Лабораторная работа
Обзор уязвимостей смарт-контрактов в блокчейне криптовалюты Ethereum

1 Описание лабораторного стенда

Лабораторный стенд состоит из компьютера, на котором установлены средства виртуализации операционных систем, позволяющие работать с виртуальной машиной в конфигурации:

а) операционная сисетма Ubuntu 20.04 (user/user);

б) программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации Docker (для запуска Remix IDE - веб-интерфейса для работы с контрактами). Установлены в среде виртуальной машины;

в) статический анализатор кода смарт-контрактов Slither.

2 Общий сценарий и результат выполнения работы

Выполнение работы включает в себя предварительный этап и три учебных этапа. Предварительный этап заключается в подготовке виртуальной машины. Эта подготовка включает в себя установку на виртуальной машине программных продуктов, указанных в описании лабораторного стенда. Хотя ниже приводится описание предварительного этапа, предполагается, что обучающиеся начинают выполнение лабораторной работы с п.4 (первый учебный этап), используя настроенный и готовый лабораторный стенд. Первый учебный этап посвящен обзору атаки Reentrancy (атака повторного входа), второй - атаки DoS (Denial of Service), третий - в применении статического анализатора кода смарт-контрактов Slither.

Целью лабораторной работы является выявление и проведение атак Reentrancy и DoS для получения навыков работы с нетривиальными смарт-контрактами (размещение смарт-контрактов в сети блокчейна Ethereum, взаимодействие с контрактами через графический интерфейс, анализ смарт-контрактов на предмет уязвимостей).

3 Предварительный этап

Установка программного обеспечения производится запуском команд:

Устанавливаем служебную программу командной строки, позволяющуя взаимодействовать с множеством различных серверов по множеству различных протоколов с синтаксисом URL:

- *sudo apt install curl*

Устанавливаем Docker - платформа контейнеризации с открытым исходным кодом, с помощью которой можно автоматизировать создание приложений, их доставку и управление:

- *sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose*

Загружаем контейнер, в котором запакован Remix IDE:

- *sudo docker pull remixproject/remix-ide:remix_ live*

- *sudo apt-get install python3-pip*

- *pip install slither-analyzer*

4 Первый учебный этап. Уязвимость Reentrancy

Атака с повторным входом (Reentrancy атака) может быть проведена если возможно создать цикл между целевым контрактом и контрактом злоумышленника. Цикл между контрактами может возникнуть если вызов внешней функции в целевом контракте происходит до изменения его внутреннего состояния. Атаку повторного входа используют в различных целях, в частности для постепенного вывода средств из целевого контракта.

Размещение смарт-контрактов

1. Открыть терминал комбинацией клавиш *Ctr + Alt + T*

2. Ввести команду

   sudo docker run -p 8080:80 remixproject/remix-ide:remix-live

   для запуска docker контейнера, в котором упакован оффлайн компилятор Remix (в данный момент предложенный способ является самым простым для использования **Remix IDE**).

3. Открыть Mozilla Firefox, ввести в адресную строку localhost:8080 (появится веб-интерфейс Remix IDE для работы со смарт-контрактами).

4. Создать новый файл контракта. Для этого нужно нажать на **иконку файла**, находящуюся над папкой contracts. В появившейся строке ввести имя контракта Reentrancy.sol. Нажать **Enter**.

5. Скопировать код смарт-контракта из файла Reentrancy.sol, который находится на рабочем столе (Desktop) вирутальной машины и вставить его в редактор remix в созданный файл.

6. Перейти во вкладку **Solidity Compiler**.

Рис. 1: Вкладка **Solidity Compiler**.

   Нажать на кнопку **Compile Reentrancy.sol**. Дождаться пока скомпилируется контракт.

7. Перейти во вкладку **Deploy & Run Transactions**.

