**Что такое смарт-контракт?**

Смарт-контракт происходит от английского термина smart contract, что переводится как «умный контракт». Это самоисполняемый компьютерный код, который записывается в блокчейн — децентрализованную цепочку блоков, хранящуюся на множестве компьютеров. Он позволяет обмениваться активами — деньгами, акциями и другими видами собственности напрямую без участия третьих лиц.

В контракте прописаны все условия сделки и, если участники их выполняют, автоматически получают требуемое.

В этом и заключается одно из основных преимуществ — смарт контракт на блокчейне исключает из процесса посредников, и максимальная оптимизирует процесс. Усилий потрачено минимум, время сэкономлено, исключен человеческий фактор. Умный контракт сам все проверяет и сам себя исполняет.

**Как работает смарт контракт**

Смарт контракт и блокчейн связаны неразрывно. Ведь компьютерный алгоритм должен быть где-то записан и для этого используется так называемая децентрализованная цепочка блоков blockchain. Смарт-контракт имеет: четко прописанные условия, при достижении которых он будет выполнен; цифровые подписи участников договора; доступ к предмету договора — иначе говоря, к товарам или услугам, о которых идет речь. Смарт контракт может: отслеживать, выполнены ли все условия, прописанные в нем; самостоятельно на основе предыдущего пункта принимать решения.

**Преимущества умных контрактов:**

* Они неизменяемы и хорошо защищены от несанкционированного внесения изменений или потери данных, что обеспечивается с помощью децентрализованного хранения.
* Прозрачность — все заинтересованные пользователи могут наблюдать за развитием событий. При этом сами участники процесса сохраняют конфиденциальность.
* Они самопроверяемы и самовыполняемы.
* Смарт контракты невозможно подкупить, перетянуть на свою сторону — здесь действуют четкие математические алгоритмы.
* Снижают расходы за счет устранения из процесса посредников. Высокая скорость благодаря автоматизации — то, что сегодня занимает дни и даже недели, благодаря контракту может быть выполнено буквально в течении нескольких минут.
* Стандартизация — чем дальше, тем больше вариантов смарт-контрактов появляется, из которых можно выбирать под свои потребности.

Уязвимости смарт-контрактов:

* Умные контракты все еще находятся на своей экспериментальной стадии развития. А потому предусмотреть при их создании все вероятные баги практически нереально. Во всяком случае пока. А критические ошибки нередко могут привести к непоправимым последствиям. Из других недостатков умных контрактов можно отметить такие моменты:
* Правовое регулирование этой сферы пока оставляет желать лучшего.
* Много работы предстоит и в усовершенствовании программ-оракулов.
* Доработки требует и сама технология блокчейна, в частности, как никогда актуальны вопросы скорости обработки транзакций и масштабируемости.
* Отсутствие гибкости — все условия четко заранее запрограммированы. В отдельных случаях задекларированное ранее преимущество в виде неизменяемости может обернуться недостатком. Например, если будет допущена ошибка при вводе данных, исправить ее не представляется возможным.
* Вероятна и возможность, что данные могут быть подменены на внешнем источнике информации.

**Анализ блокчейн-платформ**

**Stellar**

**Плюсы Stellar:**

1. Высокая скорость транзакций:

Stellar берет на себя обработку около 3000 транзакций в секунду. В результате каждая транзакция проходит проверку в течение максимум 5 секунд.

2.Незначительная стоимость транзакций:

Поскольку Stellar разрабатывается как платежный протокол, который отдает приоритет прибыли гораздо меньше, чем платежный протокол, такой как Ripple. Кроме того, быстрота транзакции и высокая операционная эффективность приводят к более низкой стоимости транзакций.

(00001 XLM ~= $0.0000002 или ~ 1 цент за 100000 транзакций)

3. Дополнительный функционал:

Stellar может включать в себя такие элементы, как блокчейн-смарт-контракты, а также функции(мульти-подписи, группировка/атомарность, последовательность, временные рамки), которые расширяют функциональность платежного протокола в целом.

4. Децентрализация Stellar:

У протокола Stellar нет установленного количества валидаторов. Наличие большего пула валидаторов по сравнению с конкурентами, делает его более децентрализованным вариантом.

5. Партнерство с известными компаниями:

Теперь, когда такой технический гигант, как IBM, начал внедрять платёжный протокол Stellar, доверие к компании возросло еще больше.

