Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИЧЕРСИТЕТ»**

**(МОСКОВСКИЙ ПОЛИТЕХ)**

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА «ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ»

Отчет выполнения по Проектной деятельности

**«Оценка уязвимостей смарт-контрактов»**

Выполнила: Николаец Д.И

Студент 3-го курса

Группы 171-361

Москва 2020

**Цели задачи 1** проекта «Оценка уязвимостей смарт-контрактов»:

* Анализ 4-5 Blockchain-платформ для написания смарт-контрактов, выделение особенностей Blockchain-платформ, преимуществ и недостатков, а также описание характеристики. Принцип разработки смарт -контрактов на каждой из Blockchain-платформ
* Сравнение проанализированных Blockchain-платформ и выбор наиболее оптимальной платформы.

**Результат работы:**

* Рассмотрен сравнительный анализ Blockchain платформ Дана оценка проблеме доверительных отношений между участниками сделки и описан способ предотвращения проблем, при их возникновении. Приведены примеры глобальных и приватных Blockchain - платформ, их характеристики, а также плюсы и минусы использования данных структур.
* На основе плюсов и минусов Blockchain-платформ сформирован вывод какую платформу целесообразно использовать для работы со смарт-контрактами.

Как выбрать правильную платформу для работы со смарт-контактами и на что стоит обратить внимание в первую очередь?

Для того, чтобы заключить любую сделку, необходимо обратиться к нотариусу или адвокату, оплатить документы и ждать их оформление. Зачастую, многие пункты этих документов содержат ссылки на законодательные статьи, которые можно интерпретировать под себя, обойти. В случае невыполнения условий сделки, в реальной жизни людям приходится обращаться в суд, снова тратить деньги на процесс и доказывать свою правоту. При заключении таких сделок вообще не может идти речь о доверии участников договора.

Smart-контракты дают возможность безопасно обмениваться деньгами, акциями, собственностью и другими активами напрямую, без участия посредников.

Для этого была разработана программа, которая следит за выполнением обязательств обеих сторон, прописанных в контракте, а также автоматически взымает штрафы за нарушение или невыполнение условий сделки. Умные контракты обеспечивают безопасность сделки и лишены риска неоднозначной трактовки условий, благодаря тому, что основаны на криптографии. Это более выгодные сделки в материальном плане, так как человеку не нужно платить юристам, посредникам или подавать в суд при невыполнении контракта. При чем выполнение условий сделки происходит автоматически с минимальными затратами на их сопровождение, без привлечения третьих лиц (посредников).

Умный контракт (англ. Smart contracts) — электронный алгоритм, описывающий набор условий, выполнение которых влечет за собой некоторые события в реальном мире или цифровых системах. Для реализации умных контрактов требуется децентрализованная среда, полностью исключающая человеческий фактор, а для возможности использования в умном контракте передачи стоимости требуется криптовалюта.

Для анализа я выбрала 5 популярных Blochain-platform: ***Ethereum***, ***EOS, NEO, WAVES, STELLAR***. Рассмотрим характеристики платформ, их преимущества и недостатки.

***Ethereum*** является первой блокчейн платформой в которой возможно исполнение тьюринг-полных смарт-контрактов. Полнота по Тьюрингу в смарт-контрактах позволяет выполнить любую математически вычислительную функцию. Ethereum завоевал популярность у пользователей благодаря возможности выпуску токенов на платформе Эфириума, легкому проведению ICO или DAICO, созданию децентрализованных приложений, а также весьма гибким возможностями по созданию смарт-контрактов.

В настоящий момент Ethereum является самой популярной криптовалютой среди разработчиков и наиболее распространенной по числу децентрализованных приложений. Большинство смарт-контрактов разработанных нашей командой также созданы с помощью языка Solidity и виртуальной машины Ethereum. Сфера применений смарт-контрактов в Ethereum чрезвычайно широка и любую логически поданную идею можно реализовать с помощью данной сети.