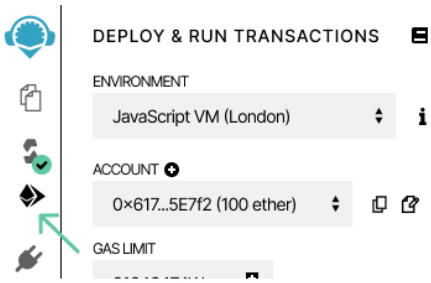

Рис. 2: Вкладка **Deploy & Run Transactions**.

   Нажать на кнопку **Deploy**. В меню *Deployed Contracts* (снизу) должна была появиться соответствующая строка (отвечающая за управление контрактом).

8. Перейти в первоначальную вкладку **File Explorer** (Рис. 3) и создать новый файл *ReentrancyAttack.sol* (по аналогии с пунктом 4).

9. Скопировать код смарт-контракта из файла *ReentrancyAttack.sol*, который находится на рабочем столе (Desktop) и вставить его в редактор remix в созданный файл.

10. Перейти во вкладку **Solidity Compiler**. Нажать на кнопку **Compile *ReentrancyAttack*.sol**. Дождаться пока скомпилируется контракт.

11. Перейти во вкладку **Deploy & Run Transactions**. Скопировать адрес задеплоенного контракта REENTRANCYAUCTION... (Рис. 4), нажав на **иконку файла**, находящегося в строке, отвечающей за управление контрактом.

12. В поле *address_ auction*, находящееся рядом с оранжевой кнопкой Deploy, вставить скопированный адрес контракта REENTRANCYAUCTION... и нажать на кнопку **Deploy**. В меню *Deployed Contracts* должна была появиться соответствующая строка (отвечающая за управление контрактом).

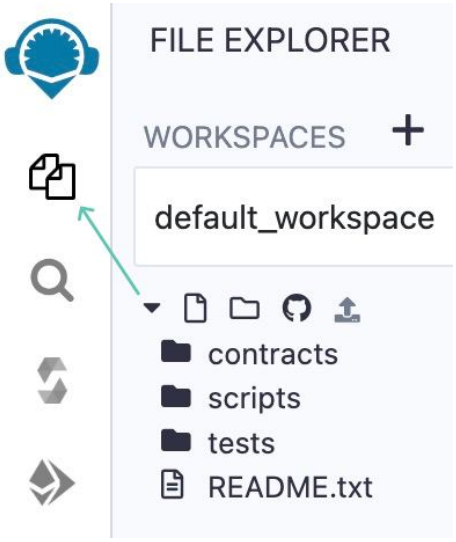

Рис. 3: **Deploy & Run Transactions**

.

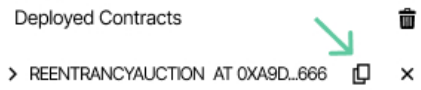

Рис. 4: Копировать адрес размещенного контракта.

.

Обзор смарт-контракта *ReentrancyAuction*

Приведенный ниже код — это уязвимый смарт-контракт, на который будет производиться атака. Он работает как кошелек для пользователей:

- пользователи могут внести любое количество эфира в этот контракт. Для этого нужно вызвать функцию *bid*, передав в нее значение количества эфира;

- пользователи могут посмотреть свой баланс, вызвав функцию *currentBalance*;

- пользователи могут узнать баланс средств любого пользователя, хранящихся в смарт-контракте *ReentrancyAuction*, передав в функцию *bidders* адрес [1] соответствующего пользователя;

- пользователи могут вывести положенный ими эфир, вызвав функцию *refund*. Пользователь, вызывающий эту функцию может вывести сумму эфира, соответсвующую его адресу на балансе смарт-контракта. Первая строка функции выполняет работу по проверке баланса вызывающего абонента. Если у пользователя на балансе хранятся средства, вызывающий пользователь получит положенное ему количество эфира. Эфир отправляется с помощью низкоуровневой функции вызова и затем внутренний балансовый счет контракта *ReentrancyAuction* обновляется. Блокчейн также автоматически обновится баланс контракта, потому что эфир, хранящийся на этом счете смарт-контракта уменьшится из-за вывода средств.

