Верификация контрактов в криптовалюте Ethereum

Введение

Ethereum – платформа для создания онлайн-сервисов на базе технологии blockchain. Сеть Ethereum была запущена 30 июля 2015 года. В сети используется специальная валюта – ether, обеспечивающая функционирующим онлайн-сервисам возможность совершать финансовые сделки.

Участники сети Ethereum называются аккаунтами. Аккаунт – единица, имеющая возможность совершать операции, изменяющие состояние сети. Существует два вида аккаунтов: внешние и внутренние. Внешний аккаунт менее формально справедливо можно назвать «аккаунт-человек», так как все действия этого аккаунта инициируются человеком извне. Внутренний аккаунт можно воспринимать как «аккаунт-контракт» - аккаунт, действия которого полностью определяются его исходным кодом. Так, контрактами (а также смарт-контрактами или умными контрактами) называются аккаунты, действия которых определяются исходным кодом. Для наглядности виды аккаунтов представлены на рис. 1.



Рис. 1. Виды аккаунтов

Действия любых аккаунтов могут изменять состояние сети. Состояние сети – это совокупность состояний всех аккаунтов. Состояние аккаунта-человека полностью определяется его балансом (в специальной валюте ether). Состояние контракта определяется его балансом и данными в его хранилище.

Состояние сети изменяется посредством транзакций (иначе говоря, транзакции – это действия аккаунтов). Транзакция инициируется некоторым аккаунтом и адресуется другому аккаунту. Если транзакция адресована аккаунту-человеку, единственное, что она может осуществить – перевод денег (ether) от инициатора к получателю. Если же транзакция направлена контракту, то она обязательно вызывает исполнение кода контракта. Что касается перевода денег в этом случае, он может быть направлен от инициатора к адресату, а может вовсе не осуществляться.

Так как состояния сети и переходы из одного состояния в другое были только что определены, сеть Ethereum можно рассматривать как конечный автомат, переходы в котором – это транзакции.

О жизненном цикле транзакции стоит сказать более подробно. После того, как транзакция была отправлена в сеть инициатором, она включается в блок (процесс сбора транзакций в блоки называется mining). После того, как блок собран, транзакции из него исполняются. За исполнение транзакции взимается комиссионная плата, называемая gas в сети Ethereum. Оплачивает комиссионную плату инициатор транзакции.

Действия контракта в сети определяются его исходным кодом. Исходный код контрактов пишется на специальном javascript-подобном языке высокого уровня – Solidity. На рис. 2 представлен код одного небольшого контракта на Solidity.

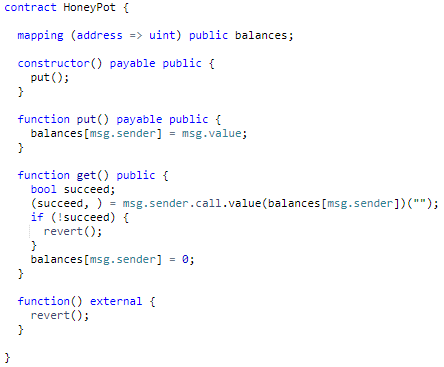


Рис. 2. Пример кода контракта на Solidity

Как видно, в коде определено несколько функций. Адресованная контракту транзакция – это вызов одной высокоуровневой функции.

Код на языке Solidity переводится в код инструкций Ethereum Virtual Machine (EVM). EVM представляет собой стековую машину, которая оперирует со стеком, памятью и хранилищем. Содержимое стека и памяти теряется после завершения транзакции, тогда как хранилище перманентно. EVM включает в себя много инструкций, подробное обсуждение которых здесь не имеет смысла. Полная документация EVM представлена в Ethereum Yellow Paper.

Из сказанного выше ясно, что контракт – это программа. В контрактах, как и в любых других программах, есть уязвимости. На сайте <https://dasp.co> представлены 10 наиболее известных видов уязвимостей. Уязвимости в контрактах привели сеть Ethereum к нескольким инцидентам с огромными финансовыми потерями. Так, в июне 2016 года произошла известная атака TheDAO, в результате которой было украдено $74 млн. За 2017 год злоумышленниками было украдено $300 млн. Такая картина потерь говорит о необходимости тщательной проверки кода контрактов на уязвимости. Анализ исходного кода человеком «вручную» может применяться в исключительных случаях, но не годится для производственных задач. Поэтому нужны инструменты, позволяющие автоматически проверять исходный код контрактов на уязвимости. Для этой цели существует несколько инструментов, например, Mythril, Oyente, Manticore, Securify. В результате анализа кода все эти инструменты формируют список рекомендаций для разработчика, на что стоит обратить внимание и что, возможно, стоит исправить. Ни один из существующих анализаторов не проводит формальную верификацию кода на отсутствие уязвимостей и не строит конкретные сценарии атаки. Цель этой работы – создание инструмента для автоматического поиска уязвимостей в кодах контрактов, позволяющего находить возможные сценарии атак.

Подход к решению

Формальная верификация кода – это строгое математическое доказательство его соответствия спецификациям. Спецификации иначе можно назвать формальными требованиями. В случае, если код не соответствует спецификациям, нужно привести контрпример. Существует несколько подходов к формальной верификации кода. Наиболее широко применимы проверка моделей, символьное исполнение и автоматическое доказательство теорем. В этой работе осуществляется формальная верификация кода контракта методом проверки модели. Для проверки модели используется система Spin с собственным встроенным языком программирования Promela.

