DOI 10.37882/2223–2966.2022.07–2.15

ОЦЕНКА РАБОТОСПОСОБНОСТИ ИСХОДНОГО КОДА СМАРТ-КОНТРАКТА МОРСКОГО КОНОСАМЕНТА С ПОМОЩЬЮ МОДУЛЬНОГО ТЕСТИРОВАНИЯ

RELIABILITY ASSESSMENT OF A SMART CONTRACT SEA BILL OF LADING

E. Poleshchuk
I. Shcherbinina
S. Putilova

Summary. To date, distributed ledger technology is integrated into many types of economic activity. Smart contracts are used in various business areas as a tool for implementing the terms of transactions in software form, ensuring the reliability and efficiency of electronic document management. However, the specificity of this technology is associated with the difficulties of making changes about detecting software defects. Therefore, the process of organizing testing of this software should take into account these features, as well as minimize the risks of information security. In this paper, the authors describe the procedure for testing a smart contract that implements the functions of a shipping contract for errors that may arise as a result of vulnerabilities in the smart contract code. Checking the execution of the smart contract logic is carried out by applying unit tests to individual code functions.

Keywords: information security, blockchain, smart contract, marine logistics, bill of lading, testing.

Полещук Евгения Михайловна
Аспирант, Морской Государственный Университет им. адм. Г.И. Невельского
poleshuk@msun.ru
Щербинина Инна Александровна
К.п.н., доцент, Морской Государственный Университет им. адм. Г.И. Невельского
shcherbinina@msun.ru
Путилова Софья Евгеньевна
Аспирант, Морской Государственный Университет им. адм. Г.И. Невельского

Аннотация. В настоящее время технология распределённого реестра внедрена во многие виды экономической деятельности. Смарт контракты используются в различных сферах бизнеса в качестве инструмента документирования условий сделок в электронном виде, обеспечивая надёжность и эффективность электронного документооборота. Специфика применения данной технологии состоит в сложности внесения изменений при обнаружении дефектов программного обеспечения. Следовательно, процесс организации проведения испытаний данного ПО должен учитывать эти особенности, а также максимально сократить риски информационной безопасности. В данной работе авторы описывают процедуру тестирования смарт-контракта, реализующего функции договора морских перевозок, на предмет ошибок, которые могут возникнуть в результате уязвимостей в коде смарт-контракта. Проверка исполнения логики смарт-контракта проводится посредством применения юнит-тестов к отдельным функциям кода.

Ключевые слова: информационная безопасность, блокчейн, смарт-контракт, морская логистика, коносамент, тестирование.

Обеспечение качества программного продукта включает набор действий для проверки процессов обеспечения качества, таких как верификация, валидация, и формирование документа по управлению качеством.

В стандарте ISO 9126–01 выделяются внешние и внутренние характеристики качества. Первые отражают требования к программному продукту. Внутренние характеристики качества ПО используются при планировании путей достижения необходимых внешних характеристик качества для конечного программного продукта.

Окончательная оценка качества проводится в соответствии со стандартом ISO 15504–98. Качество может повышаться за счёт постоянного улучшения используемого продукта в связи с процессами обнаружения, устранения и предотвращения сбоев/дефектов [1].

Целью настоящей работы является анализ исходного кода, исследуемого смарт-контракта на предмет уязвимостей в коде.

В качестве метода исследования выбрано модульное тестирование программного обеспечения, при котором

анализу подвергаются отдельные блоки кода с целью проверки работоспособности каждой единицы. В данном случае в качестве программных единиц служат отдельные функции смарт-контракта [6].

Описание объекта исследования

Объектом настоящего исследования является смарт-контракт, реализующий функции договора морского коносамента на базе технологии блокчейн. В традиционном понимании коносамент является весьма сложным по своей структуре документом и включает несколько участников процессов морских грузоперевозок. Смарт-контракт, реализующий функции такого договора, представляет сложную логику, и предназначен для представления условий договора морских грузоперевозок и исполнения его функций в электронном виде:

- идентификация участников цепочки поставки;
- ввод информации о состоянии груза;
- расписка перевозчика в получении груза для перевозки;
- подтверждение договора перевозки груза;
- передача прав на перевозимый груз.

Средства реализации, рассматриваемого смарт-контракта

- язык программирования: Golang;
- тип и версия операционной системы: Linux XFCE v. 21.1.2;
- блокчейн-платформа: Hyperledger Fabric.

Для реализации логики смарт-контракта Hyperledger Fabric использует понятие чейнкод, поэтому для проверки качества его исполнения применима общая логика тестирования смарт-контрактов.

