Исходные данные:

Адрес Смарт-контракта в сети Ethereum: 0x1245ef80F4d9e02ED9425375e8F649B9221b31D8

Исходный код доступен на сайте etherscan.io по адресу:
https://etherscan.io/address/0x1245ef80f4d9e02ed9425375e8f649b9221b31d8#code

Токен соответствует спецификации ERC20

Страница токена на сайте etherscan.io:
https://etherscan.io/token/0x1245ef80f4d9e02ed9425375e8f649b9221b31d8
Токен верифицирован командой etherscan.io.

Сайт проекта: https://arbitragect.com/ru/index.html
Reddit: https://www.reddit.com/r/arbitrageCT/
Facebook: https://www.facebook.com/groups/arbitragect/
Twitter: https://twitter.com/arbitrage_ct
Bitcointalk: https://bitcointalk.org/index.php?topic=2242346
Telegram: https://t.me/ArbitrageCT_en
White paper: https://arbitragect.com/arbitragect_wp_en.pdf

Текущее состояние контракта:
ICO завершено.
Владельцу присвоен адрес: 0x0000000000000000000000000000000000000000
Hammer-у присвоен адрес: 0x0000000000000000000000000000000000000000

Выпущено 152,029,112 токенов.
Токен имеет точность до 8-ми знаков после запятой.

Анализ уязвимостей

1. Анализ исходного кода

1.1. Проверка случаев использования небезопасной математики

В коде смарт-контракта обнаружено 11 случаев использования небезопасной математики. Требуется анализ каждого случая на возможность возникновения случаев переполнения переменных.

В функции «transfer» обнаружены 2 случая использования.

```
153  function transfer(address _to, uint _value) returns (bool) {
154      if (balances[msg.sender] >= _value) {
155          balances[msg.sender] -= _value;
156          balances[_to]       += _value;
157          Transfer(msg.sender, _to, _value);
158          return true;
159      }
160      return false;
161  }
```

Строка 155. Операция «balances[msg.sender] -= _value» безопасна, так как в строке 154 выполняется проверка того, что значение «balances[msg.sender]» больше или равно значения «_value».

Строка 156. Проверка на переполнение переменной «balances[_to]» не производится. Однако, исходя из текущего состояния контракта и логики его работы можно установить, что баланс любого пользователя не может превзойти суммарный объем выпущенных токенов. Т.е. числа «152029112000000000». Таким образом переполнение переменной «balances[_to]» в строке 156 невозможно.

В функции «transferFrom» обнаружены 3 случая использования:

```
171  function transferFrom(address _from, address _to, uint256 _value) returns (bool) {
172      var avail = allowances[_from][msg.sender]
173              > balances[_from] ? balances[_from]
174                                : allowances[_from][msg.sender];
175      if (avail >= _value) {
176          allowances[_from][msg.sender] -= _value;
177          balances[_from] -= _value;
178          balances[_to]   += _value;
179          Transfer(_from, _to, _value);
180          return true;
181      }
182      return false;
183  }
```

Строки 176 и 177. Исходя из строки 172 переменная «avail» выбирается равной наименьшему значению из переменных «balances[_from]» и «allowances[_from][msg.sender]». Строки 176-180

исполняются только если значение «_value» меньше или равно значения «avail» Таким образом переменные «balances[_from]» и «allowances[_from][msg.sender]» не могут стать меньше «_value».

Строка 178. Проверка на переполнение переменной «balances[_to]» не производится. Однако, исходя из текущего состояния контракта и логики его работы можно установить, что баланс любого пользователя не может превзойти суммарный объем выпущенных токенов. Т.е. числа «15202911200000000». Таким образом переполнение переменной «balances[_to]» в строке 178 невозможно.

В функции «approve» обнаружен 1 случай использования:

```
190  function approve(address _spender, uint256 _value) returns (bool) {
191      allowances[msg.sender][_spender] += _value;
192      Approval(msg.sender, _spender, _value);
193      return true;
194  }
```

Строка 191. Проверка переполнения отсутствует. Вызов функции «approve» несколько раз одним и тем же пользователем для одного и того же «_spender» с большими значениями «_value» приведет к переполнению переменной allowances[msg.sender][_spender].