6. Проверенный криптопроект:

Криптопроект уже 5 лет на рынке, можно считать его вышедшим из стадии тестирования.

7. Разработка смарт-контрактов на популярных языках программирования:

Stellar поддерживается разработку смарт-контрактов на всех популярных языках программирования: JavaScript, GO, Python и PHP.

8. Открытый код:

Stellar – это open-source блокчейн-платформа.

9. Поддержка всех существующих криптовалют и синхронизация с различными финансовыми системами;

**Минусы Stellar:**

1. Минимальный предел для получения доступа:

Вы должны иметь по крайней мере 20 монет XLM в вашем кошельке, чтобы он работал. Эта система минимального баланса может быть немного проблематичной для пользователей, которые не хотят инвестировать в XLM.

2.Подверженность инфляции Каждый год генерируется дополнительный миллиард токенов XLM.

**Cardano**

**Плюсы Cardano:**

1. Высокая скорость транзакций:

Этот показатель у Cardano достигает около 10 000 сделок в секунду.

2. Совместимость:

Cardano полностью совместим с другими блокчейнами.

3. Дополнительный функционал:

Структура платформы представлена в виде слоев, что позволяет получить большую гибкость в работе с ней.

4. Известные разработчики:

Над криптопроектом работает сразу несколько команд. Среди разработчиков известные личности, в том числе Чарльз Хоскинсон – соучредитель Ethereum.

5. Открытый код:

Cardano – это open-source блокчейн-платформа.

6. За счет использования инновационного алгоритма майнинга Ouroboros существенно сокращается расход электроэнергии для подтверждения транзакций;

7. Безопасность:

Платформа Cardano написана на языке программирования Haskell, считающемся самым защищенным языком в плане хакерских атак и критических ошибок программирования;

**Минусы Cardano:**

1.Стоимость транзакций:

Для ее расчета создатели ADA предлагают использовать специальную формулу: А+Б\*размер.

При этом А = 0,155381ADA, Б – тоже константа, равная 0,000043946 ADA/byte.

2. Проблемы Централизации:

Хотя протокол Proof of Stake действительно упрощает майнинг, он вызывает опасения по поводу централизации, позволяя только нескольким валидаторам решать, какой блок будет проверен.

3. Относительно новый проект:

Криптопроект все еще находится в стадии развития. Планы у Cardano серьезные, поэтому многое будет зависеть от их реализации.

4. Уязвимость безопасности:

Протокол Proof of Stake оставляет лазейку, где злоумышленник может удвоить расходы, отправив транзакцию, а затем просто выполнить форк блокчейна прямо перед указанной транзакцией.

5. Малое количество поддерживаемых языков для разработки смарт-контрактов:

Cardano поддерживает разработку смарт-контрактов только на языке программирования Solidity.

**NEO**

**Плюсы** **NEO:**

1. Высокая скорость транзакций:

Этот показатель у NEO достигает около 4000 сделок в секунду.

2. Разработчики взаимодействуют с китайским правительством.

3. Разработка смарт-контрактов на популярных языках программирования:

NEO поддерживается разработку смарт-контрактов на всех популярных языках программирования: JavaScript, Golang, Python, Java, Kotlin и С#.

**Минусы NEO:**

1. Проблемы Централизации.

2. OnChain имеет техническую возможность следить за транзакциями владельцев монет, передавать информацию властям, а также самолично блокировать средства на счетах пользователей и регулировать курс.

3. Все ноды сети принадлежат компании OnChain.

**Etherium**

**Плюсы Etherium:**

1. Известные разработчики:

Основатель Ethereum Виталик Бутерин часто упоминается в крипто-заголовках. Его активное участие в мире криптографии придает большое доверие к сети, которую он разработал. Кроме того, компетентное сообщество разработчиков, таких как Lubin или Zamfir, добавляет надежности и репутации этой сети.

2. Простая интеграция с биржами.

3. Проверенный криптопроект:

Первая децентрализованная платформа, которая стала ориентиром для следующего поколения и открыла миру целую индустрию новых возможностей.

4. Открытый код:

Cardano – это open-source блокчейн-платформа.

**Минусы** **Etherium:**

1. Низкая скорость транзакций:

Этот показатель у Etherium достигает около 30 сделок в секунду.

