. Разработка и внедрение анализатора уязвимостей смарт-контрактов. 42](#_Toc166692376)

[3.1 Введение в нейронные сети и глубокое обучение 42](#_Toc166692377)

[3.2 Задача представления признаков смарт-контрактов 47](#_Toc166692378)

[3.3 Принцип работы нейросети архитектуры трансформер 55](#_Toc166692379)

[3.4 Создание анализатора уязвимостей с использованием архитектуры трансформер 65](#_Toc166692380)

[3.5 Вывод 69](#_Toc166692381)

[4. Демонстрация работоспособности разработанного анализатора уязвимостей смарт-контрактов 70](#_Toc166692382)

[4.1 Тестирование разработанного анализатора смарт-контрактов на поиск уязвимых смарт-контрактов 70](#_Toc166692383)

[4.2 Метрики оценивания качества модели 70](#_Toc166692384)

[4.3 Результаты оценивания работоспособности разработанной модели 75](#_Toc166692385)

[4.4 Вывод 77](#_Toc166692386)

[Заключение 78](#_Toc166692387)

[Список используемой литературы 79](#_Toc166692388)

Введение

В современном мире технологии blockchain продолжают привлекать внимание как в академических кругах, так и в индустрии из-за своих уникальных свойств в области безопасности, децентрализации и прозрачности. Одним из наиболее значимых нововведений в блокчейн-технологиях является концепция умных контрактов, особенно на платформе Ethereum. Умные контракты — это программы, которые автоматически выполняют, контролируют или документируют юридически значимые события и действия согласно условиям контракта, что исключает необходимость в посредниках. [1]

Однако, несмотря на их эффективность и автоматизацию, умные контракты также подвержены разнообразным уязвимостям. Эти уязвимости могут быть эксплуатированы злоумышленниками, что приводит к значительным финансовым потерям. Например, возможно, самый известный случай, который привел к большим потерям произошел в 2016 году. The DAO была децентрализованной автономной организацией, созданная на платформе Ethereum. Злоумышленник смог эксплуатировать уязвимость в рекурсивных вызовах функций смарт-контракта, что позволило ему вывести около 3,6 миллиона ETH, что на тот момент составляло примерно 50 миллионов долларов США. Еще инцидент затронул Uniswap, популярный децентрализованный обменник. Злоумышленник использовал уязвимость повторного входа для кражи 300 000 долларов в Ethereum, и таких случаев огромное количество.

С учетом указанных проблем, важность создания инструментов для анализа уязвимостей умных контрактов не может быть переоценена. В данной дипломной работе рассматривается разработка и внедрение анализатора уязвимостей умных контрактов, который использует передовые методы машинного обучения и нейронных сетей для автоматизации и усиления процессов проверки и обеспечения безопасности кода.

Целью данной работы является разработка программного обеспечения, способного эффективно определять потенциальные уязвимости в коде умных контрактов. Для достижения этой цели предлагается использовать алгоритмы машинного обучения, которые могут анализировать, интерпретировать и обучаться на результатах больших данных, выбирая из них закономерности, которые могут указывать на наличие уязвимостей.[2]

Дипломный проект включает в себя исследование существующих методик анализа уязвимостей, выбор и подготовку соответствующих технологий машинного обучения, разработку прототипа системы и его тестирование на примерах реальных умных контрактов. Итоговый продукт данной работы может использоваться как дополнительное средство анализа умных контрактов, что значительно улучшить их текущее состояние безопасности.

1. **Анализ актуальных уязвимостей смарт-контрактов в системе передачи данных на основе блокчейна Ethereum**

Ethereum является одной из наиболее известных и широко используемых блокчейн-платформ, появившейся в 2015 году благодаря усилиям разработчика Виталика Бутерина и его команды. Основным отличием Ethereum от первого поколения блокчейн-систем, таких как Bitcoin, является внедрение концепции смарт-контрактов, что делает эту платформу не просто системой для транзакций криптовалюты, но и универсальной основой для создания различного рода распределенных приложений.

1.1 Терминология blockchain Ethereum

Ethereum Virtual Machine (EVM) - виртуальная машина, являющаяся исполняющей средой для смарт-контрактов в Ethereum. EVM изолирует смарт-контракты от остальной сети, обеспечивая безопасность и надежность.

Смарт-контракт - код, размещенный на Ethereum blockchain, который автоматически выполняет, управляет или документирует юридически значимые события и действия согласно условиям контракта.

Газ - единица измерения, используемая для расчета количества вычислительной работы, необходимой для выполнения операций, таких как транзакции или выполнение смарт-контрактов. Газ необходим для предотвращения спама и ограничения ресурсов сети. Каждая операция в Ethereum, будь то простая транзакция или выполнение сложного смарт-контракта, требует определенного количества газа.[3]

Gas Limit - максимальное количество газа, которое пользователь готов потратить на транзакцию или выполнение смарт-контракта. Это помогает контролировать затраты на использование ресурсов сети.

Gas Price - Цена газа отражает, сколько Ether (ETH) пользователь готов заплатить за каждую единицу газа. Цена газа измеряется в Gwei, где 1 Gwei равен 0.000000001 ETH. Цена газа варьируется в зависимости от загруженности сети; чем больше транзакций обрабатывается в сети, тем выше может быть цена газа

Ether (ETH) - криптовалюта Ethereum, используемая в качестве цифрового топлива для сети Ethereum. Используется для оплаты транзакций и услуг в рамках сети Ethereum. Цена 1 Ether (ETH) на момент написания работы составляет порядка 3000 долларов США.

Транзакция – это отдельная операция в блокчейне от имени участника, изменяющая стейт сети. Отправляя ту или иную транзакцию, участник отправляет в сеть запрос с набором данных, необходимых для соответствующего изменения стейта.

Децентрализованное приложение (dApp) - приложение, которое действует автономно без постороннего контроля. Оно строится и работает на блокчейн-платформе, часто используя смарт-контракты для обработки данных.

Token (Токен) - цифровой актив, создаваемый и управляемый через смарт-контракты на Ethereum. Токены обычно выполняют роль цифровых активов или активов, представляющих доступ к услугам или функциям приложения.

ERC-20 - стандарт токена на Ethereum, который определяет общий список правил для Ethereum токенов, таких как передача токенов и доступ к данным о токенах, что позволяет различным токенам взаимодействовать друг с другом.

Хэш-функция – это функция, преобразующая массив входных данных произвольного размера в выходную битовую строку определённого (установленного) размера в соответствии с определённым алгоритмом

Keccak256 — это криптографическая хеш-функция, используемая в языке программирования Solidity для смарт-контрактов Ethereum.

* 1. Принцип работы blockchain

Blockchain - это распределенная база данных, которая используется для хранения и передачи информации. Она состоит из блоков, которые содержат информацию о транзакциях и других данных, а также хеш-коды предыдущих блоков. Блоки связаны между собой цепочкой, что обеспечивает целостность и безопасность системы. [4]

Блокчейн был создан для усиления безопасности и обеспечения прозрачности в транзакциях с криптовалютами, такими как Bitcoin. Эта технология также применима в различных других сферах, включая банковский сектор и государственные структуры.

Одним из ключевых достоинств блокчейна является его децентрализованный характер: данные распределены по множеству узлов, вместо хранения на одном центральном сервере, что повышает защиту от кибератак и технических неисправностей.

Блокчейн также гарантирует прозрачность и защиту данных от фальсификаций, делая его отличным выбором для использования в управлении цепями поставок, выборах и других сферах, где важны прозрачность и надежность.

Принципы работы блокчейна базируются на распределенном хранении информации и подтверждении транзакций всеми участниками сети. У каждого участника есть своя копия базы данных, которая сохранена на его компьютере, и каждый может участвовать в проверке транзакций. Это создает надежную и защищенную систему для хранения и передачи данных, применяемую в широком спектре отраслей, включая финансы, электронную коммерцию и здравоохранение.

При добавлении новой транзакции в сеть, она распространяется среди всех участников, которые проверяют ее на соответствие установленным правилам и достоверность. Если транзакция проходит проверку, она включается в новый блок, который связывается с предыдущим блоком через хеш-значение, создавая таким образом цепочку.

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

Рисунок 1. Демонстрация Blockchain

Каждый блок содержит не только информацию о транзакции, но и хеш предыдущего блока, гарантируя таким образом непрерывность цепи и защиту данных в блокчейне от изменений.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2 – Попытка изменить данные в блоке

Если злоумышленник пытается изменить информацию в любом из блоков, хеш этого блока изменится. В блокчейне каждый блок содержит хеш предыдущего блока, формируя таким образом цепочку. Изменение хеша одного блока разрушит цепочку. Поскольку каждый блок включает в себя хеш предыдущего блока, изменение данных в одном блоке делает все последующие блоки недействительными, так как их хеши больше не совпадают с актуальными данными. Таким образом, блокчейн отображает ошибку целостности. В Ethereum в качестве метода формирования блока используется концепция Proof-of-stake

Proof-of-stake (PoS) метод защиты в криптовалютах, при котором вероятность формирования участником очередного блока в блокчейне пропорциональна доле, которую составляют принадлежащие этому участнику расчётные единицы данной криптовалюты от их общего количества.

Данный метод является альтернативой методу подтверждения выполнения работы (PoW), при котором вероятность создания очередного блока выше у обладателя более мощного оборудования

В доказательстве доли Ethereum валидаторы явно вносят капитал в форме ETH в смарт-контракт на Ethereum. Затем валидатор отвечает за проверку правильности новых блоков, распространяемых по сети, а также иногда сам создает и распространяет новые блоки. Если они попытаются обмануть сеть (например, предложив несколько блоков, хотя они должны отправить один, или отправив конфликтующие подтверждения), часть или все их поставленные ETH могут быть уничтожены.

В доказательстве доли Ethereum валидаторы явно вносят капитал в форме ETH в смарт-контракт на Ethereum. Затем валидатор отвечает за проверку правильности новых блоков, распространяемых по сети, а также иногда сам создает и распространяет новые блоки. Если они попытаются обмануть сеть (например, предложив несколько блоков, хотя они должны отправить один, или отправив конфликтующие подтверждения), часть или все их поставленные ETH могут быть уничтожены.

Ниже приводится подробное объяснение того, как транзакция выполняется в Ethereum Proof-of-Stake.

Пользователь создает и подписывает транзакцию своим закрытым ключом. Обычно это обрабатывается кошельком или библиотекой, такой как ether.js.(откроется в новой вкладке), веб3js(откроется в новой вкладке), веб3пи(откроется в новой вкладке)и т. д., но под капотом пользователь делает запрос к узлу, используя API Ethereum JSON-RPC. Пользователь определяет количество газа, которое он готов заплатить в качестве чаевых валидатору, чтобы побудить его включить транзакцию в блок. Чаевые выплачиваются валидатору, а базовый сбор сжигается.

Транзакция передается клиенту исполнения Ethereum, который проверяет ее действительность. Это означает, что у отправителя достаточно ETH для выполнения транзакции и что он подписал ее правильным ключом.