***EOS*** на сегодняшний день является второй по популярность блокчейн платформой по применению смарт-контрактов и первой по активному числу транзакций. Отличительной особенностью EOS можно назвать масштабируемость, параллельные вычисления и высокую пропускную способность сети. В EOS используется алгоритм консенсуса Delegated Proof-of-Stake что повышает пропускную способность сети, но плохо сказывается на её децентрализации.

***NEO*** - блокчейн-платформа разрабатываемая сообществом китайских программистов. Изначально блокчейн NEO создавался как альтернатива Ethereum, но с развитием конкуренции в этой области разработчики NEO стали внедрять собственные решения. Монеты NEO были изначально добыты премайном, где половина была продана на ICO и половина осталась в руках разработчиков. Общая эмиссия NEO - 100 миллионов монет.

Интересным отличием NEO является поддержка популярных языков программирования для разработки смарт-контрактов, например таких как Java, F#, C#, Kotlin, Go и Python. Ещё одной особенностью является использование отдельной криптовалюты GAS для оплаты транзакций и выполнения смарт-контрактов, но в отличие от газа в Ethereum криптовалюта GAS автоматически начисляется на баланс пользователя при хранении NEO на аккаунте. Смарт-контракты NEO не поддерживают сторонние библиотеки, а используется только библиотека зависимостей NEO-SmartContract-DevPack. Байт код смарт-контрактов представлен в формате .avm. Развертывание смарт-контракта в блокчейне NEO стоит 100-1000 GAS.

***Stellar*** появился в 2014 году, когда часть команды провела хардфорк Ripple и запустила собственную сеть, предназначенную в первую очередь для валютных операций. Ноды Stellar хранят записи о всех счетах в сети и синхронизируются между собой образуя так называемую бухгалтерскую книгу. Алгоритм консенсуса сети позволяет проводить транзакцию за 4-5 секунд.

Stellar позволяет комфортно управлять микротранзакциями, в сети низкая комиссия и повышенная технологическая совместимость с финансовым рынком. Блокчейн Stellar поддерживает работу смарт-контрактов, однако они обладают ограничениями и сложный функционал на нём невозможен. В то же время на Stellar доступны мультиподписи, атомарность транзакций их последовательность и временные ограничения.

***Waves*** - блокчейн платформа имеющая российское происхождение была основана в 2016 году. Её отличительно особенностью является алгоритм консенсуса LPoS, что расшифровывается как Leased Proof of Stake или арендованное подтверждение доли. Владельцы монет Waves могут отдавать их в аренду нодам и получать за это вознаграждение от её работы при создании новых блоков блокчейна.

В основной сети Waves сегодня работают ограниченные по функционалу смарт-контракты которые позволяют выполнять следующие сценарии: создание токена, заморозка токена, выплаты по расписанию, двухфакторная аутентификация, кошельки с мультиподписью, смарт-аккаунты, смарт-активы, атомарные свопы, работа с оракулом внешних данных.

В таблице представлено сравнение наиболее популярных блокчейн-платформ.

|  |  |  |
| --- | --- | --- |
| Платформа | Достоинства | Недостатки |
| Ethereum | Широкое распространение  Гибкость  Низкий порог вхождения  Четкие рекомендации для разработчиков  Много литературы / справок доступно | Оплата транзакций в сети (gas)  Нагрузка на сеть  Проблемы с безопасностью  Проблемы масштабируемости  Относительно медленное подтверждение транзакции  Дороже, чем другие платформы |
| NEO | Алгоритм консенсуса dBFT  Высокая скорость транзакций  Централизация     * Поддержка многих языков программирования, расширяющая перспективы коммерческого применения платформы. | Разработчики могут самостоятельно влиять на сеть и её участников – замораживать счета, следить за операциями, предоставлять данные властям по их требованию.  Перспективы NEO находятся в большой зависимости от политической воли руководства Китайской Народной Республики.  Отсутствие анонимности. |
| EOS | Высокая масштабируемость  Высокая скорость транзакций  Достаточно широкий функционал  Бесплатные транзакции | Отсутствие главной сети EOS. Чрезмерная централизация |
| STELLAR | Высокая скорость  Микроплатежи | Не очень подходит для разработки более сложных смарт-контрактов  Ограниченный функционал |
| WAVES | Высокая скорость  Имеет сильные стратегические партнерства  Имеет функционирующую децентрализованную биржу  Алгоритм консенсуса LPoS  Приватный блокчейн Vostok | Есть некоторые проблемы в области безопасности |