2. Проблемы Централизации:

Ситуация с The DAO в 2016 году показала, что у платформы имеются проблемы. Это вылилось в появление форка Etherium Classic.

3. Малое количество поддерживаемых языков для разработки смарт-контрактов:

Etherium поддерживает разработку смарт-контрактов на языке программирования Solidity, который не является популярным и разработан специально для него.

4. Высокая стоимость транзакций:

Комиссия за каждую транзакцию составляет в среднем 3-4 цента(может варьироваться от условия пользователей)

5. Подверженность сети перегрузкам.

**Сравнение блокчейн – платформ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Etherium | Stellar | Cardano | NEO |
| Дата запуска | 07/2015 | 09/2015 | 09/2017 | 10/2016 |
| Языки программирования | Solidity | JavaScript, GO, Python, PHP, C#, Ruby | Solidity | JavaScript, GO, Python, Java, Kotlin, С# |
| Создание блока(время) | 15 с. | 5 с. | 20 с. | 15 с. |
| Транзакций в сек. | 15 | 1000 | 200 | 1000 |
| Проблемы централизации | + | - | + | + |
| Open-source | + | + | + | + |
| Цена транзакций | 3-4 цента | 1 цент за 100000 транзакций | Минимум 0,0056$ | 0 для своей валюты и 0.001$ для других |
| Анонимность | + | + | + | - |
| SDK | + | + | + | + |

После сравнения блокчейн-платформ, я считаю наиболее подходящей Stellar. Stellar проверенная временем, безопасная, быстрая блокчейн-платформа, позволяющая разрабатывать на многих языках программирования смарт - контракты с низкой комиссией.

**Методы анализа кода**

Статический анализ кода - это процесс выявления ошибок и недочетов в исходном коде программ. Статический анализ можно рассматривать как автоматизированный процесс обзора кода. Остановимся на обзоре кода чуть подробнее.

Обзор кода (code review) – один из самых старых и надежных методов выявления дефектов. Он заключается в совместном внимательном чтении исходного кода и высказывании рекомендаций по его улучшению. В процессе чтения кода выявляются ошибки или участки кода, которые могут стать ошибочными в будущем. Также считается, что автор кода во время обзора не должен давать объяснений, как работает та или иная часть программы. Алгоритм работы должен быть понятен непосредственно из текста программы и комментариев. Если это условие не выполняется, то код должен быть доработан.

Задачи, решаемые программами статического анализа кода можно разделить на 3 категории:

* Выявление ошибок в программах.
* Рекомендации по оформлению кода. Некоторые статические анализаторы позволяют проверять, соответствует ли исходный код, принятому в компании стандарту оформления кода. Имеется в виду контроль количества отступов в различных конструкциях, использование пробелов/символов табуляции и так далее.
* Подсчет метрик. Метрика программного обеспечения - это мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций.

Главное преимущество статического анализ состоит в возможности существенной снижении стоимости устранения дефектов в программе. Чем раньше ошибка выявлена, тем меньше стоимость ее исправления. Так согласно данным, приведенным в книге Макконнелла "Совершенный Код", исправление ошибки на этапе тестирования обойдется в десять раз дороже, чем на этапе конструирования (написания кода).

Другие преимущества статического анализа кода:

* Полное покрытие кода. Статические анализаторы проверяют даже те фрагменты кода, которые получают управление крайне редко. Такие участки кода, как правило, не удается протестировать другими методами. Это позволяет находить дефекты в обработчиках редких ситуаций, в обработчиках ошибок или в системе логирования.
* Статический анализ не зависит от используемого компилятора и среды, в которой будет выполняться скомпилированная программа. Это позволяет находить скрытые ошибки, которые могут проявить себя только через несколько лет. Например, это ошибки неопределённого поведения. Такие ошибки могут проявить себя при смене версии компилятора или при использовании других ключей для оптимизации кода.
* Можно легко и быстро обнаруживать опечатки и последствия использования Copy-Paste. Как правило, нахождение этих ошибок другими способами является кране неэффективной тратой времени и усилий. Обидно после часа отладки обнаружить, что ошибка заключается в выражении вида "c"=="c". Обсуждая типовые ошибки, про такие ляпы, как правило, не вспоминают. Но на практике на их выявление тратится существенное время.