Если транзакция действительна, клиент выполнения добавляет ее в свой локальный мемпул (список ожидающих транзакций), а также передает ее другим узлам через сеть сплетен уровня выполнения. Когда другие узлы узнают о транзакции, они тоже добавляют ее в свой локальный мемпул. Опытные пользователи могут воздержаться от трансляции своей транзакции и вместо этого перенаправить ее специализированным создателям блоков, таким как Flashbots Auction.(откроется в новой вкладке). Это позволяет им организовывать транзакции в предстоящих блоках для получения максимальной прибыли ( MEV ).

Один из узлов в сети является предлагающим блок для текущего слота, предварительно выбранным псевдослучайно с помощью RANDAO. Этот узел отвечает за создание и трансляцию следующего блока, который будет добавлен в блокчейн Ethereum, а также за обновление глобального состояния. Узел состоит из трех частей: клиента выполнения, клиента консенсуса и клиента валидатора. Клиент выполнения объединяет транзакции из локального мемпула в «полезную нагрузку выполнения» и выполняет их локально для генерации изменения состояния. Эта информация передается клиенту консенсуса, где полезная нагрузка выполнения заворачивается как часть «блока маяка», который также содержит информацию о вознаграждениях, штрафах, косых чертах, аттестациях и т. д., которые позволяют сети согласовывать последовательность блоков в заголовке. цепи. Связь между клиентами исполнения и консенсусными клиентами более подробно описана в разделе « Соединение клиентов консенсуса и выполнения».

Другие узлы получают новый блок маяков в сети сплетен консенсусного уровня. Они передают его своему исполняющему клиенту, где транзакции повторно выполняются локально, чтобы гарантировать допустимость предлагаемого изменения состояния. Затем клиент-валидатор подтверждает, что блок действителен и является логическим следующим блоком в его представлении о цепочке (это означает, что он строится на цепочке с наибольшим весом аттестаций, как определено в правилах выбора вилки ). Блок добавляется в локальную базу данных на каждом узле, который его подтверждает.

Транзакция может считаться «завершенной», если она стала частью цепочки с «сверхбольшинством звена» между двумя контрольными точками. Контрольные точки возникают в начале каждой эпохи и существуют для того, чтобы учитывать тот факт, что только подмножество активных валидаторов подтверждает каждый слот, но все активные валидаторы подтверждают каждую эпоху. Таким образом, только между эпохами можно продемонстрировать «связь сверхбольшинства» (именно здесь 66% общего количества поставленных ETH в сети согласовываются по двум контрольным точкам).

1.3 Отличительная особенность blockchain Ethereum

Главной ключевых особенностью блокчейна Ethereum является поддержка смарт-контрактов. Смарт-контракты — это программы, которые автоматически выполняют, контролируют или документируют юридически значимые события и действия согласно условиям контракта. Они запускаются в блокчейне и выполняются точно так, как были запрограммированы, без возможности цензуры, мошенничества или вмешательства третьих сторон.

Вот некоторые из основных применений и контекстов использования смарт-контрактов:

1. Финансовые услуги

- Децентрализованные финансы (DeFi): Смарт-контракты являются основой для DeFi приложений, которые предлагают широкий спектр финансовых услуг, включая кредитование, заимствование, страхование и торговлю без участия традиционных финансовых институтов.

- Автоматические выплаты: Смарт-контракты могут автоматически выполнять выплаты или переводы средств, когда выполняются определенные условия, что может быть использовано для регулярных платежей, таких как арендные платежи или выплаты по займам.

2. Цепочки поставок и логистика

Смарт-контракты могут применяться для улучшения прозрачности и эффективности в цепочках поставок. Они могут автоматически отслеживать происхождение товаров, обеспечивать соответствие контрактов о доставке и снижать риски мошенничества.

3. Интеллектуальная собственность и авторские права

Смарт-контракты могут использоваться для автоматизации управления авторскими правами, лицензирования контента и сделок, связанных с интеллектуальной собственностью, обеспечивая при этом прозрачность и справедливое распределение доходов между создателями.

4. Игры и цифровые коллекционные предметы

В индустрии видеоигр и цифровых коллекционных предметов (например, NFTs) смарт-контракты используются для создания и управления уникальными цифровыми активами, которыми могут владеть и торговать пользователи.

5. Идентификация и аутентификация

Смарт-контракты могут играть роль в системах цифровой идентификации, предоставляя надежный механизм для проверки и управления доступом к личной информации или услугам без необходимости раскрытия большего объема личных данных.

6. Биржи и торговые платформы

Смарт-контракты используются для автоматизации и облегчения торговли активами, включая криптовалюты, акции, деривативы и другие финансовые инструменты на децентрализованных биржах.

7. Системы голосования

В голосовании смарт-контракты могут использоваться для создания прозрачных и неподдельных систем голосования, где каждый голос записывается на блокчейн, обеспечивая безопасность данных о голосовании и невозможность их подделки.

8. Страхование

Смарт-контракты могут улучшить и ускорить процессы в страховом секторе, автоматизируя утверждения и выплаты на основе верификации событий, записанных в блокчейне.

Преимущества смарт-контрактов:

- Автоматизация: уменьшение объема ручной работы и ускорение процессов.

- Снижение затрат: сокращение расходов за счет отсутствия необходимости в посредниках.

- Прозрачность: все действия записываются на блокчейн, обеспечивая открытость и доступность информации.

- Безопасность: защита от мошенничества и недопустимости вмешательства благодаря криптографической защите и децентрализации.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 3. Демонстрация работы умных контрактов

Процесс взаимодействия со смарт контрактом заключается в следующем:

Сперва необходимо написать сам контракт. Смарт-контракты пишутся на языке программирования Solidity (или других поддерживаемых языках), которые затем компилируются в байт-код, который Ethereum может исполнять. Solidity — это объектно-ориентированный язык высокого уровня для реализации смарт-контрактов. статически типизирован, поддерживает наследование, библиотеки и сложные пользовательские типы, а также другие функции.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 4 – Пример кода смарт-контракта на языке Solidity

После создания смарт-контракт развертывается в блокчейн Ethereum.

Это делается путем создания транзакции, которая включает скомпилированный код контракта. Для развертывания можно использовать различные среды разработки, например Remix.

**Remix** - это простой и доступный онлайн-редактор и компилятор для Solidity, который позволяет тестировать и развертывать смарт-контракты в тестовых сетях (например, Rinkeby, Ropsten) прямо из браузера

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 5. Пример развертывания смарт-контракта в Remix

Основные особенности Remix:

1. Бесшовная интеграция с Solidity: Remix поддерживает последние версии Solidity и предоставляет удобные средства для написания, компиляции и управления версиями кода.

2. Встроенная виртуальная машина Ethereum (EVM): Remix содержит встроенный эмулятор EVM, который позволяет тестировать смарт-контракты прямо в браузере, не прибегая к использованию реальных сетей или токенов.

3. Отладчик: Важной особенностью Remix является мощный отладчик смарт-контрактов. Он позволяет шаг за шагом проходить по выполнению смарт-контракта, что полезно для выявления ошибок и оптимизации кода.

4. Поддержка тестовых сетей: Remix позволяет подключаться к публичным тестовым сетям Ethereum (например, Rinkeby, Ropsten и Kovan) или к локальным блокчейн-серверам (например, Ganache), что делает возможным реальное взаимодействие с сетью Ethereum в процессе разработки.

5. Плагины: Remix поддерживает плагины, что позволяет расширять его функциональность. Это могут быть инструменты для различных видов анализа кода, интеграция с другими сервисами и многое другое.

6. Графический интерфейс пользователя: IDE предоставляет пользователю удобный графический интерфейс, который включает в себя редактор кода, панель управления транзакциями, а также инструменты для работы с файлами и проектами.

7. Деплой и интеракция с контрактами: Пользователи могут развертывать контракты в различных сетях, а также взаимодействовать с уже развернутыми контрактами, используя встроенный интерфейс.

Как начать работу с Remix

Чтобы начать использовать Remix, достаточно выполнить следующие шаги:

1. Перейти на веб-сайт Remix.

2. Начать новый проект: Создайте новый файл или проект, используя интерфейс Remix, или загрузите существующий.

3. Написание и компиляция кода: Напишите свой смарт-контракт на Solidity и скомпилируйте его, используя компилятор, доступный в Remix.

4. Тестирование контракта: Используйте встроенные инструменты для тестирования, включая симуляцию транзакции и отладчик.

5. Деплой контракта: Разверните свой контракт локально или в одной из тестовых/главных сетей Ethereum, используя встроенные функции Remix.

6. Взаимодействие с контрактом: После развертывания контракта вы можете взаимодействовать с его функциями прямо из Remix.

**Truffle** предоставляет более развитый набор инструментов для разработки с возможностью локального тестирования в среде Ganache и последующего развертывания в сеть. Truffle является одним из самых старых и устоявшихся наборов разработчика для Ethereum.

Основные функции Truffle:

1. Управление проектом

Truffle предоставляет структуру проекта, которая помогает организовать и управлять всеми аспектами разработки смарт-контрактов, включая код контрактов, тесты, скрипты развертывания и конфигурации сети.

1. Компиляция контрактов

Truffle автоматически компилирует смарт-контракты, обеспечивая стабильность и совместимость с сетью Ethereum. Он использует компилятор Solidity (solc) для преобразования кода на Solidity в байткод, который можно развернуть на блокчейне Ethereum.

3. Миграции

Миграции в контексте Truffle – это сценарии развертывания, которые используются для размещения смарт-контрактов на блокчейне. Эта система позволяет управлять последовательностью развертывания и обновлений контрактов, обеспечивая удобное управление версиями и изменениями в процессе разработки.

4.Тестирование

Truffle поддерживает тестирование смарт-контрактов с помощью JavaScript и Solidity. Это позволяет разработчикам написать модульные и интеграционные тесты, которые помогают в поддержании стабильности кода и предотвращении регрессии функций при обновлениях кода.

5. Сети развертывания

Truffle позволяет легко конфигурировать и использовать различные сети блокчейна (как тестовые, так и основные сети), что делает развертывание смарт-контрактов гибким и контролированным. В файле truffle-config.js можно определить параметры для соединения с локальными, тестовыми или живыми сетями Ethereum.

6. Консоль и интерактивный режим

Truffle предоставляет командную консоль, которая позволяет взаимодействовать с смарт-контрактами и Ethereum сетями в интерактивном режиме. Это полезно для быстрого прототипирования и отладки.

7. Интеграция с Metamask

Metamask – это браузерное расширение, которое функционирует как Ethereum кошелек и используется для взаимодействия с блокчейном. Truffle может интегрироваться с Metamask, обеспечивая удобное и безопасное развертывание и тестирование DApps в реальной сетевой среде.

8. Плагины и интеграция

Truffle поддерживает различные плагины и инструменты интеграции, такие как Truffle Teams, Infura и другие, что позволяет расширять его функциональность и интегрировать в широкий спектр инфраструктурных платформ и услуг.

