Резюме

В этом отчёте, мы изучаем безопасность смарт контрактов проекта Cubo Lodge Club. В нашем анализе мы изучали код смарт контрактов , версия последнего коммита `aa40f0f` 9 декабря 2017 года

Методика анализа

Код контракта просматривается вручную на наличие известных уязвимостей, ошибок в логике, соответствие WhitePaper. Также были использованы инструменты автоматического анализа кода. Все найденные инструментами ошибки были протестированы вручную, в результате чего, либо добавлены в отчет , либо отклонены

Классификация уязвимостей

КРИТИЧНЫЕ - возможность кражи эфира/токенов или их блокировки без возможности восстановления доступа или иной потери эфира/токенов.

СЕРЬЕЗНЫЕ - возможность нарушений работы контракта, при которых для восстановления его корректной работы необходима модификация состояния контракта вручную или его полная замена.

ПРЕДУПРЕЖДЕНИЯ - возможность нарушения запланированной логики контракта или не соответствие обьявленной логики в WhitePaper

ЗАМЕЧАНИЯ - все остальные замечания.

Оглавление

Автоматический анализ

Oyente

Timestamp Dependency

CommonCrowdsale : line 357

Timestamp Dependency : True

GOTokenCrowdsale : line 357

Timestamp Dependency:True

Securify

Transaction Reordering

Transactions May Affect Ether Receiver

Matched lines: L.501

Transactions May Affects Ether Amount

Matched lines: L.501

Все найденные уязвимости были проверены вручную и те из них, которые ведут к багам, отражены в отчёте.

Ручной анализ

Критичные ошибки

CommonCrowdsaele.sol, line 499 :

function createTokens()

При вызове payable функции createTokens проверяется только не превышен ли уже хардкап invested < hardcap , но не проверяется будет ли превышен хардкап после текущего транзакции (перевода) . Например до хардкапа не достаёт 10 эфиров, инвестор переводит 20 эфиров и транзакция пройдет все проверки, несмотря на то что хардкап будет превышен. Аналогично и с возможностью превысить личный лимит investedInWei >= maxInvestedLimit . Рекомендуется проверять этот факт внутри функции и в случае если инвестор отправил большую, чем нужно - возвращать сдачу.

Серьезные

Нежелательные циклы

CommonCrowdsaele.sol, line 431 :

function payExtraTokens(**uint** count)

Циклы не желательны в солидити, там, где возможно, лучше их избегать. В данном случае, мы рекомендуем переделать эту функцию, таким образом, чтобы инвестор сам инициировал

CommonCrowdsaele.sol, line 412 :

function end()

В данном случае можно завести переменную “foo” и добавить в функцию

```
uint256 foo;
……..
function addMilestone(uint periodInDays, uint discount) public onlyOwner {
 milestones.push(Milestone(periodInDays, discount));
 foo+= periodInDays;
}
```

делать foo+=periodInDays

и тогда в функция end() примет вид

```
function end() public constant returns(uint) {
 uint last = start+foo;
 return last;
}
```

Версия компилятора задана не чётко

Рекомендуется использовать чётко заданную версию компилятора, т.к более поздние версии компилятора могут воспринимать некоторые языковые конструкции по другому

```
pragma solidity ^0.4.17; // плохо: компилируется с версиями  0.4.17 и выше
pragma solidity 0.4.17; // хорошо : компилируется только с версией 0.4.17
```

Инвестор может превысить лимит инвестиций

CommonCrowdsale.sol, line 486

```
function calculateAndTransferTokens(address to, uint investedInWei) internal {
 invested = invested.add(msg.value);
 uint tokens =  investedInWei.mul(price.mul(PERCENT_RATE)).div(PERCENT_RATE.sub(getDiscount())).div(1 ether);
 mint(to, tokens);
 if(investedInWei >= maxInvestedLimit) token.lock(to);
}
```

В данной вариации ограничивается только единоразовый размер инвестиций. Предположим если maxInvestedLimit = 1000 то я могу инвестировать 10 раз по 999 и не быть заблокирован.
Чтобы заморозить токены инветсора, который превысил лимит, необходимо хранить mapping balances(address => uint) и тогда функция calculateAndTransferTokens будет выглядеть:

```
function calculateAndTransferTokens(address to, uint investedInWei) internal {
 invested = invested.add(msg.value);
 uint tokens = investedInWei.mul(price.mul(PERCENT_RATE)).div(PERCENT_RATE.sub(getDiscount())).div(1 ether);
 mint(to, tokens);
 if(investedInWei >= maxInvestedLimit) token.lock(to);
}
```

Предупреждения

ShortAdressAttack

BasicToken.sol, line 70 :

Функции transfer рекомендуется закрыть от ShortAddressAttack
(http://vessenes.com/the-erc20-short-address-attack-explained/)

Избыточная fallback функция

StantardToken.sol, line 171 :

```
function () public payable {
 revert();
}
```

Эта функция является избыточной, до версии Solidity 0.4.0 это делалось в ручную:
function () payable { throw ; }
но после версии Solidity 0.4.0 контракты без payable функций автоматически отклоняет платежи.

Unchecked math

CommonCrowdsale.sol, line 472 :

```
function getDiscount() public constant returns(uint) {
prevTimeLimit += milestone.periodInDays * 1 days;
```

Рекомендуется для математических операциях, которые могут привести к

переполнению, использовать библиотеку safeMath

Замечания

Константные функции

CommonCrowdsale.sol, line 472 :

```
function tokenHoldersCount() public constant returns(uint) {}
```

Функция объявлена как constant. В текущей версии предпочтительно использовать view вместо constant для функций, так как в дальнейших версиях solidity constant не будет. Тоже самое применимо к milestonesCount(), end() , getDiscount()

Излишнее присваивание

CommonCrowdsale.sol, line 361 :

```
function tokenHoldersCount() public constant returns(uint) {
 uint length = tokenHolders.length;
 return length;
}
```

Излишнее присвоение uint length = tokenHolders.length , причём в дальнейших функциях используется другой подход

```
function milestonesCount() public constant returns(uint) {
 return milestones.length;
}
```

Излишнее присваивание

CommonCrowdsale.sol, line 306

Переменные (hardcap, price,start,wallet e.t.c) объявляются дважды, лучше присваивать значения только в контракте **GOTokenCrowdsale,** а в контракте **CommonCrowdsale** только объявить, не присваивая значений.

Излишнее действие

CommonCrowdsale.sol, line 429

```
function payExtraTokens(uint count)
token.mint(this, targetValue);
token.transfer(tokenHolder, targetValue)
```

Можно сразу вызвать token.mint(tokenHolder, targetValue);

Рекомендации

Добавить софткап и возможность возврата инвестиций

Нет софткапа и как следствие, нет возможности возврата инвестиций при не достижении минимального порога сбора. Рекомендуется добавить возможность возврата инвестиций.