Динамический анализ кода - это способ анализа программы непосредственно при ее выполнении. Процесс динамического анализа можно разбить на несколько этапов - подготовка исходных данных, проведение тестового запуска программы и сбор необходимых параметров, анализ полученных данных.

Достоинства динамического анализа кода:

* В большинстве реализаций появление ложных срабатываний исключено, так как обнаружение ошибки происходит в момент ее возникновения в программе; таким образом, обнаруженная ошибка является не предсказанием, сделанным на основе анализа модели программы, а констатацией факта ее возникновения;
* зачастую не требуется исходный код; это позволяет протестировать программы с закрытым кодом.

Недостатки динамического анализа кода:

* динамический анализ обнаруживает дефекты только на трассе, определяемой конкретными входными данными; дефекты, находящиеся в других частях программы, не будут обнаружены;
* не может проверить правильность работы кода, что код делает то, что должен;
* требуются значительные вычислительные ресурсы для проведения тестирования;
* только один путь выполнения может быть проверен в каждый конкретный момент времени, что требует большого количества тестовых запусков для большей полноты тестирования;

Из вышеперечисленного можно сделать вывод, что для своевременного создания оптимизированного и надежного смарт-контракта надо комбинировать статистический и динамический анализы.

**Анализ методик**

Инструментирование(code instrumentation) - это отслеживание параметров уровня производительности кода, возможность диагностировать ошибки и записывать информацию на всём протяжении работы для отслеживания причин их возникновения. Таким образом можно проверить смарт-контракт в «боевых» условиях.

Символьное выполнение(symbolic execution) – это средство анализа программы для определения того, какие входные данные вызывают выполнение какой части программы. Интерпретатор следует за программой, принимая символические значения для входных данных, а не получая фактические входные данные, как это было бы при обычном выполнении программы. Другими словами, оно позволяет находить недочеты в коде, даже не зная, какое значение переменных будет в строке с ошибкой.

Решение ограничений(Constraint solving) означает определение разрешимости ограничений и возможность вычисления конкретного решения. Ограничение - это набор условий, которым должны удовлетворять переменные. В нашем контексте ограничения в основном возникают из условий ветвления в коде.

Абстрактная интерпретация(Abstract interpretation) игнорирует определенные инструкции или определенные эффекты инструкций при выполнении байт-кода. Это можно сделать, переведя инструкции в другой формализм, например DataLog, а затем изучив все возможные варианты выполнения.

Логика Хорна(Horn Logic)-это ограниченная форма логики первого порядка, где все формулы (предложения) являются правилами if-then. Хотя логика Хорна и ограничена, она все же является вычислительно универсальной, поэтому может выполнять те же вычисления, что и любой компьютер.

Проверка моделей(Model checking) - это метод автоматической проверки правильности свойств конечных систем. Для этого требуется модель системы, которая затем проверяется на соответствие заданной спецификации.

По моему мнению, символьное выполнение является рабочей лошадкой современного анализа программ. Это один из методов, который вырос из исследований и затем стал использоваться во множестве приложений. Например, на сегодня в Microsoft есть система под названием SAGE, которая работает с большим количеством важного программного обеспечения Microsoft, начиная от Power Point и заканчивая самой Windows, чтобы реально находить проблемы безопасности и уязвимости.

**Анализ инструментов для анализа смарт-контракта**

Scompile берет байт-код контракта, строит CFG, определяет все вычислительные пути, включающие любой поток эфира, выбирает те, которые соответствуют паттернам, характерным для определенных уязвимостей, ранжирует их эвристически в соответствии с релевантностью и, наконец, применяет символическое исполнение, прежде чем представить результат пользователю для ручной проверки.

Mythril - это инструмент командной строки в Python для интерактивного анализа смарт-контрактов. Он выполняет байт-код виртуальной машины символически и визуализирует CFG, причем узлы, содержащие разобранный код, и ребра помечаются формулами пути. Проверенные уязвимости подробно описаны в интерактивной документации. Mythril разрабатывается и поддерживается компанией Consensus, а также доступен на Github под лицензией MIT с сентября 2017 года.