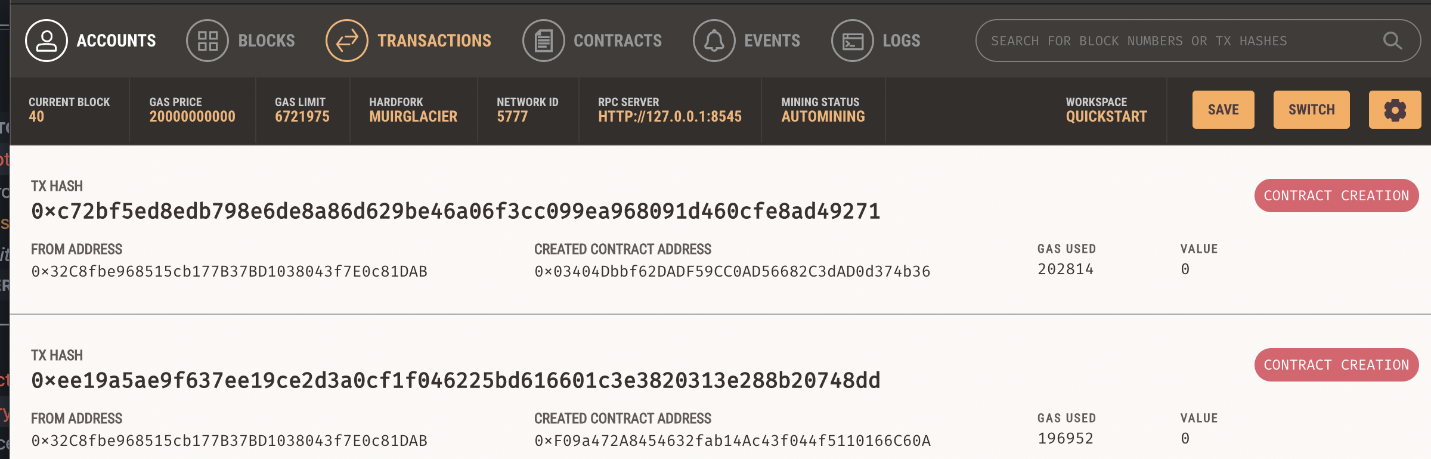


Рисунок 6. Пример развертывания смарт-контракта в Truffle

**Hardhat** — еще одна популярная среда разработки, которая ориентирована на расширенные потребности разработки и тестирования смарт-контрактов. Hardhat сравнительно новее и быстро набирает популярность из-за своей гибкости и простоты использования

Локальная разработческая среда (EVM): Hardhat включает в себя встроенную симулированную среду Ethereum Virtual Machine (EVM), которая позволяет выполнять транзакции и вызывать смарт-контракты, не выходя из локальной разработческой среды.

2. Автоматическое управление аккаунтами: Hardhat управляет Ethereum-аккаунтами за вас. Это упрощает взаимодействие с смарт-контрактами во время разработки и тестирования.

3. Solidity стек-трейсы: Hardhat автоматически представляет стек вызовов ошибок в Solidity, что значительно упрощает отладку и устранение неполадок.

4. Плагинная архитектура: Hardhat поддерживает широкий спектр плагинов, что позволяет расширять его функциональность. Например, можно использовать плагины для интеграции с другими инструментами, такими как ethers.js, Waffle, TypeChain и др.

5. Сценарии развертывания: Hardhat позволяет писать скрипты для систематического развертывания смарт-контрактов, что может быть полезно для сложных развертываний или развертываний в продакшен.

6. Тестирование: Hardhat облегчает написание, выполнение и отладку модульных и интеграционных тестов для ваших смарт-контрактов, используя JavaScript или TypeScript.

Для начала работы с Hardhat требуется следующее:

1. Установка Node.js: Hardhat требует Node.js. Рекомендуется использовать последнюю стабильную версию.

2. Создание проекта Hardhat.

3. Инициирование проекта Hardhat: Появится интерактивное руководство, которое поможет создать базовую конфигурацию проекта.

4. Написание смарт-контрактов: После создания проекта можно начать писать смарт-контракты в директории contracts.

5. Компиляция смарт-контракта:

6. Написание тестов: Тесты помогают убедиться, что смарт-контракты работают как ожидается. Тесты пишутся в директории test.

7. Запуск тестов.

8. Развертывание смарт-контрактов: Развертывание контрактов на локальной EVM или на тестовой сети Ethereum.

Hardhat также поддерживает продвинутые настройки, такие как конфигурация компилятора Solidity, настройка сетей и многое другое, которые могут быть настроены в файле hardhat.config.js.



Рисунок 7. Пример развертывания смарт-контракта в Hardhat

После выполнения транзакции перейдем по ссылке "view on Etherscan" и ознакомьтесь со всеми деталями транзакции.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 8. Страница контракта в EtherScan

На открывшейся странице EtherScan видны данные о развертывании нашего контракта. В каком блоке транзакция, адрес создателя (адрес нашего кошелька), сколько заплатили за транзакцию, и другое. Далее нам надо верифицировать смарт-контракт.

На открывшейся странице сайта EtherScan представлена информация о нашем контракте, которая включаем номер блока, в котором произошла транзакция, адрес создателя (соответствующий адресу нашего кошелька), стоимость транзакции и другие данные.

* 1. Преимущества и недостатки смарт-контрактов

Смарт-контракты, являясь одной из ключевых технологий, используемых в блокчейне, предоставляют множество преимуществ, но также имеют и некоторые недостатки. Давайте рассмотрим их подробнее:  
Преимущества смарт-контрактов:

1. Автоматизация процессов: Смарт-контракты автоматически выполняют условия сделки как только подтверждаются входные данные, уменьшая необходимость в посредниках и сокращая время выполнения.

2. Снижение затрат: Устранение посредников и автоматизация сделок помогают снизить операционные издержки.

3. Прозрачность: Каждое действие записывается в блокчейне, обеспечивая полную прозрачность операций для всех участников.

4. Безопасность: Блокчейн технология делает смарт-контракты крайне сложными для взлома благодаря криптографической защите и распределённому характеру хранения данных.

5. Надежность: Один раз развернутые, смарт-контракты не могут быть изменены, предотвращения мошенничество или вмешательство.

6. Скорость транзакций: Транзакции с помощью смарт-контрактов обрабатываются быстрее традиционных юридических процедур благодаря цифровизации процессов.

Недостатки смарт-контрактов:

1. Ошибки в коде: Ошибки или уязвимости в написании кода смарт-контрактов могут привести к серьезным финансовым потерям или эксплуатации контракта.

2. Юридическая неопределённость: Недостаточная или непостоянная правовая регулировка смарт-контрактов может привести к юридическим проблемам и неопределенности.

3. Затраты на газ: Для выполнения смарт-контрактов на платформах вроде Ethereum необходимо оплачивать газ, что может быть дорого в периоды высокого спроса.

4. Технические барьеры: Создание и взаимодействие со смарт-контрактами требует технических знаний, что может быть барьером для некоторых организаций.

1.5 Анализ актуальных уязвимостей смарт-контрактов в системе передачи данных блокчейна Ethereum

Проблема безопасности смарт-контрактов является одной из основных уязвимостей в системах передачи данных на основе блокчейна Ethereum.[5] Ниже приведены некоторые основные уязвимости, которые могут быть использованы злоумышленниками:

1. Reentrancy уязвимость: Это тип уязвимости, когда злоумышленник использует функцию смарт-контракта для повторного вызова другой функции перед завершением выполнения первой. Это может привести к потере средств или изменению состояния контракта.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 9. Reentrancy уязвимость

Суть уязвимости заключается в следующем:

Запуск вызова: Функция refund отправляет эфир, хранящийся в bidders[msg.sender], на адрес отправителя (msg.sender).  
 В процессе отправки средств код в call может быть выполнен во внешнем контракте (если msg.sender является контрактом). Внешний контракт может в этот момент вызвать refund() рекурсивно или другие функции оригинального контракта ReentrancyAuction.

Это позволяет злоумышленнику извлекать средства или изменять состояние контракта несколько раз в одной транзакции, что может привести к непредвиденным последствиям, включая потерю средств.

**2. Tx.origin vulnerability**— это уязвимость, при которой вредоносный контракт может обманным путем заставить владельца контракта вызвать функцию, которую должен иметь возможность вызывать только владелец.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 10. Tx.origin уязвимость

Дело в том, что tx.origin возвращает исходного отправителя транзакции, которым в данном случае является авторизованная учетная запись и этим могут воспользоваться злоумышленники.

Представим, что есть смарт-контракт, который использует tx.origin для проверки того, что вызывающий является авторизованным пользователем При этом tx.origin будет соответствовать адресу владельца, а не адресу контракта злоумышленника, и require проверка пройдет успешно, что позволит выполнить злонамеренные действия

**3. DoS vulnerability-** это еще одна распространенная уязвимость, о которой должен знать разработчик Solidity

Распространенные типы DoS-атак в Solidity

Исчерпание лимита газа. Эта атака включает отправку транзакций с намеренно низкими лимитами газа. Поскольку транзакции с недостаточным количеством газа не будут выполнены, злоумышленник может наводнить сеть такими транзакциями, что затруднит обработку законных транзакций с надлежащими лимитами газа.

Повторный вход с чрезмерными вызовами. Хотя повторные атаки обычно направлены на кражу средств, их также можно использовать в целях DoS. Злоумышленник может воспользоваться уязвимостью повторного входа, чтобы многократно запускать функцию, потребляя газ и замедляя выполнение контракта.

Атаки, связанные с перенасыщением хранилища. Многократная запись больших объемов данных в смарт-контракт может постепенно истощать доступное хранилище в блокчейне. Это может затруднить или даже сделать невозможным развертывание новых контрактов или обновление существующих.

DoS с самоуничтожением. Злоумышленник может развернуть смарт-контракт, специально предназначенный для многократного самоуничтожения. Это создает большое количество транзакций в сети, что может привести к перегрузке.

Транзакционный спам. Наводняя сеть нерелевантными или бессмысленными транзакциями, злоумышленники стремятся перегрузить вычислительную мощность сети, препятствуя обработке законных транзакций. В качестве примера, рассмотрим одну из разновидностей DoS уязвимостей.

**Gas limit reached**. Это уязвимость, которая злоумышленнику перегрузить контракт и сделать его недоступным для других пользователей.

Каждый смарт-контракт на платформе Ethereum имеет ограничение по расходу газа, которое представляет собой максимальный объем вычислений, который может быть выполнен до того, как выполнение контракта будет остановлено. Если контракт потребляет больше газа, чем указано в лимите, он генерирует исключение и прекращает выполнение. Это может быть использовано злоумышленником путем отправки транзакций в контракт с высоким потреблением газа, в результате чего контракт генерирует исключение и становится недоступным для других пользователей.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 11. **Gas limit reached** уязвимость

На данном примере каждый вызов bidder.call{value: bidders[bidder]}("") не ограничен определенной суммой газа, что означает, что вызываемый контракт может потреблять столько газа, сколько считает нужным.  
- Если во время выполнения вызова происходит ошибка или вызов требует больше газа, чем доступно, операция возврата средств прерывается.

**4. Short Address vulnerability**— это тип уязвимости, которая может возникнуть в смарт-контрактах, написанных на Solidity, языке программирования, используемом для написания контрактов на платформе Ethereum. Эта уязвимость возникает, когда контракт использует короткий адрес или адрес длиной менее 20 байт в качестве входного или выходного значения.