Эта функция имеет уязвимость Reentrancy (повторного входа), которую будет использоваться в атаке. Позже будет объясняться как работает атака.

Обзор смарт-контракта *ReentrancyAttack*

Чтобы запустить повторную атаку на контракт жертвы, злоумышленнику необходимо развернуть смарт-контракт для атаки. Ниже приведен код атакующего контракта

Наиболее важными функциями этого контракта являются *attack* и *receive*. Мы объясним, как этот контракт может быть использован для кражи всех средсв из контракта жертвы. После развертывания контракта злоумышленник вызывает функцию *proxyBid* и отправляет по крайней мере один эфир в этот контракт. Эта функция внесет один эфир в контракт жертвы (*ReentrancyAuction*), вызвав его *bid* функцию. После того как будут внесены средства в контракт *ReentrancyAuction*, контракт злоумышленника (*ReentrancyAttack*) выводит один эфир из контракта жертвы, вызвав функцию attack. Таким образом, злоумышленники могут создать последовательность рекурсивных вызовов для вывода средств из *ReentrancyAuction*.

В результате получилась следующая последовательность действий (изображена на Рис. 7):

[1] **Адрес в блокчейне** – уникальная неповторимая комбинация из цифр и букв латинского алфавита, которая представляет собой хеш открытого ключа.

```
contract ReentrancyAuction {
    mapping (address => uint) public bidders;

    function bid() external payable {
        bidders [msg.sender] += msg.value;
    }

    function refund() external {
        uint refundAmount = bidders[msg.sender];
        if (refundAmount > 0) {
            (bool success,) = msg.sender.call{value: refundAmount} ("");
            require (success, "failed!");
            bidders[msg.sender] = 0;
        }
    }

    function currentBalance() external view returns(uint) {
        return address (this).balance;
    }
}
```

Рис. 5: Реализация контракта *ReentrancyAuction.sol*

.

```
contract ReentrancyAttack {
    uint constant BID_AMOUNT = 1 ether;
    ReentrancyAuction auction;

    constructor (address _auction) {
        auction = ReentrancyAuction (_auction);
    }

    function proxyBid () external payable {
        require (msg.value == BID_AMOUNT, "incorrect");
        auction.bid{value: msg.value} ();
    }

    function attack () external {
        auction.refund ();
    }

    receive () external payable {
        if (auction. currentBalanace () >= BID_AMOUNT{
            auction. refund ();
        }
    }

    function currentBalanse() external view returns (uint) {
        return address (this).balance;
    }
}
```

Рис. 6: Реализация контракта *ReentrancyAttack.sol*

.

1. Смарт-контракт *ReentrancyAttack* запрашивает вывода средств (вызывая функцию attack()).
2. Смарт-контракт *ReentrancyAuction* проверяет баланс пользователя, убеждается, что баланс больше нуля (так как предварительно мы вызвали функцию proxyBid контракта *ReentrancyAttack*).
3. Смарт-контракт *ReentrancyAuction* отправляет средства контракту *ReentrancyAttack*.
4. После того, как средства будут получены контрактом *ReentrancyAttack* автоматически будет вызвана функция recieve. Это снова вызовет функцию refund контракта *ReentrancyAuction*. Процесс будет повторяться до тех пор, пока баланс контракта *ReentrancyAuction* не опустится ниже 1 эфира.