Рассмотрим непосредственно характеристики блокчейн-платформ.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NAME | Ethereum | Waves-NG | EOS | NEO | Stellar |
| Mainnet launch | 07/2015 | 04/2016 | 06/2018 | 10/2016 | 09/2015 |
| Contract Language | Solidity (Serpent Mutan LLL ) | Ride | C++, С | C#, JAVA, JS, PYTHON.NET | Js,go,python,php,swift, ruby, scala |
| Price | 3-4 цента | 0.001-0.009 $ | 0.03 $ | 0 для своей валюты и 0.001 для других | 1 цент для 10000 транзакций |
| TPS | 12-25 p/s | 100 p/s | Max:4000  Avg:50 | 1000 p/s | 2000 p/s |
| SDK | + | + | + | + | + |
| Consensus mechanixm | PoW | LPos | DPos | dBFT | SCP |
| Block time | 15 seconds | 3 sec | 0.5 sec | 15 Sec | 5 sec |

Cложно выбрать «лучшую» платформу. Все они имеют свои преимущества и недостатки. Что важно иметь в виду, так это выбрать платформу, которая лучше всего подходит для вас.

* Сфера применений смарт-контрактов в Ethereum чрезвычайно широка и любую логически поданную идею можно реализовать с помощью данной сети. Если вы хотите платформу, которая имеет сильное сообщество разработчиков, то вы идете на Ethereum.
* Если вы хотите платформу, которая подчеркивает скорость и масштабируемость, но не очень заботится о децентрализации, то EOS-это именно то, что вам нужно.
* Waves-это перспективная платформа, которая делает ее простой для среднего человека, работающего с blockcano, crowdfunding и помогает ему обналичивать криптовалюты. Если вы хотите платформу, которая обладает высокой масштабируемостью и проводит невероятно быстрые транзакции, то вам подходит Waves. Главным ключевым моментом этой платформы является то, что даже новички могут использовать эту платформу, поскольку это не требует высокого уровня технического языка.
* Блокчейн Stellar поддерживает работу смарт-контрактов, однако они обладают ограничениями и сложный функционал на нём невозможен. В то же время на Stellar доступны мультиподписи, атомарность транзакций их последовательность и временные ограничения.
* Но если вы хотите платформу, которая соединяет банки, платежные системы и людей, то ваш выбор STELLAR. Интегрируйтесь, чтобы быстро, надежно и почти бесплатно перемещать деньги.
* Если вы хотите платформу, которая имеет все преимущества виртуальной машины Ethereum, но отличается большей масштабируемостью и не мешает разработчикам языковыми барьерами, то ваш выбор NEO. В ethereum вам нужно будет научиться основательно кодировать смарт-контракты, в то время как в Neo вы можете даже использовать Javascript для кодирования смарт-контрактов.

Считаю, что лучшей платформой для написания смарт-контрактов является EOS. Рассмотрев ее характеристики, преимущества и недостатки, можно сделать вывод, что платформа EOS является технологически интересной и потенциально востребованной в широком диапазоне применений смарт-контрактов.

**Цели задачи 2** проекта «Оценка уязвимостей смарт-контрактов»:

* Изучение принцип работы методов анализа уязвимостей смарт-контрактов (статический и динамический), анализ их преимуществ и недостатков
* Выбор популярных существующих методик и инструментов анализа уязвимостей смарт-контрактов. Анализ и сравнение методик и инструментов путем выделения их особенностей
* Выбор наиболее сбалансированную комбинацию методики и инструмента, которую можно использовать для поиска и анализа уязвимостей в коде смарт-контракта.