В Ethereum каждая учетная запись имеет уникальный адрес, который представляет собой 20-байтовое значение, которое используется для идентификации учетной записи в блокчейне. Когда контракт получает адрес в качестве входного значения, важно, чтобы адрес был действительным, 20-байтовым значением. Однако если в контракте используется короткий адрес или адрес длиной менее 20 байт, он может быть уязвим для атаки по короткому адресу.

Например, рассмотрим следующий код Solidity:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 12.Short Address vulnerability

Эта функция контракта предназначена для передачи указанного значения из контракта на указанный адрес. Чтобы убедиться, что предоставленный адрес является допустимым 20-байтовым значением, контракт проверяет длину адреса с помощью свойства length. Однако это length свойство не является надежным способом проверки длины адреса, поскольку оно возвращает количество байтов в адресе, а не количество битов. Это означает, что контракт по-прежнему уязвим для атаки по короткому адресу, если злоумышленник предоставляет короткий адрес длиной менее 20 байт, но длиной 20 байт или более.

1.6 Меры профилактики некоторых уязвимостей смарт-контрактов

**Reetrancy**

Атаку повторного входа в контракте DAO можно было избежать несколькими способами. Использование функций send() или Transfer() вместо call.value() не позволит выполнять рекурсивные вызовы вывода средств из-за низкой стипендии на газ. Ограничение количества газа, передаваемого в call.value() вручную, приведет к тому же результату [15].

Тем не менее, существует гораздо более простая практика, которая делает невозможным любую повторную атаку. Обратите внимание, что контракт обновляет баланс пользователя после передачи эфира. Если это было сделано до перевода, любой рекурсивный вызов вывода будет пытаться перевести баланс в 0 эфиров. Этот принцип применяется в целом: если после передачи эфира или вызова внешней функции внутри метода не происходит обновлений внутреннего состояния, метод защищен от уязвимости повторного входа.

**Tx Origin**

Лучший способ предотвратить атаки Tx Origin — не использовать tx.origin для целей аутентификации. Вместо этого желательно использовать msg.sender

**DoS vulnerability solidity**

Есть несколько других рекомендаций, которым разработчики могут следовать, чтобы снизить риск DoS-атак в Solidity:

Используйте функции «view» и «pure»:

«Просмотр» и «чистые» функции в Solidity — это функции, которые не изменяют состояние контракта и не имеют каких-либо внешних побочных эффектов (таких как вызов других контрактов или отправка эфира). Эти функции могут быть помечены ключевым словом «view» или «pure», чтобы указать, что они доступны только для чтения и их не нужно включать в блокчейн. Это может помочь снизить потребление газа этими функциями и сделать их менее уязвимыми для DoS-атак.

Избегайте использования циклов:

Циклы в Solidity могут потреблять большое количество газа, что делает их уязвимыми для DoS-атак. Чтобы избежать этого, разработчики могут использовать альтернативные методы для достижения того же результата, например, использование рекурсивных функций или ключевого слова Solidity «assembly».

Используйте функцию «assert»:

Функция «assert» в Solidity используется для проверки условий, которые всегда должны быть истинными. Если условие, проверенное с помощью «assert», не истинно, контракт выдаст исключение и прекратит выполнение. Это можно использовать для предотвращения DoS-атак, гарантируя, что функции контракта выполняются только при наличии действительных входных данных.

Используйте функцию «revert»:

Функция «возврата» в Solidity используется для остановки выполнения функции контракта и возврата любых изменений, внесенных в состояние контракта. Это можно использовать для предотвращения DoS-атак, гарантируя, что функции контракта выполняются только при наличии действительных входных данных, и позволяя контракту вернуться в известное состояние в случае возникновения ошибки.

Используйте функцию «onERC20Received»:

Если ваш контракт получает токены ERC20, вы можете использовать функцию «onERC20Received» для более экономичной обработки входящих передач токенов. Эта функция позволяет обрабатывать несколько передач токенов за одну транзакцию, снижая риск DoS-атак.

Следуя этим рекомендациям, разработчики могут помочь защитить свои контракты Solidity от DoS-атак и гарантировать, что они останутся доступными и функциональными для предполагаемых пользователей.

**Short Address vulnerability**

Чтобы снизить риск атак по коротким адресам в Solidity, разработчики могут рассмотреть несколько потенциальных решений. Один из вариантов — использовать bytes20тип данных для адресов, который гарантирует, что адрес представляет собой 20-байтовое значение фиксированной длины.

Другой вариант — использовать isContract функцию, чтобы проверить, является ли предоставленный адрес адресом контракта. Это может помочь гарантировать, что адрес является действительным 20-байтовым значением, поскольку адреса контракта всегда имеют размер 20 байт.

1.7 Вывод

В данной главе были рассмотрены основные понятия, связанные с блокчейном Ethereum, что является фундаментальным для дальнейшего понимания механизмов взаимодействия в системе. Эти знание необходимы для анализа и диагностики уязвимостей. Помимо этого, были разобраны отличительные особенности Ethereum, в частности смарт-контракты, а также рассмотрены самые распространенные уязвимости, которые в них встречаются, чтобы в дальнейшем было понимание того, что именно будет пытаться найти написанный анализатор. В последующих разделах будет подробно рассмотрена архитектура анализатора уязвимостей и процесс обучения модель.

Блокчейн Ethereum предоставляет уникальные возможности за счет использования смарт-контрактов, однако эти же особенности делают его подверженным определенным уязвимостям. Важно фокусироваться на разработке безопасного кода и поддержании актуальных знаний о новейших уязвимостях и методах их устранения для обеспечения надежности и эффективности этих технологий.

2. Оценка эффективности использования существующих средств анализа уязвимостей смарт-контрактов

Эффективность существующих инструментов анализа уязвимостей определяется их способностью предсказывать, обнаруживать и предотвращать потенциальные угрозы и уязвимости в коде смарт-контрактов. В ходе этой главы будут проанализированы ключевые средства и методы, которые в настоящее время применяются в индустрии для проверки и защиты смарт-контрактов.

2.1 Существующие средства анализа уязвимостей смарт-контрактов

Средства анализа уязвимостей смарт-контрактов — это специализированные программные инструменты, предназначенные для обнаружения ошибок, потенциальных уязвимостей и других проблем безопасности в коде смарт-контрактов.[6] Эти инструменты играют ключевую роль в процессе разработки и аудита смарт-контрактов, написанных преимущественно на языке программирования Solidity для блокчейн платформы Ethereum, хотя подобные средства существуют и для других блокчейн-платформ и языков. Ниже приведены некоторые популярные инструменты для анализа и проверки уязвимостей в смарт-контрактах:

**Mythril** является частью основных инструментов Consensys Mythx, одной из крупнейших служб безопасности смарт-контрактов для Ethereum, основная цель которой — обеспечить, чтобы команды разработчиков избегали дорогостоящих ошибок и делали Ethereum более безопасным и заслуживающим доверия… или, по крайней мере, так написано на их странице.

Процесс сканирования может занять от нескольких секунд до нескольких минут и в конечно счете выдать нам следующий результат:

1. Идентификатор проблемы.
2. Какая функция вызывает проблему.
3. Количество используемого газа.
4. Подробное объяснение уязвимости. В данном случае происходит разрешение отправлять внешний адрес контракту в качестве параметра, и этот адрес может быть другим контрактом, который выполняет атаку.
5. Точная линия, на которой возникает проблема.
6. Вся последовательность выполнения.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 13. Результат сканера **Mythril**

**Slither** — один из самых популярных инструментов в индустрии аудита безопасности смарт-контрактов. Этот инструмент был разработан Trailofbits и был одним из самых первых инструментов на рынке. Slither разработан на Python 3 и предлагает удобный API, который помогает специалистам по безопасности быстро писать свои собственные анализаторы.

Slither был чрезвычайно оптимизирован для обнаружения уязвимостей с низким уровнем ложных срабатываний и средним временем выполнения менее 1 секунды на контракт. Однако это среднее время зависит главным образом от сложности проверяемого смарт-контракта. Для проверки более сложных смарт-контрактов может потребоваться больше времени.

Slither может анализировать контракты, созданные с использованием версии Solidity, начиная с 0.4, что позволяет охватывать широкий спектр существующих смарт-контрактов. Кроме того, Slither можно легко интегрировать в среду CI/CD, что повышает уровень автоматизации и упрощает использование для разработчиков.

Slither способен обнаруживать следующие уязвимости:

1. Реентерабельные уязвимости
2. Dos уязвимости
3. Опасное использование tx.origin

Изображение выглядит как текст, снимок экрана, документ, дизайн

Автоматически созданное описание

Рисунок 14. Результат сканераSlither

**Oyente,** созданный Лой Луу и его командой в Национальном университете Сингапура, представляет собой инструмент символического анализа для выявления уязвимостей безопасности в контрактах Ethereum и байт-коде EVM.

Разработчики могут использовать Oyente для оценки локальных и удаленных контрактов и проверки утверждений для этих контрактов. Полная установка зависит от таких пакетов, как Web3.js, Z3 Therome Prover. Один из старейших анализаторов контрактов в таких проектах, как Quantstamp и Augur, использовал Oyenete с момента их создания.

**MAIAN** – инструмент с открытым кодом для анализа трех типов уязвимостей:

• Reetrancy

• Suicidal contracts (”cуицидальные” контракты, т.е. могут деструктор контракта может быть вызван любым пользователем).

Также может анализировать как на основе исходного кода, так и адреса контракта.

**SAILFISH** - масштабируемая система для автоматического поиска ошибок, связанных с несоответствием состояний в смарт-контрактах.Чтобы сделать анализ более понятным, в нем применяется гибридный подход, который включает в себя (i) упрощенную фазу исследования, которая значительно сокращает количество инструкций для анализа, и (ii) точную фазу уточнения, основанную на символьной оценке, основанной на новом анализе обобщения значений, который создает дополнительные ограничения для чрезмерного анализа. В общей сложности SAILFISH обнаружил 47 ранее неизвестных уязвимостей.смарт-контракты из 89 853 смарт-контрактов от ETHERSCAN

**Smartian** — это фаззер для смарт-контрактов Ethereum. Smartian использует статический и динамический анализ потоков данных для повышения эффективности фаззинга. Технические подробности Smartian можно найти в статье «Smartian: улучшение фаззинга смарт-контрактов с помощью статического и динамического анализа потоков данных», опубликованной в ASE 2021.

Securify 2.0 — это сканер безопасности для смарт-контрактов Ethereum, поддерживаемый Ethereum Foundation и ChainSecurity. Основное исследование Securify проводилось в Лаборатории безопасных, надежных и интеллектуальных систем ETH Zurich.

Он является преемником популярного сканера безопасности Securify

2.2 Оценка эффективности использования анализаторов;

Проверка действующих сканеров уязвимостей:

В качестве данных возьмем reentrancy\_information.csv с github. Тут предоставлена выборка из смарт контрактов содержащих уязвимости и не содержащих их а так же сканеры, которыми проверяли данные smart-контракты. В случае нахождения уязвимости в таблице появлялось значение “1” и в случае ее не нахождения “0” соответственно. Если вдруг анализатор не выдавал результата сканирования, в таком случае в колонке появлялось значение NaN [7].В качестве ответов выступает колонка **true\_positive.**

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 15 «Данные по сканерам»

Для начала посмотрим, сколько Nan значений присутствует в таблице.