Демонстрация уязвимости Reentrancy

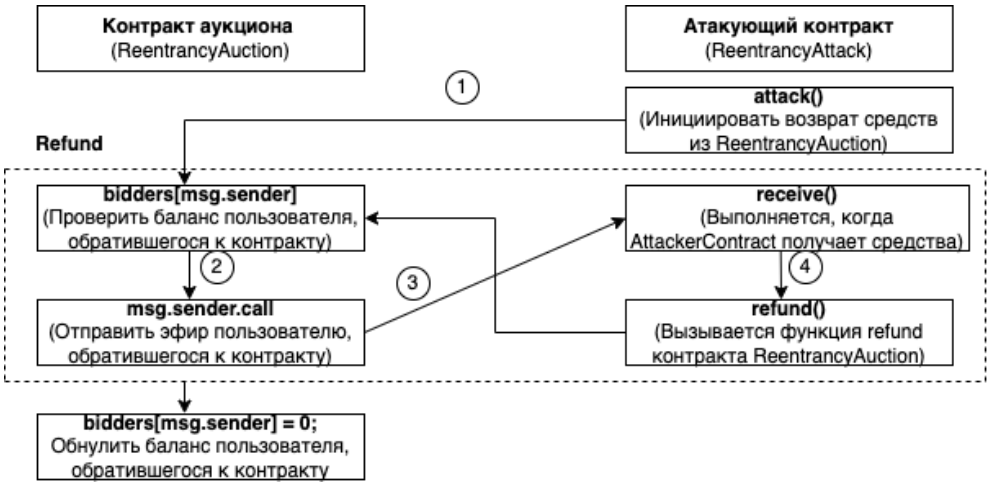

Рис. 7: Описание атаки Reentrancy

.

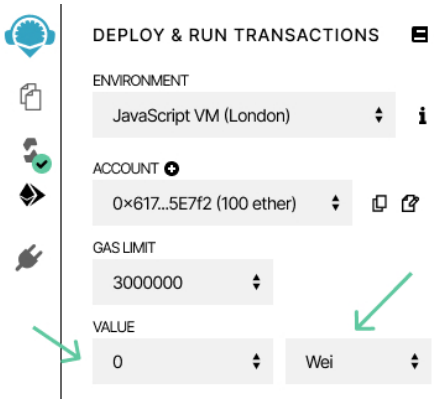

Рис. 8: Изменение значений передаваемых в транзакции

.

1. В графе *VALUE* поставить значение 10. В рядом стоящей графе сменить параметр *Wei* на *Ether* (Рис. 8)

2. В меню *Deployed Contracts* в поле REENTRANCYAUCTION... нажать на стрелочку, после чего будет развернут интерфейс для управления контрактом.

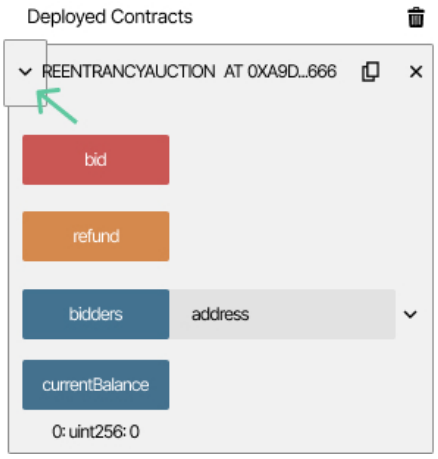

Рис. 9: Раздел *Deployed Contracts*

.

3. Нажать на кнопку **currentBalance**. Снизу появится результат выполнения функции (0: uint256: 0). Это означает, что баланс *ReentrancyAuction* контракта равен нулю.

4. Нажать на кнопку **bid**. Далее нажать на кнопку currentBalance. Снизу появится результат выполнения функции (0: uint256: 100...000). Это означает, что баланс *ReentrancyAuction* контракта изменился.

5. В графе *Account* выбрать второй сверху адрес. В графе *VALUE* поставить значение 1. Убедиться, что в рядом стоящей графе стоит параметр *Ether*.

6. В меню *Deployed Contracts* в поле REENTRANCYATTACK... нажать на стрелочку, после чего будет развернут интерфейс для управления контрактом. Нажать на кнопку **proxyBid**.

7. Проверить баланс контракта *ReentrancyAuction*, нажав на кнопку **currentBalance** в меню управления контрактом *ReentrancyAuction*. Убедиться, что баланс увеличился.

8. Проверить баланс контракта *ReentrancyAttack* нажав на кнопку **currentBalance** в меню управления контрактом *ReentrancyAttack*. Убедиться, что баланс контракта равен нулю.