Верификации подвергается низкоуровневый код контракта (в инструкциях стековой машины EVM), так как он, в отличие от высокоуровневого кода на Solidity, открыт у всех контрактов. Низкоуровневый исходный код транслируется в код на языке Promela. Также на языке Promela сформулированы требования, эквивалентные отсутствию в коде нескольких видов уязвимостей. Затем система Spin проверяет, соответствует ли код на языке Promela сформулированным требованиям. Если код не соответствует требованиям (т.е. он подвержен уязвимости), Spin не только сообщает этот факт, но и строит контрпример, на основе которого элементарно строится сценарий атаки.

Здесь имеет смысл кратко рассказать о принципе работы системы Spin. По полученному для анализа коду на языке Promela строится автомат Бюхи, каждое состояние которого соответствует одному набору значений всех глобальных переменных, а операции над переменными соответствуют переходам в автомате. Кроме того, строится автомат Бюхи по отрицанию формальных требований к коду. Затем строится пересечение языков, порождаемых первым и вторым автоматами. Если пересечение оказывается пусто, значит, код требованиям соответствует, иначе – не соответствует.

Далее речь пойдет о принципе трансляции кода контракта на язык Promela. Однако этот принцип представляет из себя не что иное, как реализацию на языке Promela следующей схемы. Будем называть жертвой контракт, который проверяется на уязвимости (чисто условный термин, вовсе не означающий, что контракт уязвим). Предполагаемый злоумышленник – это тоже контракт, который отправляет жертве несколько транзакций таким образом, чтобы нанести ущерб (например, вывести все деньги или вовсе разрушить жертву). Как было сказано во введении, транзакция – это вызов одной высокоуровневой функции контракта (в нашем случае жертвы). Так, если злоумышленник поставил цель нанести ущерб жертве, ему нужно построить последовательность вызовов функций жертвы, которая нанесет ей вред.

Трансляция кода контракта проецирует приведенную схему на понятия системы Spin. Каждая инструкция виртуальной машины EVM реализована на Promela. Так, инструкции, из которых состоит код контракта, при трансляции заменяются на свои реализации. Язык Promela позволяет определить множество типов процессов (proctype), исходный код каждого из которых задается отдельно. Процесс может принимать на вход аргументы, но не может вернуть значение. Есть процесс, отличающийся от всех остальных – это процесс с именем init, аналог функции main, не принимающий на вход аргументов. Все остальные процессы запускаются асинхронно по вызову run <имя процесса> из кода на Promela. Код каждой функции контракта выделяется в отдельный тип процесса. Роль злоумышленника играет процесс init.

В начале работы процесс init назначает жертве и злоумышленнику одинаковые ненулевые балансы. Promela позволяет задать несколько действий, из которых будет недетерминированно выбрано и исполнено одно. Также есть возможность задать цикл, который может прекращаться при наступлении некоторого условия либо быть вечным. Пользуясь этими двумя возможностями, в процессе init (злоумышленник) задан цикл, прекращающийся либо при нулевом балансе на счету жертвы, либо при разрушении жертвы, либо после вызова функций жертвы определенное число раз. В теле цикла происходит недетерминированный выбор и вызов одной из функций контракта. Очевидно, в такую модель вписывается любая последовательность вызовов функций.

Некоторую сложность представляет тот факт, что вызовы функций в Spin асинхронны, тогда как в Ethereum они всегда синхронны. Для синхронизации введена специальная глобальная переменная run\_verifier, истинность которой совпадает с возможностью злоумышленника (init) в данный момент вызывать функцию жертвы. Изначально run\_verifier = true, перед тем, как init запускает какой-то из процессов, ее значение устанавливается false, когда же процесс завершает работу, переменной run\_verifier снова присваивается значение true.

Возможен и другой случай, когда процесс (не init) присваивает переменной run\_verifier значение true. Это случай, когда жертва отправляет транзакцию злоумышленнику. Такой вызов синхронен, как и все вызовы в Ethereum. Чтобы синхронизировать исполнение функций с учетом возможности отправки транзакции злоумышленнику, используется стек вызовов. Каждый элемент этого стека содержит некоторые служебные данные (какие именно – сейчас неважно). Если жертва отправляет транзакцию злоумышленнику, она сначала запоминает текущую высоту стека вызовов, потом присваивает run\_verifier = true и кладет на стек вызовов элемент, соответствующий этому вызову транзакции. Тот экземпляр процесса жертвы, который только что отправил злоумышленнику транзакцию, приостанавливает работу до момента, когда высота стека вызовов снова станет равной запомненному значению.

Корректная обработка транзакций как от злоумышленника к жертве, так и в обратную сторону, чрезвычайно важна для обнаружения реентерабельности – уязвимости, позволяющей вывести все средства с баланса контракта, которая была использована при известной атаке TheDAO. Инструмент, созданный в этой работе, позволяет обнаруживать реентерабельность. Также созданный инструмент позволяет обнаруживать незащищенный деструктор (функция, после вызова которой контракт перестает существовать) и недостаточно защищенные функции перевода средств.

Серьезный недостаток инструмента – реализация символьного исполнения лишь на самом элементарном уровне, из-за чего он не способен находить многие сценарии атак, в которых ключевую роль играют правильно подобранные аргументы для вызова функций жертвы. Скорее всего, продуманное и правильное использование символьного исполнения позволит инструменту находить гораздо более сложные сценарии, зависящие от аргументов, чем он сейчас способен найти. Однако, реализация символьного исполнения на серьезном уровне – очень трудоемкая задача, браться за которую имеет смысл только в том случае, если есть заказчик, для которого инструмент представляет реальный практический интерес.

Заключение

В рамках этой работы создан инструмент для автоматического поиска уязвимостей в кодах смарт-контрактов Ethereum, анализирующий исходные коды на уровне инструкций EVM, основанный на выполняемой системой Spin формальной верификации путем проверки модели. Инструмент был проверен на нескольких тестовых примерах контрактов и показал корректные результаты.