Изображение выглядит как текст, снимок экрана, дисплей, Шрифт

Автоматически созданное описание

Рисунок 16 «Количество NaN значений»

Увидим огромное количество Nan значений. Посчитаем процентное соотношение Nan ячеек относительно заполненных.

Посчитаем количество Nan значений для сканера уязвимостей oyente. Увидим,что процент Nan значений выше 59 процентов.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 17. Количество NaN значений для oyente

Выполним аналогичный подсчет для сканера mythril. Заметим, что процент Nan значений выше 41 процентов. Это лучше, чем у предыдущего сканера, однако все равно довольно высокий процент (почти половина всей выборки)

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

Рисунок 18. Количество NaN значений для mythril

Теперь выполним подсчет для сканера для securify1. Процент Nan значений составил примерно 55 процентов.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 19. Количество NaN значений для securify1

Для сканера securify2 процент Nan значений составил примерно 85 процентов. Это наихудший результат среди всех сканеров

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 20. Количество NaN значений для securify2

Лучше всех себя проявил сканер smartian. Процент Nan значений составил 28 процентов, что все равно является довольно большим значением(почти 1/3 всей выборки)

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 21. Количество NaN значений для smartian

Последним сканером в таблице является sailfish.Его результат Процент Nan значений составил примерно 53 процента(чуть больше чем половина всей выборки).

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 22. Количество NaN значений для sailfish

Таким образом, можно сделать вывод, что большое количество NaN ячеек в результатах сканирования сканеров уязвимостей связано с использованием статистических методов обнаружения уязвимостей, которые основаны на поиске паттернов или шаблонов уязвимостей. В контексте смарт-контрактов на языке Solidity это может означать, что многие уязвимости не были обнаружены, поскольку они не соответствуют известным шаблонам или паттернам, используемым сканерами.

Далее посчитаем процент правильных прогнозов сканеров. Для этого сравним построчно результаты каждого сканера с ответом true\_positive. Затем просуммируем совпадения и разделим их на количество не Nan строк.

Тут результаты намного лучше. Например, процент точных прогнозов у сканера oyente составил >99,1%. Причем это еще не самый лучший результат.

Изображение выглядит как текст, линия, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 23. Количество точных прогнозов для oyente

Для сканера уязвимостей mythril точность составила >80,9%,что тоже является не плохим результатом.

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

Рисунок 24. Количество точных прогнозов для mythril

Точность сканера securify1 составила >74 %, что, в целом, является утешительным результатом (3/4 ответов были правильными)

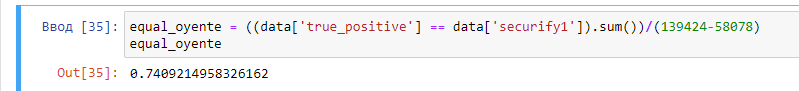


Рисунок 25. Количество точных прогнозов для securify1

Наихудшую точность показал сканер securify2 < 29%. Почти 3/4 всех прогнозов оказались неверными.

Изображение выглядит как текст, линия, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 26. Количество точных прогнозов для securify2

Наибольшую точность показал анализатор уязвимостей Smartian.Точность его прогнозов составил более >99,9% что является отличным результатом.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 27. Количество точных прогнозов для smartian

Последним анализатором в таблице был sailfish. Точность его прогнозов составилa более >96,5% что является так же отличным результатом.

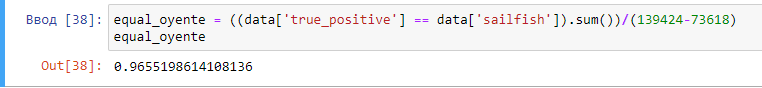


Рисунок 28. Количество точных прогнозов для sailfish

2.3 Вывод

В данной главе были рассмотрены основные анализаторы уязвимостей смарт-контрактов на языке solidity для блокчейна Ethereum. Анализаторы занимались поиском уязвимости reetrancy, которая была рассмотрена подробно в первой главе.

Как видно из примеров выше, поиск уязвимостей по шаблонам, которые имели анализаторы, дает довольно высокую точность в обнаружение уязвимостей. Однако, данный метод показывает весьма неутешительные результаты в анализе контрактов в целом. В зависимости от сканера, количество Nan значений варьировалось от 28% до 85% процентов.

В тоже время (о чем уже упоминалось ранее) многие сканеры продемонстрировали довольно хорошую точность прогнозов (свыше 90 процентов) что является отличным результатом, но все же огромное количество Nan значений не позволяет назвать данные анализаторы успешными.

Так же следует отметить, что все сканеры находятся на стадии beta-тестирования. На данный момент, полностью готовых продуктов для анализа смарт-контрактов в мире нет.

3. Разработка и внедрение анализатора уязвимостей смарт-контрактов.

В данной главе освещается процесс разработки и внедрения анализатора уязвимостей смарт-контрактов, который базируется на использовании современных технологий искусственного интеллекта, а именно нейронных сетей. Учитывая повышенные требования к безопасности и надежности смарт-контрактов, которые непрерывно взаимодействуют с значительными финансовыми потоками и конфиденциальной информацией, поиск инновационных подходов к их защите становится особенно актуален [8].

Нейронные сети уже доказали свою способность к эффективному анализу и обработке больших объемов данных в самых разнообразных областях, от распознавания изображений до обучения машин. В контексте смарт-контрактов, применение нейронных сетей открывает новые перспективы для распознавания сложных шаблонов уязвимостей, которые могут быть неочевидны для традиционных методов анализа.

3.1 Введение в нейронные сети и глубокое обучение

Нейронные сети — это подкласс моделей в машинном обучении, которые были разработаны по аналогии с человеческим мозгом. В основе их работы лежит концепция "нейронов" — основных вычислительных единиц, которые обрабатывают входящую информацию и передают результаты далее по сети.

Структура нейронной сети включает в себя:

1. Входной слой – принимает входные данные.

2. Скрытые слои – обрабатывают данные посредством весов и функций активации. Могут быть один или несколько слоёв.

3. Выходной слой – выдает результат работы сети.

Каждый "нейрон" в сети связан с другими "нейронами" посредством "весов", которые управляют степенью влияния одного нейрона на другой. "Веса" подлежат оптимизации в процессе обучения для улучшения работы модели.

Архитектура нейронной сети определяет конкретные используемые слои и их порядок Конкретная нейронная сеть задается с помощью архитектуры и размеров слоев. Для преобразования входных данных в выходные в каждом нейроне так же присутствует функция активации. Это функция в нейроне, которая преобразует входные данные в выходные для их последующей передачи на следующий слой. Входными данными является линейная сумма всех весов. Рассмотрим необходимость ее использования на примере:

Изображение выглядит как линия, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 29. Два полносвязных слоя (без функции активации)

Распишем уравнение для получения y, подставив x вместо z:

Тогда

Аналогично находим

Упростив данное выражение можно заметить, что все наши полносвязные слои преобразовались в одну линейную модель. Получается, что смысла в добавлении промежуточных слоев нет. Такая ситуация справедлива для сколь угодно большого количества слоев и называется вырождением слоя. Eе можно добавлением между полносвязными слоями нелинейной функции — функцию активации. Наиболее известные функции активации — сигмоида и ReLU:

Изображение выглядит как снимок экрана, линия, диаграмма, дизайн

Автоматически созданное описание

Рисунок 30. Простейшая архитектура нейронной сети

Глубинное обучение (Deep Learning) — это вид машинного обучения, включающий алгоритмы, обрабатывающие входные данные путём последовательного применения к ним большого количества линейных и нелинейных преобразований (слоев).

Результат одного слоя подается на вход последующему;

Главное условие к слоям — они должны быть дифференцируемы;

Обучение осуществляется методом градиентного спуска для параметров каждого слоя;

Градиенты вычисляются методом «обратного распространения ошибки»

Основная идея градиентного спуска заключается в нахождении локального минимума или максимума функции с помощью движения вдоль градиента. Градиент — это вектор, направленный в сторону наискорейшего роста функции, а, поскольку нужно минимизировать функцию потерь, то необходимо двигаться по нему но с минусом. Процесс оптимизации итеративный:

1. Считаем градиент в случайной точке , в окрестности которой функция должна быть определена и дифференцируема. Обычно это точка 0.

2. Считаем градиент в новой точке и опять сдвигаемся по нему. Таким образом чередуется обновление параметров и вычисление градиента

1. Ведем оптимизацию до какого-то критерия остановы.

Критерий остановы выбирается самостоятельно и обозначается буквой ξ. Варианты нахождения критерия останова:

1. Определенное минимальное значение производной. Если достигаем выбранного значения производной, то останавливаем спуск.

2. Маленькая длина шага. Если шаг градиента становится маленьким, то останавливаем спуск.

3. В глубинном обучении оптимизация ведётся до тех пор, пока ошибка на отложенной выборке продолжает падать.

Поскольку функция потерь состоит из суммы некоторых функций потерь на отдельных объектах обучающей выборки, то и градиент этой функции потерь будет состоять из суммы градиентов по отдельным объектам. Это называется полный градиентный спуск, он вычисляет среднее значение для всего набора данных. Т.е. для каждого шага оптимизации нужно будет посчитать все градиенты и поделить на количество объектов — это долго и дорого.

Взамен этого можно оценить градиент по одному объекту или по подмножеству [10].

Оптимизации нейронных сетей ведётся следующим образом: на каждом шаге выбираются некоторые случайные объекты из обучающей выборки. Для них считаются градиенты, затем, суммируются и усредняются. Выбранные объекты называются batch, а размер — batch size.

3.2 Задача представления признаков смарт-контрактов

Для того чтобы искусственные нейронные сети могли эффективно анализировать текстовые данные и извлекать из них нужные характеристики, они должны уметь понимать значения слов.

Эмбединг (от английского "embedding") в контексте машинного обучения (ML) — это представление каких-либо данных (обычно высокоразмерных, как тексты или изображения) в виде векторов меньшей размерности, которые сохраняют ключевые характеристики исходных данных. Векторное представление слов используются для трансформации сложных данных в форму, более удобную для анализа и обработки алгоритмами машинного обучения.

В NLP, например, Векторное представление слов используются для того, чтобы компьютер мог понять, что слова «кошка» и «котенок» связаны между собой ближе, чем, скажем, «кошка» и «окошко». Это достигается путем присвоения словам векторов, которые отражают их значение и контекстное использование в языке.

Сходство между векторами измеряет степень "похожести" между двумя векторами. В векторных представлениях слов это используется для определения степени семантической или контекстной близости между элементами.

Одна из наиболее популярных метрик — это **косинусное сходство**. Косинусное сходство измеряет косинус угла между двумя векторами. Если угол между векторами мал, косинус приближается к 1, что указывает на высокое сходство. Математически оно выражается как:

Изображение выглядит как текст, Шрифт, рукописный текст, белый

Автоматически созданное описание

Рисунок 31. Косинус угла между двумя векторами

После того текст будет представлен в виде векторного представления слов, получится некий словарь.

Словарь векторов (embedding dictionary или embedding matrix) — это структура данных, в которой хранятся векторные представления элементов некоторого множества, например, слов в случае обработки естественного языка, или ID пользователей в системах рекомендаций. Каждый такой вектор в словаре соответствует одному элементу из этого множества и представляет собой вектор чисел, который кодирует информацию о данном элементе.