9. В меню управления контрактом *ReentrancyAttack* нажать на кнопку **attack**. Дождаться выполнения транзакции.

10. Проверить баланс контракта *ReentrancyAuction*. Убедиться, что на балансе больше не осталось средств.

11. Проверить баланс контракта *ReentrancyAttack*. Убедиться, что на баланс увеличился на сумму, которая хранилась на балансе контракта *ReentrancyAuction*.

5 Второй учебный этап. Уязвимость DoS

Отказ в обслуживании (отсюда и название DoS) ограничивает законных пользователей в использовании смарт-контрактов постоянно или в течение определенного периода времени.

Размещение контрактов

Размещение контрактов *DosAuction* и *DosAttack* происходит аналогично первому учебному этапе (п.4).

1. Создать новый файл контракта. Для этого нужно нажать на **иконку файла**, находящуюся над папкой contracts. В появившейся строке ввести имя контракта *Dos.sol*. Нажать **Enter**.

2. Скопировать код смарт-контракта из файла *Dos.sol*, который находится на рабочем столе (Desktop) и вставить его в редактор remix в созданный файл.

Задание.

- Разместить в блокчейне контракт *Dos.sol* (по аналогии с п. 4-7 первого учебного этапа)
- Разместить в блокчейне контракт *DosAttack.sol* (по аналогии с п. 8-12 первого учебного этапа)

Обзор смарт-контракта *DosAuction* и *DosAttack*

Контракты *DosAuction* и *DosAttack* принципиально не отличаются от контрактов *ReentrancyAuction* и *ReentrancyAttack*.

После развертывания контрактов злоумышленник вызывает функцию proxyBid контракта *DosAttack* и отправляет по крайней мере один эфир в контракт *DosAuction*.

В ходе атаки получается следующая последовательность действий (также изображенная на Рис. 10):

1. Смарт-контракт *DosAttack* запрашивает вывода средств (вызывая функцию refund контракта *DosAuction*).

2. Смарт-контракт *DosAuction* начинает возвращать эфир все, кто сделали ставки.

3. Очередь доходит до контракта *DosAttack*.и контракт *DosAuction* отправляет средства контракту *DosAttack*.

4. После того, как средства будут получены контрактом *DosAttack*, автоматически будет вызвана функция recive. Это действие инициализирует вызов бесконечного цикла, который далее не позволит нам взаимодействовать с контрактом *DosAuction*.

Демонстрация уязвимости DoS

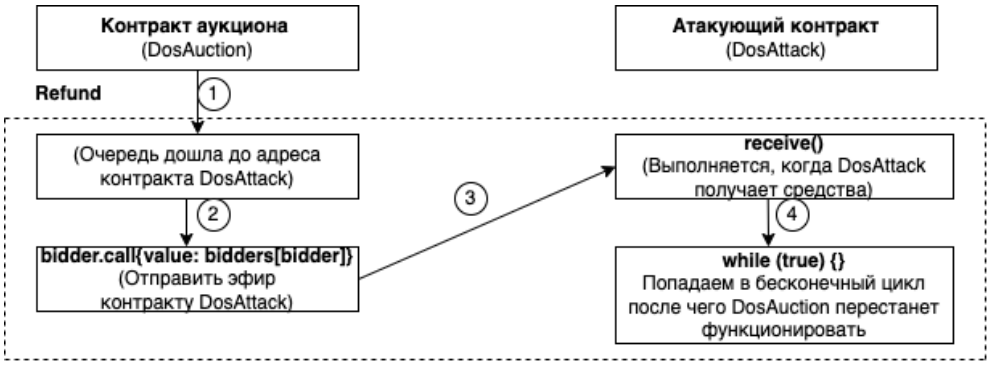

Рис. 10: Описание атаки DoS

.