Тем не менее необходимо учитывать узкую направленность договора морского коносамента (поля коносамента), а также особенности технологии (генерация блоков, формирование транзакций в соответствии с используемым протоколом консенсуса).

Таким образом, научная новизна данной работы заключается в оценке использования модульного метода тестирования в применении к специфическому программному обеспечению (смарт-контракту) на базе технологии блокчейн.

Чейнкод даёт возможность участникам цепочек контрактов получить автономность и независимость от третьих лиц и структур, а также обеспечивают иными преимуществами:

- надёжность;
- безопасность;
- точность;
- экономичность.

Однако, в случае возникновения ошибок или инцидентов информационной безопасности невозможно изменить чейнкод, развёрнутый в сети [3].

Таким образом, даже небольшая ошибка в логике чейнкода может привести к серьезным последствиям. Так, например, летом 2016 года крипто кошельки Ethereum были заблокированы из-за уязвимостей в смарт-контрактах. Хотя уязвимости можно было обнаружить на этапе тестирования до выпуска продукта на рынок.

При проведении испытаний чейнкода необходимо обеспечить максимально тщательную проверку следующих прецедентов:

- применение электронной подписи;
- наличие кода контракта и фиксация его изменений;
- предмет договора;
- инструменты, которые нужны для исполнения контракта;
- условия исполнения, зафиксированные в коде договора;
- события;
- ошибки и отправка сообщений;
- изменений состояния договоров, а также их баланса.

При этом необходимо смоделировать и учитывать все возможные условия, заложенные в логике договора:

- какие события должны инициироваться;
- какие методы должны выполняться;
- как изменяется состояние контракта;
- какую ошибку вызывать;
- проверить отправителя сообщения и правильность использования текущего времени [4].

В таком случае для тестирования чейнкода создаётся набор эталонных вызовов методов в предопределённом окружении, для которых прописываются ожидаемые результаты [7].

Для достижения поставленной цели данной работы определены следующие задачи:

- сформировать список функций, содержащихся в коде рассматриваемого смарт-контракта;
- для каждой функции кода написать функцию тестирования. Полученные функции кода скомпилировать в файл-скрипт;
- провести тестирование смарт-контракта посредством юнит-тестов;
- сделать выводы о работе функций код смарт-контракта.

Рис. 1. Запрос переданных параметров на стороне T

При тестировании рассматриваемого смарт-коносамента мы использовали юнит-тесты. Для каждой вызываемой функции чейнкода написана отдельная функция для тестирования. В них указывается, какая именно функция вызывается, поскольку на каждом узле вызывается определённый набор команд [2].

Таким образом, для проведения данного тестирования мы имеем систему, которая состоит из файла-скрипта, содержащего исходный код рассматриваемого смарт-контракта, и файла-скрипта, содержащего функции тестирования отдельных функций кода.

Функции тестирования запускаются с помощью команд вида “go test”, посредством командного интерпретатора BASH. С помощью bash производится чтение команд из файла-скрипта.

Запуск команды происходит из окружения контейнера узла сети. Команда **screen -S user1_term** выполняет вход в консоль узла user1.

Рассмотрим процесс отправки данных от одного пользователя сети к другому. В данном случае рассмотрим отправку параметров цепочки поставки пользователем PD стороне T. Доступ пользователям сети к отдельным категориям данных предоставляется согласно матрице разрешений. Для обеспечения хранения и извлечения информации в блокчейн-сети для авторизованных одноранговых узлов используется механизм создания коллекций приватных данных, которые позволяют определённому подмножеству организаций на канале поддерживать, фиксировать или запрашивать частные данные без необходимости создавать отдельный канал взаимодействия. Для отправки параметров цепочки поставки реализована функция SendDeliveryParametrs, которая выполняется с помощью следующей команды:

peer chaincode invoke –o orderer.smart-konosament.com:7050 —tls true —cafile $ORDERER_CA -C konosamentchannel -n konosament -c ‘{“function”: “SendDeliveryParametrs”, “Args”: []}’

Метод Invoke вызывается при обращении к любой функции чейнкода. При этом первый элемент массива при вызове Invoke содержит название функции чейнкода, так как через метод Invoke проходит вызов любого метода чейнкода. В этом методе идет работа с состоянием смарт-контрактов [5].

Запрос переданных параметров на стороне T:

peer chaincode query -C konosamentchannel -n konosament -c ‘{“Args”: [“GetPrivateBlockById”, “PersonalDirection_and_TrustedSide”, “DeliveryParametrs”]}’

Как видно, данная команда ссылается на определение коллекции приватных данных только для пользователей PD и T.