Securify принимает байт-код EVM и свойства безопасности в качестве входных данных. Инструмент декомпилирует байт-код, ориентированный на стек, в форму, основанную на присвоении, и представляет код в виде фактов журнала данных. Затем он выводит дополнительные факты, которые описывают управление и поток данных в абстрактной форме. Свойство безопасности состоит из паттернов соответствия и нарушения, чрезмерно приближенных как к удовлетворению, так и к неудовлетворению этого свойства. Эти паттерны кодируются как правила ведения журнала данных, которые можно сверить с фактами, используя Soufflé. Этот подход гарантирует, что при обнаружении паттерна код определенно обладает/нарушает соответствующее свойство безопасности. Этот инструмент написан на Java и доступен на Github под лицензией Apache 2.0 с сентября 2018 года. Кроме того, доступ к закрытой версии исходного кода можно получить через веб-сайт компании Chain Security.

Manticore использует символическое выполнение, чтобы найти уникальные пути вычислений в ЭВМ и двоичные ELF-файлы. Он записывает соответствующие следы выполнения. Что касается ЭВМ, то Мантикора компилирует код солидности в байт-код для его анализа, проверяет трассировки на наличие уязвимостей, таких как повторное проникновение и достижимые операции саморазрушения, и сообщает о них в контексте исходного кода. Информация о методах и их ограничениях весьма скудна. Этот инструмент разработан и поддерживается компанией Trail of Bits и доступен на Github под лицензией AGPL-3.0 с февраля 2017 года. Его можно использовать из командной строки или через API Python.

MAIAN рассматривает атаки, требующие нескольких транзакций. Он выполняет байт-код VM символически и проверяет наличие следов исполнения, указывающих на то, что контракт может быть самоуничтожен или слит эфира с произвольных адресов, или что он принимает эфир без выплаты. Чтобы отбросить ложные срабатывания, контракты динамически анализируются, развертывая их на частном блокчейне и атакуя их с помощью вычисленных транзакций. MARIAN написана на Python и доступна на Github под лицензией MIT с марта 2018 года. Он использует компилятор солидности для компиляции исходного кода в байт-код и Go-Ethereum для запуска частного блокчейна. Mayan -это в основном инструмент командной строки, но также предоставляет простой графический пользовательский интерфейс, который требует графической библиотеки Qt.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Инструменты** | **SCompile** | **Manticore** | **Mythril** | **Securify** | **MAIAN** |
| Проблемы с безопсностью | Галочка | Галочка | Галочка | Галочка | Галочка |
| Эксплоиты |  | Галочка | Галочка |  | + |
| Общий анализ | Галочка |  |  | Галочка | Галочка |
| Байткод |  | + | + | + | + |
| Solidity code | Галочка |  |  |  |  |
| Статический анализ | Галочка | Галочка | Галочка | Галочка | Галочка |
| Динамический анализ |  |  |  |  | Галочка |
| Contextualization | Галочка | Галочка | Галочка | Галочка |  |
| Disassembly | Галочка | Галочка | Галочка | Галочка | Галочка |
| Control ﬂow graph | Галочка |  | Галочка |  | Галочка |
| Call graph |  |  |  |  |  |
| ASTanalysis |  |  |  |  |  |
| Decompilation |  |  |  | Галочка |  |
| «Инструментарий» |  |  |  |  |  |
| «Символьное выполнение» | Галочка | Галочка | Галочка |  | Галочка |
| «Решение ограничений» | Галочка | Галочка | Галочка |  | Галочка |
| «Абстрактная интерпретация» |  |  |  | Галочка |  |
| «Логика хорна» |  |  |  | Галочка |  |
| «Проверка модели» |  |  |  |  |  |

Mythril способен обнаруживать большой спектр потенциальных проблем. Ко всем прочему, он умеет делать это все в реальном блокчейне, находя по сигнатурам нужные контракты и уязвимости, строить красивые графы вызовов, форматировать отчеты. Также, Mythril позволяет писать собственные сценарии тестирования, предоставляя python-based интерфейс к контракту и позволяющие с произвольной степенью гибкости протестировать отдельные функции, зафиксировать значения параметров, или вообще реализовать собственную стратегию работы с дизассемблированным кодом.

Я считаю, что Mythril — отличный и крайне мощный инструмент для анализа смарт-контрактов и на данный момент должен быть в арсенале любого аудитора. Он позволяет как минимум обратить внимание на критические части кода и обнаружить скрытые связи между переменными.

Подводя итоги, самой сбалансированной, по моему мнению, связкой методики и инструмента является символьное выполнение + Mythril.