**Результат работы:**

* Рассмотрен сравнительный анализ методик и инструментов. Приведены примеры методик и инструментов анализа уязвимостей смарт-контрактов, их характеристики
* На основе анализа преимуществ и недостатков, сформирован вывод какую наиболее сбалансированную комбинацию методики и инструмента, можно использовать для поиска и анализа уязвимостей в коде смарт-контракта.

Рассмотрим два направления в анализе безопасности уязвимостей статический и динамический анализ.

**DAST**   
  
Динамический анализ приложений методом черного ящика самый простой и распространённый способ поиска уязвимостей. Собственно, каждый раз «вставляя кавычку в URL» или вводя '>  мы осуществляем эту мудреную процедуру. По сути это [fault injection](http://en.wikipedia.org/wiki/Fault_injection) (aka fuzzing) приложения, путем эмуляции клиентской части и попытки отправки на вход «хорошо известных плохих данных ».  
Простота метода приводит к большому количеству реализаций, и в «магическом квадранте» Gartner просто тесно от конкурентов. Более того, «движки» DAST присутствуют в большинстве систем контроля уязвимостей и соответствия стандартов, таких как MaxPatrol, за исключением разве что Symantec Control Compliance Suite. Распространены и некоммерческие решения, так базовый (очень-очень базовый) модуль DAST присутствует в Nessus, более продвинутые механизмы существуют в w3af или sqlmap.  
К преимуществам DAST относится простота использования и отсутствие необходимости доступа к серверной части приложения. Также серьезным плюсом является относительная независимость от платформы, фреймворков и языков, на которых разработано приложение. Учет этих нюансов может повысить эффективность анализа, но это скорее оптимизация, добавляющая проценты к общей эффективности.  
  
Однако у DAST есть и обратная сторона. Перечислим списком:

* Невысокая степень покрытия. Далеко не все вызовы API и точки входа можно легко обнаружить методом черного ящика.
* Драматическое падение эффективности при усложнении клиента/протокола. Приложения Web 2.0, JSON, Flash, HTML 5.0 и JavaScript требует либо динамического (например, эмуляцией выполнения JavaScript) либо статического разбора («отгрепать» Flash или taint-анализ того же JavaScript) клиентской части, что значительно усложняет клиентскую часть фазера, приближая его к «полноценному» браузеру.
* Ненулевая вероятность нарушения целостности и доступности (например при попадании конструкций типа or 1=1 в UPDATE через SQL Injection).
* Долгое время работы. Вашему покорному слуге практически ни разу не удавалось наблюдать завершения работы большинства из утилит класса DAST на достаточно «разлапистом» сайте, раньше заканчивалось окно на тестирование. Тут хорошо подходит правило Парето – за 20% времени найти 80% уязвимостей.
* Сложность выявления многих типов. Например, ошибки использования криптографии, такие как слабые механизмы генерации cookie или session ID (кроме самых примитивных случаев) DAST обнаруживает крайне плохо.

**SAST: Quick and dirty**  
  
Многие из методов статического анализа генерируют большое количество «подозрений на уязвимость», которые на проверку оказываются ложными срабатываниями, что существенно увеличивает трудозатраты. Невозможность выявления некоторых классов уязвимостей. Собственно «статичность». Примеров тут множество, и генерируемый «на лету» код, и хранение кода/данных в СУБД, файловой системе и т.д. Отдельных трудозатрат разработчикам добавляет зависимость от языков (и даже версий),

фреймворков. Так, чтобы понять точки входа и точки выхода, фильтрующие функции недостаточно понимания «голого» языка, необходимо распознавать те библиотеки и фреймворки, которые используют разработчики в реальном мире.

И у SAST и DAST есть свои преимущества и недостатки и достаточно долгое время муссируется идея об объединении результатов работы этих методов или о «гибридном анализе» (hybrid analysis), что позволит взять лучшее из этих двух подходов. Не смотря, на то, что данная гипотеза, что называется «лежит на поверхности» и относится к интуитивно понятным, реализация данной концепции на практике не дает ожидаемых результатов.