Результат выполнения запроса представлен на рисунке 1.

Для проверки работы функции отправки параметров цепочки поставки реализована функция TestSendDeliveryParametrs, которая выполняется с помощью следующей команды:

Таблица 1. Тестирование функций чейнкода морского коносамента

№ п/п	Функция чейнкода	Результат функции
Функция чейнкода	CreateGenesisBlock	Формирование нулевого блока
Функция теста	TestCreateGenesisBlock(t *testing.T)	OK Command-line arguments 0.248 s
Функция чейнкода	SendDeliveryParametrs	Отправка параметров цепочки поставки
Функция теста	TestSendDeliveryParametrs(t *testing.T)	OK Command-line arguments 2.193 s
Функция чейнкода	GenerateKeyPair	Генерация ключевой пары
Функция теста	TestGenerateKeyPair(t *testing.T)	OK Command-line arguments 1.991 s
Функция чейнкода	SendPublicKey	Отправка открытого ключа
Функция теста	TestSendPublicKey(t *testing.T)	OK Command-line arguments 0.089 s
Функция чейнкода	SignTransactionsAndGenerateBlockQ	Подписание транзакций и генерации блока
Функция теста	TestSignTransactionsAndGenerateBlockQ (t *testing.T)	OK Command-line arguments 0.097 s
Функция чейнкода	GenerateUserID	Генерация параметра пользователя сети
Функция теста	TestGenerateUserID(t *testing.T)	OK Command-line arguments 0.083 s
Функция чейнкода	SendGenerAndPublicKey	Отправка сгенерированного параметра и публичного ключа в зашифрованном виде
Функция теста	TestSendGenerAndPublicKey(t *testing.T)	OK Command-line arguments 0.120 s
Функция чейнкода	SendUsersID	Отправка списка идентификаторов пользователей в зашифрованном виде
Функция теста	TestSendUsersID(t *testing.T)	OK Command-line arguments 0.087 s
Функция чейнкода	GenerateListOfSignedDocuments	Формирование списков подписанных документов
Функция теста	TestGenerateListOfSignedDocuments (t *testing.T)	OK Command-line arguments 1.177 s
Функция чейнкода	SendPrivateKey	Отправка закрытого ключа
Функция теста	TestSendPrivateKey(t *testing.T)	OK Command-line arguments 0.175 s
Функция чейнкода	VerifySignedDocuments	Проверка подписанных документов
Функция теста	TestVerifySignedDocuments(t *testing.T)	OK Command-line arguments 21.839 s
Функция чейнкода	ResendSignedDocumentAndGeneratedParametr	Пересылка пользователям зашифрованных подписанных документов и сгенерированных параметров
Функция теста	TestResendSignedDocumentAndGeneratedParametr(t *testing.T)	OK Command-line arguments 0.096 s
Функция чейнкода	SendGeneratedParametrs	Отправка сгенерированных параметров пользователей
Функция теста	TestSendGeneratedParametrs(t *testing.T)	OK Command-line arguments 0.519 s
Функция чейнкода	SendDocumentAndGeneratedParametr	Отправка подписанного документа и сгенерированных параметров пользователей в зашифрованном виде

№ п/п	Функция чейнкода	Результат функции
Функция теста	TestSendDocumentAndGeneratedParametr (t *testing.T)	OK Command-line arguments 0.139 s
Функция чейнкода	ZKP_Proof	Запрос верификации секретного значения s. Посредством выполнения данной функции пользователи отправляют свои секретные ключи, сгенерированные параметры и идентификаторы стороне T в зашифрованном виде, используя доказательство с нулевым разглашением
Функция теста	TestZKP_Proof(t *testing.T)	OK Command-line arguments 0.101 s
Функция чейнкода	ZKP_Verify	Верификация значения секретного s в доказательстве с нулевым разглашением
Функция теста	TestZKP_Verify(t *testing.T)	OK Command-line arguments 0.127 s

```
bash-5.0# go test smart-konosament_test.go -v -run TestSendDeliveryParametrs -marble="$MARBLE"
=== RUN   TestSendDeliveryParametrs
--- PASS: TestSendDeliveryParametrs (2.19s)
PASS
ok      command-line-arguments  2.193s
```