Основные особенности словаря векторов:

- Идексация: Каждый элемент множества (например, каждое слово в словаре языка) ассоциируется с индексом, по которому можно быстро находить его векторное представление матрице.

- Размерность: Все вектора в словаре обычно имеют одинаковую размерность, определяемую параметрами модели

- Использование в ML моделях: вектора могут использоваться как начальный слой нейронных сетей, преобразуя входные данные в векторы, которые затем обрабатываются последующими слоями.

Для представления текста в виде векторов чаще всего применяется технология семантического анализа, среди которых одним из наиболее эффективных методов является Word2Vec. Эта техника использует нейронные сети для трансформации слов в векторы. Векторы слов, семантически или контекстуально близких, имеют схожие значения, что облегчает процесс обучения: чем чаще слово встречается в похожих контекстах, тем более близкими становятся его векторы к векторам семантически схожих слов. Создание таких моделей требует значительных вычислительных ресурсов, однако существуют уже готовые модели, доступные для использования [11].

Основные идеи

1. Векторные представления: Word2Vec преобразует каждое слово в вектор фиксированной длины в высокомерном пространстве. Эти векторы представляют семантические значимости слов. Например, вектор для «короля» будет близок к вектору для «королевы», отражая их семантическую близость в человеческом языке.  
Изображение выглядит как линия, диаграмма, График, скат

Автоматически созданное описание

Рисунок 32. Идея Word2Vec

Классический пример, демонстрирующий способность представлений word2vec кодировать аналогии, — это классический пример «король + женщина — мужчина ≈ король», показанный ниже.

2. Модели: Word2Vec реализует две архитектуры:

CBOW (Continuous Bag of Words): Эта модель предсказывает текущее слово на основе контекста (например, соседних слов). Цель состоит в том, чтобы оптимизировать веса нейронной сети для того, чтобы входной контекст приводил к максимально точному предсказанию целевого слова.

Изображение выглядит как текст, диаграмма, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 33. Пример работы CBOW

Принцип работы

В отличие от модели Skip-Gram, где алгоритм предсказывает контекстные слова, основываясь на целевом слове, CBOW предсказывает целевое слово на основе контекста. Это можно представить как задачу предсказания слова по нескольким его окружающим словам. Процесс тренировки модели CBOW заключается в минимизации ошибки между реальным словом и его предсказанием, производимым на основе контекста.

Для этого сначала выбирается размер "окна" – количество слов до и после целевого слова, которые будут использованы для предсказания. Например, если окно установлено в 2, то для предсказания слова используются два слова до и два слова после него.

Процесс векторизации:

Для каждого слова в корпусе CBOW строит векторы, так называемые встраивания :

1. При обучении модели на корпусе текстов создаются векторы для всех слов контекста.

2. Эти векторы суммируются или усредняются, чтобы сформировать единый вектор контекста.

3. На основе полученного вектора контекста модель предсказывает целевое слово, обычно с использованием методов мультиклассовой классификации, таких как softmax.

Преимущества и недостатки

Преимущества:

- Быстрая скорость, так как за один раз обрабатывается целый контекст.

- Эффективная обработка нескольких примеров одновременно.

Недостатки:

- Тенденция к менее точным представлениям редких слов по сравнению.

- Меньшая эффективность в задачах, требующих углубленного понимания контекстуальных взаимосвязей слов.

CBOW хорошо подходит для больших данных и предложений, где контекстные слова дают достаточную информацию для предсказания целевого слова. Это делает его особенно полезным для начального этапа построения моделей языка или трансляции при прочих равных условиях.

Skip-gram: Отличается от CBOW тем, что вместо предсказания целевого слова по контексту, модель предсказывает контекст по целевому слову. Это особенно полезно для обучения на малых данных, так как обеспечивает большее количество обучающих примеров для каждого наблюдения.

Суть следующая:

Есть какой-то текст,по нему можно пройти окном. Для каждого центрального слова взять контекст вокруг него и нагенерировать обучающие пары. Т.е цель заключается в том, чтобы вероятность встретить одно слово около другого была больше. В итоге у нас получается большое количество обучающих пар.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 34. Пример работы Skip-gram

Основной принцип работы Skip-Gram - предсказание слов на основе их расположения относительно данного слова в тексте. Алгоритм анализирует последовательности и выявляет связи, исходя из контекста. Чтобы предсказать следующее слово, алгоритм ищет его в базе данных на основе ранее встреченных слов.

Кроме того, Skip-Gram использует такие параметры, как окно просмотра, список положительных и отрицательных слов, а также матрицу весов для повышения точности предсказаний.

В процессе построения связей между словами Skip-Gram задействует параметры, такие как размер окна просмотра, которое определяет число рассматриваемых слов для анализа связей, а также учитывает положительные и отрицательные слова для более точного предсказания слов-ассоциаций. Матрица весов применяется для повышения точности этих предсказаний. Все эти элементы способствуют более глубокому анализу и пониманию контекстуальных взаимосвязей слов.

В конечном итоге, технология Skip-Gram обеспечивает глубокое понимание данных, что позволяет алгоритму эффективно распознавать связи между словами, что находит применение в различных задачах, включая анализ настроений, обработку изображений и многие другие.

3. Оценка вероятности: Одной из ключевых задач в процессе обучения Word2Vec является оценка вероятности встречаемости слова в заданном контексте, что обычно достигается при помощи методов, таких как иерархическая softmax оптимизация или отрицательная выборка [12].

Преимущества:

1) Понимание контекста: векторы улавливают многоуровневые отношения между словами, включая синонимы, антонимы и ассоциации.

Рассмотрим фрагмент кода, в котором происходит реализация представления смарт контракта:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 35. Кодирование смарт-контракта

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 36. Смарт-контракты до токенизации

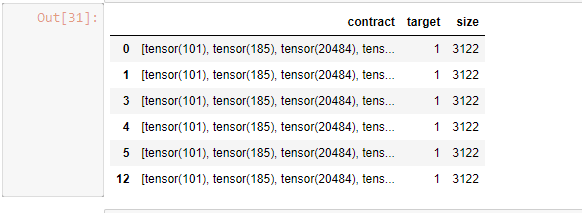
Функция `encode\_plus` от библиотеки `transformers` (о трансформерах речь пойдет чуть позже), предоставляет механизм для токенизации текста, в данном случае строки или текстовой переменной `biggest` а `3122` — это длина последовательности, указанная в `max\_length`.

Рисунок 37. Смарт-контракты после токенизации

3.3 Принцип работы нейросети архитектуры трансформер

Для создания анализатора уязвимостей в дипломной работе используется одна из наиболее современных и успешных архитектур нейронных сетей- архитектура трансформер [13].

Трансформеры — это тип архитектуры нейронных сетей, который был впервые представлен в 2017 году.

Архитектура трансформеров оказала значительное влияние на последующее развитие методов машинного обучения, особенно в области обработки естественного языка (NLP). В отличие от предыдущих моделей, таких как рекуррентные нейронные сети (RNNs), трансформеры полагаются исключительно на механизмы внимания и полностью отказываются от рекуррентности, что делает их более эффективными с точки зрения параллельной обработки и обучения на больших объемах данных.[14]

Трансформеры широко используются для решения ряда задач в области NLP, таких как машинный перевод, автоматическое резюмирование текстов, генерация текстов, а также в задачах, выходящих за рамки NLP, включая обработку изображений и видео. С появлением архитектур, таких как BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pre-trained Transformer), и других, качество и возможности моделей значительно расширились.

Трансформеры являются особенно важным инновационным достижением, предоставляя мощную и гибкую архитектуру для решения множества задач обучения с учителем и без учителя.

Рассмотрим подробнее принцип работы данной архитектуры:

Собственно, данная архитектура пришла из задачи машинного переводa.Компания Google пыталась решить данную задачу.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 38. Задача машинного перевода

На примере выше видно: подаем текст на французском языке, на выход получаем текст на английском.

Схематически архитектура трансформера выглядит следующим образом:

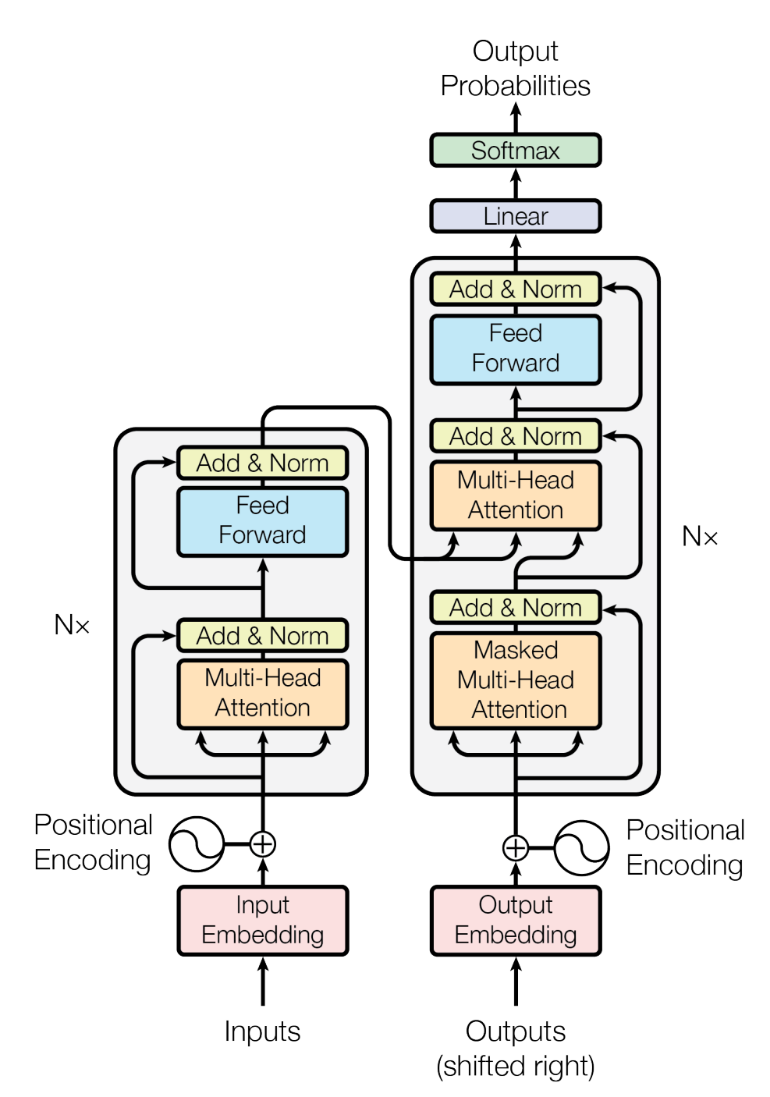


Рисунок 39. Схематическое изображение архитектуры трансформера

На схеме видно два больших блока. Слева изображен кодирующий компонент, справа-декодирующий компонент.

Кодирующий компонент – это стек кодеров; в иллюстрации ниже изображены 6 кодеров, расположенных друг над другом (в числе 6 нет ничего магического, можно экспериментировать и с любым другим числом).

Декодирующий компонент – это стек декодеров, представленных в том же количестве.