Однако, ввиду того, что легальному пользователю нет смысла передавать значения «value» больше суммы имеющихся у него токенов, то вероятность возникновения проблем у него низка. Намеренное переполнение данной переменной не даст каких-либо несанкционированных возможностей. Наличие функции «unapprove» дает возможность в любой момент выставить эту переменную в 0.

В функции «emission» обнаружены 3 случая использования.

```
215  function emission(uint _value) onlyOwner {
216      // Overflow check
217      if (_value + totalSupply < totalSupply) throw;
218
219      totalSupply    += _value;
220      balances[owner] += _value;
221  }
```

Строка 217. Переполнение является предусмотренной частью логики.

Строки 219 и 220. Переполнение данных переменных невозможно, ввиду проверки в строке 217.

В функции «burn» обнаружены 2 случая использования.

```
228  function burn(uint _value) {
229      if (balances[msg.sender] >= _value) {
230          balances[msg.sender] -= _value;
```

```
231         totalSupply     -= _value;
232     }

233 }
```

Строка 230. Переполнение данной переменной невозможно, ввиду проверки в строке 229.

Строка 231. Проверка на переполнение переменной «totalSupply» не производится. Однако, исходя из текущего состояния контракта и логики его работы можно установить, что «totalSupply» не может быть меньше «balances[msg.sender]». Таким образом проверка в строке 229 защищает эту переменную от переполнения.

1.2. Проверка прав доступа к ключевым функциям

Функция «setOwner» может быть вызвана только владельцем контракта. Учитывая текущее состояние контракта, владельцу присвоен адрес 0x0. И дальнейшее изменение владельца невозможно.

Функция «setHammer» может быть вызвана только пользователем, чей адрес указан как «hammer». Учитывая текущее состояние контракта, «hammer»-у присвоен адрес 0x0. И дальнейшее изменение «hammer»-а невозможно.

Функция «destroy» может быть вызвана только пользователем, чей адрес указан как «hammer». Учитывая текущее состояние контракта, «hammer»-у присвоен адрес 0x0. Вызов функции «destroy» невозможен.

Функция «emission» может быть вызвана только владельцем контракта. Учитывая текущее состояние контракта, владельцу присвоен адрес 0x0. Вызов функции «emission» невозможен.

Функция «burn» может быть вызвана кем угодно, однако сжигание токенов произойдет только в случае наличия их на балансе вызвавшего функцию.

2. Анализ логики контракта

2.1. В логике контракта отсутствует возможность прекращения эмиссии токенов по завершению ICO. Однако, ввиду того, что только владелец контракта может эмитировать токены, а владельцу присвоен адрес 0x0, дальнейшая эмиссия невозможно.

2.2. У контракта предусмотрена возможность вызова метода «suicide». Однако, ввиду того, что только «hammer» может вызвать этот метод, а ему присвоен адрес 0x0, уничтожение контракта кем-либо более невозможна.

2.3. Логика работы функции «approve» отличается от принятой для смарт-контрактов соответствующих спецификации ERC20. Так как она увеличивает количество разрешенных к управлению сторонним пользователем токенов на переданную величину, а не ставит его равным данной величине. Однако, такое решение оправдано ввиду наличия потенциальной уязвимости в общепринятом варианте

https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit#heading=h.m9fhqynw2xvt

3. Рекомендации

Метод «suicide» не рекомендован. Вместо него рекомендуется использовать «selfdestruct».

Метод «throw» не рекомендован. Вместо него рекомендуется использовать «revert()».

В модификаторах рекомендуется использовать выражение «require(condition);», вместо «if (condition) {revert();}»

Заключение

После проведения аудита исходного кода смарт контракта, и изучения его состояние нашими специалистами были сделаны выводы:

В исходном коде не содержится критических уязвимостей. Потенциальная возможность переполнения переменной в функции «approve» не представляет угрозы ввиду того, что легальному пользователю нет смысла передавать значения в нее больше суммы имеющихся у него токенов. Намеренное переполнение данной переменной не даст каких-либо несанкционированных возможностей. Наличие функции «unapprove» позволяет в любой момент выставить эту переменную в 0.

Логика контракта и текущее состояния переменных «owner» и «hammer» не предоставляет опасности интересам инвесторов.

Аудит был проведен специалистами компании Наумов Лаб:

- Василий Алексеев
- Александр Наумов

  http://smart-contracts.ru