*Далее рассмотрим частые методики.*

**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.

**Устранение 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.

*Рассмотрим инструменты:*

**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.

Есть много академических проектов, которые оказали большое влияние на реальный мир, например, на обнаружение важных ошибок в программном обеспечении с открытым исходным кодом, с помощью символьного выполнения. Я считаю, что красота **символьного выполнения** как техники заключается в том, что по сравнению с тестированием, это дает вам возможность представить, как ваша программа будет себя вести с потенциально бесконечным набором возможных входных данных. Это позволяет исследовать массивы входных данных, которые было бы совершенно нецелесообразно и непрактично исследовать, скажем, случайным тестированием даже при наличии очень большого количества тестеров. С другой стороны, по сравнению с более традиционными методами статического анализа оно имеет следующее преимущество. При исследовании проблемы символьное выполнение может создать входные данные и трассировку, путь выполнения, которые можно запустить в реальной программе и выполнить эту программу на основе этих входных данных. И после этого мы можете выявить реальный баг и приступить к его исправлению, используя традиционные механизмы отладки. И это особенно ценно, когда вы находитесь в промышленной среде разработки, где у вас, вероятно, нет времени, чтобы заботиться о каждой маленькой проблеме в вашем коде.

Для методики «символьное выполнение» лучшим инструментом является **MAIAN**.

**Maian** анализирует смарт-контракты, определенные в файле с помощью:

* Солидность исходного кода
* Источник байт-кода
* Байт-код скомпилирован (т. е. код, сидящий на блокчейне Maian проверяет наличие трех типов контрактов на баги:

**Maian** проверяет наличие трех типов контрактов на баги:

* Самоубийственные контракты (могут быть убиты кем угодно, например, контракт с библиотекой Parity Wallet)
* Блудные контракты (может отправить эфир кому угодно)
* Жадные контракты (никто не может выйти из эфира)

Рассмотрим **сравнение** инструментов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TOOLS** | **SCompile** | **Manticore** | **Mythril** | **Securify** | **MAIAN** |
| Security issues | Галочка | Галочка | Галочка | Галочка | Галочка |
| Exploits |  | Галочка | Галочка |  | Галочка+ |
| Formal guarantees |  |  |  | Галочка |  |
| Bulkanalysis | Галочка |  |  | Галочка | Галочка |
| Bytecode |  | Галочка+ | Галочка+ | Галочка+ | Галочка+ |
| Solidity code | Галочка |  |  |  |  |
| Static analysis | Галочка | Галочка | Галочка | Галочка | Галочка |
| Dynamic analysis |  |  |  |  | Галочка |
| Contextualization | Галочка | Галочка | Галочка | Галочка |  |
| Disassembly | Галочка | Галочка | Галочка | Галочка | Галочка |
| Control ﬂow graph | Галочка |  | Галочка |  | Галочка |
| Call graph |  |  |  |  |  |
| ASTanalysis |  |  |  |  |  |
| Decompilation |  |  |  | Галочка |  |
| **Code instrumentation** |  |  |  |  |  |
| **Symbolic execution** | Галочка | Галочка | Галочка |  | Галочка |
| **Constraint solving** | Галочка | Галочка | Галочка |  | Галочка |
| **Abstract interpretation** |  |  |  | Галочка |  |
| **Horn logic** |  |  |  | Галочка |  |
| **Model checking** |  |  |  |  |  |

Статический анализ(Dynamic analysis)

Динамический анализ (Dynamic analysis)

Символьное выполнение(Symbolic execution)

Решение ограничений(Constraint solving)

Абстрактная интерпретация(Abstract interpretation)

Логика хорна (Horn logic)

Проверка модели(Model checking)

Проблемы безопасности(Security issues): обнаружение уязвимостей и потенциальных проблем безопасности.

Эксплойт(Exploit): генерирует эксплойты, причем **+** указывает на мультитранзакционные эксплойты.

Формальные гарантии(Formal guarantees): доказывает, что договор имеет определенное свойство. Массовый анализ(Bulk analysis): подходит для анализа больших наборов контрактов