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание

Рисунок 40. Схема кодирующего и декодирующего компонентов

Детальнее разберем из чего состоит кодер.

Изображение выглядит как текст, снимок экрана, линия, Прямоугольник

Автоматически созданное описание

Рисунок 41. Схема кодера

Каждый блок кодера принимает на вход N векторное представление слова размером D и на выход так же возвращает N векторных представлений слов размером D.Как видно из схема выше, кодер состоит из двух слоев:

1. Self-Attention
2. Feed Forward Neural Network

Начнем разбор с блока Feed Forward Neural Network потому что там все довольно просто. Feed Forward слой- это довольно простая конструкция. Она состоит из двух линейных слоев между которыми есть функция нелинейности (например ReLU).

Предположим у нас есть векторное представление слова D размером m, увеличиваем его в размере до ff (отсюда и название Feed Forward) затем снова возвращаем к размеру m

Таким образом, слой FFNN в трансформере работает как своего рода дополнительный полносвязный перцептрон, который обрабатывает и модифицирует выходные данные механизма внимания, помогая улучшить и углубить репрезентацию данных перед передачей к следующему слою в сети.

Блок Self-Attention позволяет модели выделить важные части входных данных путем изучения взаимосвязей между различными частями этих данных. Это особенно полезно в задачах обработки естественного языка, где контекст слова может сильно зависеть от других слов в предложении или документе. Этот механизм позволяет модели динамически сосредотачиваться на разных частях входных данных для вычисления представления каждого элемента, учитывая его контекст в последовательности. Принцип работы блока внутреннего внимания:

На вход слою внутреннего внимания приходит последовательность векторов, каждое слово из последовательности закодировано в вектор. Механизм внутреннего внимания работает на основе трех основных компонентов: Query (Запросы), Key (Ключи) и Value (Значения). Эти компоненты создаются путем умножения входной последовательности на соответствующие обучаемые матрицы весов Q V K:

Стоит отметить, что размерность новосозданных векторов меньше, чем у исходных векторов. Они имеют размерность 64, тогда как размерность векторных представлений слов и векторов входа/выхода кодера составляет 512. Это уменьшение размерности не является обязательным условием, но в данном случае, выбор такой структуры модели способствует стабильности вычислений в слое множественного внимания.

Изображение выглядит как текст, снимок экрана, диаграмма, дизайн

Автоматически созданное описание

Рисунок 42. Иллюстрация self-attention

Рассмотрим пример выше:

На вход поступает два вектора и ,размерности 1х4, умножаем на матрицу размерности 4х3(Например WQ ) и в итоге получаем векторы 1х3 и

Умножив вектор на три матрицы Q, K, V получим соответственно . Также умножив вектор на три матрицы Q, K, V получим соответственно .

Затем для каждого слова считаем сумму поэлементного умножения его вектора q на векторы k всех остальных слов, включая его самого. Считаем так называемый точечный продукт. Чем выше значения скалярного произведения, тем больше «Похожесть» двух векторов и тем больше значение точечный продукт. Получаем для этого слова набор чисел, делим каждое из этих чисел на корень из длины векторов q, k и v (они все одинакового размера). Применяем поверх этого набора чисел логистическую функцию softmax. Получили веса, складываем с этими весами векторы v, получаем итоговый вектор z. Этот итоговый вектор z считается для каждого слова из последовательности. *э-*

Весь вышеописанный алгоритм можно представить формулой:

Описанная операция происходит параллельно несколько раз — последовательность подается в несколько голов (heads). В каждой голове получаем векторы z для всей последовательности. Получается, что у нас для каждого слова из последовательности есть набор из векторов z, по числу голов.Каждая тройка матриц q, k, v будут разными для каждой из голов.В таком случае каждая из голов будет обращать внимание на абсолютно разные связи в предложениях и реагировать на разные вещи по-разному

Теперь поговорим о позиционном кодировании. Это отдельная матрица векторных представлений слов, которая имеет такую же размерность, как и исходные. Позиционное кодирование добавляет к векторным представлениям слов векторы t для определения позиций слов, эти векторы фиксированные для позиций и тоже являются обучаемыми.

В итоге получается следующий алгоритм:

Этапы прохождения последовательности через кодер:

1)Кодируем каждый токен в векторное представление слов, добавляем позиционное кодирование (сигнал про его место в последовательности).

2)Пропускаем через блоки кодеров.

3)Выдаем на выход блока кодера для каждого токена вектор такой же размерности, что и вошел.

4)Отправляем выход последнего кодера на вход всем декодерам.

Теперь рассмотрим принцип работы декодера.

На вход поступает три вектора

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

Рисунок 43. Блок Декодера

Этапы прохождения последовательности через декодер

1)Кодируем каждый токен в векторное представление, добавляем позиционное кодирование (сигнал про его место в последовательности)

2)Пропускаем через блоки декодеров

3)Выдаем на выход блока декодера для каждого токена вектор такой же размерности, что и вошел

Внутри одного блока декодера:

1)Механизм внимания, но ограниченный, можно для каждого слова смотреть только на те, что раньше его

2)Технология отправки градиентов через слои и нормализация

3)Механизм внимания, для вычисления q используется вход декодера, для вычисления k и v используется выход последнего энкодера.

4) Технология отправки градиентов через слои и нормализация

5) Два полносвязных слоя

6) Технология отправки градиентов через слои и нормализация

На выходе блок декодеров выдает вектор с плавающей точкой. Чтобы преобразовать этот вектор в слово, используются линейный слой и последующий слой логистической функции.

Линейный слой функционирует как простая полносвязная нейронная сеть, которая преобразует входящий вектор от стека декодеров в значительно больший вектор.

Допустим, модель оперирует словарем из 10 тысяч уникальных английских слов, изученных из обучающего корпуса. Это означает, что вектор будет содержать 10 000 элементов, где каждый элемент соответствует одному уникальному слову. Таким образом, работает линейный слой для интерпретации выходных данных модели.

Слой логистической функции затем преобразует данный вектор в вероятности, которые являются положительными числами, суммирующимися в единицу. Слово, соответствующее ячейке с наивысшей вероятностью, выбирается как вывод за данный временной интервал.

Стоит также упомянуть, что в данный момент хорошо себя показывают трансформеры типа Encoder-only, Decoder-only, представляющие собой вышеописанную архитектуру, но состоящую только из блоков одного типа. Яркими представителем Decoder-only моделей является семейство GPT от OpenAI. В случае Encoder-only моделей, самым известным представителем пожалуй является BERT от Google.

3.4 Создание анализатора уязвимостей с использованием архитектуры трансформер

Статистический анализатор представляет собой обученную нейронную сеть архитектуры трансформер, предсказывающая на выходе вероятность наличия какой-либо уязвимости в смарт контракте.

В качестве обучающей выборки был взять Dataset на 100000 smart контрактов.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 44. Выборка смарт контрактов

Выборка была поделена на тренировочную, валидационную и тестовую части (в соотношении 60/20/20). Для подачи в нейронную сеть трансформер проводилась предварительное разбиение кода смарт контрактов на более мелкие части (токены). Объём словаря составил порядка 30000. Итоговый смарт контракт представлял собой последовательность токенов. Длина последовательности была ограничена в 4300 токенов в связи с ограниченностью видеопамяти видеокарты.

Наша задача представляла собой бинарную классификацию: есть уязвимость -1, нет уязвимости – 0. Ниже представлена реализация градиентного спуска нашей модели. Это ни что иное, как алгоритм итеративной оптимизации, используемый в машинном обучении для получения более точного результата (то есть поиска минимума кривой или многомерной поверхности).

Градиент показывает скорость убывания или возрастания функции.

Спуск говорит о том, что имеем дело с убыванием.

Алгоритм **итеративный**, процедура проводится несколько раз, чтобы добиться оптимального результата. При правильной реализации алгоритма, на каждом шаге результат получается лучше. Таким образом, итеративный характер градиентного спуска помогает плохо обученной модели оптимально подстроиться под данные.

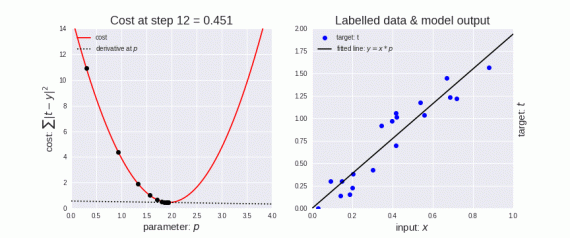


Рисунок 45. Градиентный спуск

У градиентного спуска есть параметр, называемый **скоростью обучения**. На левой верхней картинке видно, что в самом начале шаги больше, то есть скорость обучения выше, а по мере приближения точек к краю кривой скорость обучения становится меньше благодаря уменьшению размера шагов. Кроме того, значение **функции потерь** уменьшается, или просто говорят, что потери уменьшаются. Часто люди называют функцию потерь  или просто «Лосс». Важно, что если  функция потерь уменьшается, то это хорошо.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 46. Часть кода с обучением модели

Чтобы эффективно работать с большими объемами данных, требуется использовать параметры (epoch, batch size, итерации), так как зачастую нет возможности загрузить сразу все данные в обработку.

Для преодоления этой проблемы данные делят на части меньшего размера, загружают их по очереди и обновляют веса нейросети в конце каждого шага, подстраивая их под данные.

Произошла одна эпоха (epoch) — весь датасет прошел через нейронную сеть в прямом и обратном направлении только один раз. Поскольку обновления весов после одного прохождения недостаточно, поэтому реализуем несколько эпох для избежания переобучения модели.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 47. Реализация эпохи в коде

3.5 Вывод

В этой главе представлены ключевые аспекты создания инновационного анализатора уязвимостей для смарт-контрактов, задействующего передовые технологии искусственного интеллекта. Исследование начинается с введения в базовые концепции нейронных сетей и глубокого обучения

Далее идет разбор принципов работы архитектуры нейросети типа трансформер, выбранной за её способность эффективно обрабатывать последовательности данных с высокой точностью и надежностью. Это обосновывает выбор данной архитектуры для задачи анализа кода смарт-контрактов.

Последующие разделы посвящены непосредственной разработке и внедрению анализатора уязвимостей. Теперь необходимо протестировать разработанное программное средство на точность обнаружения уязвимостей в смарт-контрактах и оценить работоспособность модели по различным метрикам оценивания

4. Демонстрация работоспособности разработанного анализатора уязвимостей смарт-контрактов

В этой главе были подробно рассмотрим уязвимые части кода смарт контрактов, которые модель должна была выделить в процессе обучения и рассмотрены возможные пути профилактики данных уязвимостей. Затем оценена работоспособность модели по различным метрикам и подведены итоги.

4.1 Тестирование разработанного анализатора смарт-контрактов на поиск уязвимых смарт-контрактов

В процессе обучения модели выборка была разделена на тренировочную и тестовую. Как видно из кода анализатора ниже, 80 процентов выборки пошло на обучение модели, оставшиеся 20 - на тестирование.