Рис. 2. Тестирование функции отправки параметров цепочки поставки

export MARBLE=$(echo -n "{\"Composition\": \"полный состав\", \"CarrierBookingNumber\": \"Carrier № 1\", \"BillOfLadingNumber\": \"Bill of lading № 1\", \"DeliveryAddress\": \"Some city or street\", \"PurchaseOrder\": \"Purchase № 1\", \"BookingData\": \"Date of the order is 01.06.2021\", \"EquipmentNumber\": \"Equipment № 1\", \"EquipmentType\": \"коробки\", \"OriginCountry\": \"Russia\", \"DestinationCountry\": \"China\", \"Users\": [\"User1\", \"User2\", \"User3\"]}" | base64 | tr -d \\n)

go test smart-konosament_test.go -run SendDeliveryParametrs -marble="$MARBLE"

Мы рассмотрели процесс тестирования функции отправки параметров цепочки поставки SendDeliveryParametrs с помощью функции теста TestSendDeliveryParametrs(t *testing.T). В рамках данной работы такие юнит-тесты для каждой функции кода, рассматриваемого смарт-контракта.

В таблице 1 приведены функции, реализованные в чейнкоде морского коносамента, а также их определения. Для каждой функции чейнкода приведена функция тестирования и результат ее выполнения.

Тесты реализуются посредством вызова команд вида «peer chaincode invoke», в результате выполнения которых проверяется код ошибки. Таким образом функция, реализованная в смарт-контракте вызывается командой вида «peer chaincode invoke», а функция для её тестирования командой вида «go test smart-konosament_test.go –run». Пример тестирования функции «SendDeliveryParametrs» приведен на рисунке 2.

В результате был написан и протестирован смарт-контракт, реализующий договор морского коносамента Smart-konosament. Тесты для проверки работы каждой функции объединены в конфигурационный файл Smart-konosament_Test, взаимодействующий со смарт-коносаментом напрямую с использованием инструментов языка программирования Golang, что в целом положительно сказалось на скорости тестирования чейнкода.

Положительный результат тестирования позволяет сделать вывод об исполняемости смарт-коносамента. Данный смарт-контракт может быть использован для организации процессов документирования морских перевозок в электронном виде на базе технологии распре-

деленного реестра. Также данный контракт может быть взят за основу для дальнейшей разработки других форм документов морской логистики.

В результате тестирования выданы следующие рекомендации:

- аудит информационной безопасности, подразумевающий тестирование смарт-контракта на компьютерные атаки;
- реализация механизма взаимодействия данного чейнкода с другими смарт-контрактами, обеспечивающими функционирование банков и других систем.

ЛИТЕРАТУРА

1. Петрухин, В.А. Методы и средства инженерии программного обеспечения: курс лекций / Петрухин В.А., Лаврищева Е.М.— Москва: Интуит НОУ, 2016.— 467 с.
2. Testing Smart Contracts — Режим доступа: https://yos.io/2020/07/09/testing-smart-contracts/ (дата обращения: 20.08.2021)
3. A simple guide for how to write unit tests for smart contracts — Режим доступа: https://medium.com/upstate-interactive/a-simple-guide-for-how-to-write-unit-tests-for-smart-contracts-8ec4b645f57b (дата обращения: 18.08.2021).
4. Тестирование блокчейн: как обеспечить качество смарт-контрактов в dApp — Режим доступа: https://www.a1qa.ru/blog/testirovanie-blockchain-kak-obespechit-kachestvo-smart-kontraktov-v-dapp/ (дата обращения: 25.08.2021)
5. Разработка и тестирование смарт-контрактов Hyperledger Fabric — Режим доступа: https://habr.com/en/post/426705/ (дата обращения: 25.08.2021)
6. Ицыксон, В.М. Технологии выборочного регрессионного тестирования / В.М. Ицыксон, М.Х. Ахин // Моделирование производственных процессов и развитие информационных систем, Ставрополь, 21–22 марта 2011 года / Даугавпилсский университет, Латвия, Европейский Союз Белорусский государственный университет, Беларусь Днепропетровский университет экономики и права, Украина Московский государственный университет им. М.В. Ломоносова, Россия Санкт-Петербургский государственный политехнический университет Северо-Кавказский государственный технический университет Ставропольский государственный университет Ставропольский государственный аграрный университет.— Ставрополь: Издательство "АГРУС", 2011.— С. 248–251.— EDN TRZBNJ.
7. Гылыжов, Б. Шаблоны тестирования программного обеспечения / Б. Гылыжов, Н.И. Томилова // Инновации. Наука. Образование.— 2021.— № 26.— С. 1688–1699.— EDN XIBNTM.

Щербинина Инна Александровна (shcherbinina@msun.ru), Путилова Софья Евгеньевна .
Журнал «Современная наука: актуальные проблемы теории и практики»

Морской Государственный Университет им. адм. Г.И. Невельского