1. В графе Account выбрать третий сверху адрес. В графе Value поставить значение 1. Убедиться, что в рядом стоящей графе стоит параметр Ether.
2. В меню Deployed Contracts в поле *DosAuction* нажать на стрелочку, после чего будет развернут интерфейс для управления контрактом.
3. Нажать на кнопку doBid контракта *DosAttack* - делаем ставку в размере 1 эфира на контракт *DosAuction* (с контракта злоумышленника).
4. В графе Value поставить значение 1.
5. Нажать на кнопку bid контракта *DosAuction* - делаем ставку в размере 1 эфира на контракт *DosAuction* (от имени пользователя).
6. Нажать на кнопку refund контракта *DosAuction* - инициализируем возврат средств всем пользователям, сделавших ставки на контракт *DosAuction*.

   После этого действия в консоли появится уведомление о том, что транзакция выполняется (однако она уже не завершится). Браузер Mozilla Firefox предложит закрыть текущую страницу, так как она тормозит скорость браузера (это означает, что атака прошла успешно)
7. Закрыть браузер Mozilla Firefox.

6 Третий учебный этап. Применение статического анализатора кода смарт-контрактов Slither

Slither - статический анализатор кода, написанный на python. Он умеет следить за переменными, вызовами, и детектирует некоторые уязвимости (по адресу *https://github.com/trailofbits/slither#detectors* можно ознакомиться со списком уязвимостей), у каждой из них есть ссылка с описанием. Slither может работать, как модуль python и предоставлять программисту интерфейс, для аудита.

Демонстрация работы Sliter

1. Открыть терминал комбинацией клавиш *Ctr + Alt + T*
2. Провести автоматический аудит контракта *ReentrancyAuction* командой /home/user/.local/bin/slither /home/user/Desktop/Reentrancy.sol

   После чего в консоли будет выведена информация о результате аудита контракта с Reentrancy уязвимостью:

   - Указывается на тип уязвимости, обнаруженной в контракте (Reentrancy)

   - Указывается ссылка на описание обнаруженной уязвимости (*Reference: https://github.com/crytic/slither...*)

Задание. Проанализировать контракт DoS.sol средством Slither. Какой тип уязвимости анализатор обнаруживает в данном контракте? Почему?

Приложение.

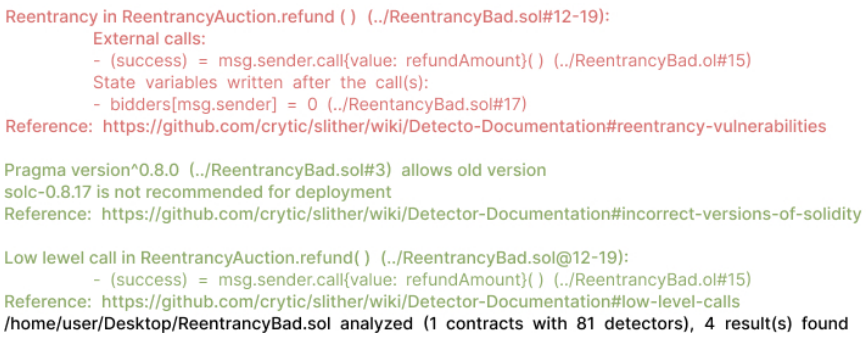

Рис. 11: Результат анализа контракта Reentrancy анализатором **Slither**

.

В данном приложении приводятся некоторые уязвимости, которые может обнаруживать статический анализатор кода смарт-контрактов Slither. Описывается их номер в таблице уязвимостей (*https://github.com/trailofbits/slither#detectors*), название детекотора и описывается, что обнаруживает данный детектор

ID	**Детектор**	**Что обнаруживает**
11	suicidal	Функции, позволяющие любому пользователю уничтожить контракт
21	reentrancy-no-eth	Уязвимости повторного входа (кража эфира)
41	tx-origin	Опасное использование tx.origin
45	unused-return	Неиспользуемые возвращаемые значения
59	timestamp	Опасное использование block.timestamp
80	external-function	Публичная функция, которая может быть объявлена внешней