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

Рисунок 48. Деление выборки на тренировочную и тестовую

После того, как модель обучилась, она была протестирована на 20 процентах выборки. Главной задачей было нахождение уязвимых смарт-контрактов. В качестве уязвимостей в тестовой выборке использовались уже ранее рассмотренные в первой главе reetrancy, dos, tx\_origin Short Address уязвимости. Помимо смарт-контрактов, содержащих уязвимости, в тестовой выборке, для более точного тестирования модели, так же находились контракты в которых данных уязвимостей нет. В качестве оценивания результата использовались различные метрики машинного обучение, речь о которых пойдет далее.

4.2 Метрики оценивания качества модели

Оценку эффективности моделей начнём с рассказа об используемых метриках и построения матрицы ошибок, в колонках которой будут расположены истинные классы наших объектов, а в строках - те классы, которые предсказывает модель [16]..

|  |  |  |
| --- | --- | --- |
|  | y = 1 | y = -1 |
| a(x) = 1 | TP | FP |
| a(x) = -1 | FN | TN |

Где a(x) – это прогноз модели, y = истинная метка класса.

Если модель предсказала, что класс положительный и он в действительности оказался положительным: такую ситуацию называют True Positive.

Если модель предсказала класс как положительный, а на самом деле он - отрицательный: такая ситуация False Positive.

Алгоритм предсказывает отрицательный класс, а на самом деле - положительный: False Negative. И наоборот - True Negative.

Первая метрика, что использовалась для оценки качества моделей – Accuracy, доли правильно размеченных объектов.

Соответственно для каждого объекта можно сказать, в какую ячейку из таблицы его можно отнести. Тогда каждую из этих ячеек можно заполнить количеством объектов, которые относятся к данным типам и посчитать, Accuracy.

Из минусов этой метрики пожалуй стоить отметить её неприменимость перед данными с большим дисбалансов классов. Имея к примеру 999 экземпляров положительного класса к 1 отрицательному, константная модель выдающая всегда положительный класс будет иметь accuracy = 0,999.

Перейдем непосредственно к метрикам, которые позволяют нам более справедливо оценивать качество работы наших моделей по сравнению с Accuracy.

Две самые популярные метрики - это так называемые Precision и Recall.

Precision позволяет понять, какая доля объектов среди тех, которые назвали положительным классом, действительно к нему относится.

Recall же показывает, сколько объектов среди тех, которые в общем были положительные, наша модель смогла выявить.

Изображение выглядит как снимок экрана, круг, текст, диаграмма

Автоматически созданное описание

Рисунок 49. Визуализация метрики Recall

Но модель достаточно редко обладает как хорошим значением Precision, так и хорошим значением Recall одновременно. И, как правило, приходится балансировать между двумя этими метриками.

Recall будет высоким, в случае, когда модель покрывает как можно большее количество положительных классов изначально которые были в выборке.

Модель достаточно редко обладает как хорошим значением Precision, так и хорошим значением Recall одновременно. И, как правило, приходится балансировать между двумя этими метриками.

Есть несколько вариантов:

Посчитать арифметическое среднее:

В чем минус такого подхода:

Пусть есть одна модель, у которой recall = 0.6, precision = 0.5, average = 0.55. В среднем неплохое значение, но и не хорошее. А также есть вторая модель, у которой recall = 0.8, precision = 0.3, average = 0.55. Такая модель уже будет менее желательной.

Т.е можно сделать вывод, что среднее арифметическое не чувствует разницу между умеренными и крайними случаями.

Посчитать геометрическое среднее:

В чем недостаток такого подхода:

Пусть есть одна модель: recall = 0.6, precision = 0.5, Average = 0.54. А у второй модели: recall = 0.8, precision=0.3, Average = 0.49. В таком случае действительно получим некоторую разницу в значении среднего.

Посчитать F-меру:

Преимущество F-меры еще в том, что можно немного изменить формулу и добавить еще учет предпочтения между precision и recall.

Модифицированная формула выглядит следующим образом:

Чем больше , тем больше уклон в recall. И наоборот: при 0 < < 1 уклон будет больше в precision.

Последней метрикой в оценке качества моделей является roc\_auc\_score.

ROC AUC (Receiver Operating Characteristic Area Under the Curve) - это метрика качества модели, которая оценивает ее способность различать классы в задаче бинарной классификации. ROC AUC измеряет площадь под кривой ROC (Receiver Operating Characteristic), которая показывает зависимость между чувствительностью (вероятность правильной классификации положительного класса) и специфичностью (вероятность правильной классификации отрицательного класса).

Значения ROC AUC лежат в диапазоне от 0 до 1, где 1 - идеальное качество модели, а 0.5 - случайное угадывание. Чем выше значение ROC AUC, тем лучше модель различает классы.

Изображение выглядит как текст, линия, диаграмма, График

Автоматически созданное описание

Рисунок 50. Пример графика по метрике ROC-AUC

ROC AUC является предпочтительной метрикой в задачах, где классы не сбалансированы и важно как правильно идентифицировать положительные случаи, так и минимизировать количество ложных срабатываний.

Метрика ROC AUC позволяет сравнивать различные модели между собой и выбирать лучшую из них для конкретной задачи.

4.3 Результаты оценивания работоспособности разработанной модели

Ниже представлены результаты обучения модели.

Оранжевая линия – функция ошибки на данных которая модель не видел, синея – те данные, на которых модель обучалась:

Изображение выглядит как линия, График, диаграмма, Параллельный

Автоматически созданное описание

Рисунок 51. График обучение

Также при процессе обучения строилась и кривая функции ошибки, оранжевая линия – функция ошибки на данных которая модель не видел, синея – те данные, на которых модель обучалась:

Изображение выглядит как линия, График, диаграмма, Параллельный

Автоматически созданное описание

Рисунок 52. Кривая функции ошибки

Для итоговой оценки модели использовалась метрика ROC-AUC score. Она часто считается одной из наилучших метрик для оценки производительности классификационных моделей, особенно когда речь идет о бинарной классификации. Ее будем использовать в качестве основной. Результат составил порядка ~95 процентов.

Изображение выглядит как текст, Шрифт, линия, число

Автоматически созданное описание

Рисунок 53. Результат по метрике ROC-AUC

В качестве дополнительной оценки модели использовались метрики accuracy и f1\_score.Результат составил порядка ~81 процента для accuracy

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 54. Результат по метрике accuracy

Результат для f1\_score составил ~83 процента что тоже является довольно не плохим результатом, учитывая ограниченность данных а так же видеопамяти компьютера.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 55. Результат по f1\_score

4.4 Вывод

В данной главе были предоставлены оценки работоспособности разработанного анализатора уязвимостей смарт-контрактов по различным метрикам оценивания машинного обучение. С учетом ограниченности данных и вычислительных возможностей видеокарты результат оказался довольно успешным

Разработанный анализатор уязвимостей эффективен (результат по метрике ROC-AUC составил свыше 95%, по метрикам accuracy и f1\_score свыше 80%) и может быть интегрирован как часть превентивных мер безопасности в процессе разработки и аудита смарт-контрактов. Тем не менее, для повышения точности и надёжности модели необходимо продолжать качественное обучение с учётом новых угроз и уязвимостей, что предполагает непрерывное обновление и адаптацию системы.

Заключение

Вначале работы был рассмотрен принцип функционирования блокчейна Ethereum, а также смарт-контракты и связанные с ними уязвимости. Затем были рассмотрены ныне существующие анализаторы уязвимостей смарт-контрактов, и выяснили их сильные (высокая точность) и слабые (по большей половине выборки у анализаторов нет результатов) стороны.

В третьем разделе были рассмотрели принципы построения нейронных сетей, для дальнейшего использования в создании анализатора уязвимостей. В результате проделанной работы нейронная сеть показала хорошие результаты в анализе смарт-контрактов в блокчейне Ethereum. Модель способна автоматически выявлять уязвимые смарт контракты, что помогает обеспечить безопасность и надежность их использования в блокчейне. Результат по метрике ROC-AUC составил свыше 95%, что является весьма хорошим результатом с учетом ограниченности данных и видеопамяти видеокарты. По метрикам accuracy и f1\_score результат составил свыше 80% что тоже является не плохим результатом.

В рамках данного анализа, проделанного в четвертой главе, архитектура трансформер продемонстрировала высокие результаты, как на тренировочной, так и на тестовых данных, что еще раз доказывает необычайную универсальность данной архитектуры.

Примененные методы и полученные результаты позволяют утверждать, что внедрение нейронных сетей, в частности архитектуры трансформеров, для анализа безопасности смарт-контрактов в блокчейне Ethereum имеет большой потенциал.. Последующие шаги могут включать расширение базы данных для обучения, оптимизацию архитектуры модели, а также разработку инструментов для более глубокой интерпретации и интеграции результатов анализа.

Список используемой литературы

1. Равал С. Децентрализованные приложения. Технология Blockchain в действии, // С. Равал. — СПб.: Питер, - 2017. - 192 с
2. Тапскотт Д., Тапскотт А. Технология блокчейн - то, что движет финансовой революцией сегодня, // Д. Тапскотт, А. Тапскотт. — М.: Эксмо, 2017. – 448 с
3. Ethereum.org [Электронный ресурс]. – URL: <https://ethereum.org/ru/developers/docs/gas/> / (дата обращения 01.01.2024).
4. Уязвимости смарт-контрактов.Примеры кода. [Электронный ресурс]. – URL: https://habr.com/ru/articles/407583/c. (дата обращения 08.01.2024).
5. Как работает блокчейн (Ethereum)? [Электронный ресурс]. – URL: https://habr.com/ru/articles/430602/. (дата обращения 18.01.2024).
6. Пять инструментов для анализа смарт-контрактов [Электронный ресурс]. – URL: <https://dev.to/0xfedev/5-tools-for-analysing-the-security-of-solidity-smart-contracts-in-2022-39e1/> (дата обращения 11.02.2024).
7. Лутц М. Программирование на Python. Том 2. М.: Символ-плюс, 2013. 334 c. 42
8. Круглов В. В., Борисов. В.В. Искусственные нейронные сети. Теория и практика. М.: Горячая линия - Телеком, 2002. 382 c.
9. Латыпова Р. Нейронные сети. М.: LAP Lambert Academic Publishing, 2012. 465 c.
10. Осовский. С. Нейронные сети для обработки информации. М.: Финансы и статистика, 2004. 343 с. 15.Редько В. Г. Эволюция, нейронные сети, интеллект: Модели и концепции эволюционной кибернетики. Москва: Наука, 2017. 224 c.
11. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечёткие системы. М.: ОСНОВА, 1997. 112 c.
12. Рышард Т. Элементарное введение в технологию нейронных сетей с примерами программ. М.: Горячая линия - Телеком, 2011. 661 c.
13. Хайкин. С. Нейронные сети. Полный курс. 2-e изд. Пер. с англ. М.: Издательский дом «Вильямс», 2006. 1104 с.
14. Attention is all you need простым языком [Электронный ресурс]. – URL: <https://habr.com/ru/articles/781770/> (дата обращения 11.03.2024).
15. Ошибки смарт контрактов и лучшие способы их защиты. [Электронный ресурс] URL: https://dev.to/dianamaltseva8/what-are-smart-contract-bugs-and-how-to-protect-against-them-1j96 / (дата обращения 29.03.2024).
16. Метрики в задачах машинного обучения. [Электронный ресурс] URL: https://habr.com/ru/companies/ods/articles/328372/ (дата обращения 11.04.2024).