# Аудит Smart-контрактов DApp

11.10.2018

## Исходные данные:

Адрес предоставленного исходного кода в сети Интернет: https://github.com/StayBitDev/RentalContracts

По этому адресу расположено 8 Smart контрактов: BaseEscrowLib.sol, DateTime.sol, FlexibleEscrowLib.sol, ModerateEscrowLib.sol, MyToken.sol, Ownable.sol, StayBitContractFactory.sol, StrictEscrowLib.sol.

Описание работы: https://staybit.io/demo/Help.aspx

Есть работающий DApp в тестовой сети Ropsten и в Главной сети: https://staybit.io/demo/ и https://staybit.io/main/. Разница между ними в том, что тестовая версия позволяет симулировать текущую дату и работает с тестовым токеном. Версия в Главной сети работает с TrueUSD токеном и игнорирует симуляцию даты.

## Краткое описание проекта:

В области кратковременной посуточной аренды происходит много обмана: фальшивые листинги на бесплатных сайтах предлагают перечислить всю сумму вперед за резервацию жилья, а потом арендаторы приезжают заселяться и оказываются ни с чем. Арендодатели также страдают от жуликов, которые резервируют за день до заселения и платят украденными кредитками или поддельными чеками. Данное DApp позволяет используя эскроу-договора на блокчейне Ethereum финансово обезопасить проведение сделок по аренде жилья.

## Цель проведения аудита:

Проверить утверждение, что арендаторы и арендодатели могут пользоваться этим DApp, не опасаясь за потерю своих финансовых средств. С помощью фреймворка Truffle создать тесты для проверки работоспособности основного функционала DApp.

## *Анализ исходного кода*

**Замечания:**

1. Исходный код смарт-контрактов оформлен не по code style. В связи с этим тяжело воспринимается логика работы функций, использование переменных и констант. Последняя версия code style доступна в сети Интернет по адресу: https://solidity.readthedocs.io/en/v0.4.25/style-guide.html
2. Ни в одном из представленных к изучению смарт-контрактов не используется актуальная версия Solidity – 0.4.25.

### 1. Контракт DateTime

```
24            uint private constant DAY_IN_SECONDS = 86400;
25            uint private constant YEAR_IN_SECONDS = 31536000;
```

(Обратить внимание).В языке Solidity уже есть временные константы. Данный код можно было бы записать по другому, более наглядно:

```
uint private constant DAY _IN_SECONDS = 1 days;
uint private constant YEAR_IN_SECONDS = 1 years;
```

```
251                         uint endOfDay = t2 + (60 * 60 * 24);
```

```
uint endOfDay = t2 + 1 days;
```

(Критично) В контракте не используется безопасная математика.
(Критично) Код в следующих строках является не безопасным и его исполнение может привести к непредсказуемым последствиям.

```
46          function leapYearsBefore(uint year) public constant  returns (uint) {
47                  year -= 1;
48                  return year / 4 - year / 100 + year / 400;
49          }
```

Строки 47 и 48

```
66          function parseTimestamp(uint timestamp) internal constant returns (_DateTime dt) {
67                  uint secondsAccountedFor = 0;
68                  uint buf;
69                  uint8 i;
70
71                  // Year
72                  dt.year = getYear(timestamp);
73                  buf = leapYearsBefore(dt.year) - leapYearsBefore(ORIGIN_YEAR);
74
75                  secondsAccountedFor += LEAP_YEAR_IN_SECONDS * buf;
76                  secondsAccountedFor += YEAR_IN_SECONDS * (dt.year - ORIGIN_YEAR - buf);
77
```

Строки 73, 76

```
117                 year = uint16(ORIGIN_YEAR + timestamp / YEAR_IN_SECONDS);
118                 numLeapYears = leapYearsBefore(year) - leapYearsBefore(ORIGIN_YEAR);
119
120                 secondsAccountedFor += LEAP_YEAR_IN_SECONDS * numLeapYears;
121                 secondsAccountedFor += YEAR_IN_SECONDS * (year - ORIGIN_YEAR - numLeapYears);
122
123                 while (secondsAccountedFor > timestamp) {
124                         if (isLeapYear(uint16(year - 1))) {
125                                 secondsAccountedFor -= LEAP_YEAR_IN_SECONDS;
126                         }
127                         else {
128                                 secondsAccountedFor -= YEAR_IN_SECONDS;
129                         }
130                         year -= 1;
131                 }
```

Строки 118, 121, 124, 125, 128, 130
Обезопасить использование данного кода позволяет проведение проверки на допустимость значений, используемых в этих выражениях переменных.

## 2. Контракт (BaseEscrowLib)

(Обратить внимание) Использование типов данных enum позволит сделать код более компактным. Вместо такого кода:

```
52          //Pre-Move In
53          int internal constant ContractStateActive = 1;
54          int internal constant ContractStateCancelledByTenant = 2;
55          int internal constant ContractStateCancelledByLandlord = 3;
56
57          //Move-In
58          int internal constant ContractStateTerminatedMisrep = 4;
```

можно использовать такой код:

```
enum ContractState {ACTIVE, CANCELLED_BY_TENANT,
CANCELLED_BY_LANDLORD, TERMINATED_MISREP }
```

Описания функций были такими:

```
103         function GetContractStateActive() public constant returns (int)
104         {
105                 return ContractStateActive;
106         }
107
108         function GetContractStateCancelledByTenant() public constant returns (int)
109         {
110                 return ContractStateCancelledByTenant;
111         }
112
113         function GetContractStateCancelledByLandlord() public constant returns (int)
114         {
115                 return ContractStateCancelledByLandlord;
116         }
```

Можно сократить исходный код. Вместо описания вызова функциий GetContractState..() в 9 местах достаточно описать геттер только один раз.

Можно будет заменить в библиотеке BaseEscrowLib этот код на следующий код:

```
function getContractState(ContractState _value) public pure
returns (uint result) {
       result = uint256(_value);
}
```

Тогда обращение будет осуществляться, например, так:

```
getContractState(ContractState.CANCELLED_BY_TENANT);
```

Аналогично можно поступить и для 3 функций типа GetContractStage…() и для

Рассмотрим далее исходный код этого контракта.

```
515                 else if (digit < 48 || digit > 57) {
516                     throw;
517                 }
518                 ret *= 10;
519                 ret += (digit - 48);
520             }
```

Код в строке 519 является безопасным, т.к. в строке 515 проводится проверка на допустимость значения переменной.

```
504             if (v == 0x0) {
505                 throw;
506             }
```

```
515                 else if (digit < 48 || digit > 57) {
516                     throw;
517                 }
```

(Обратить внимание) В строках 505 и 516 использование throw считается устаревшим.
В версии Solidity 0.4.10 были введены функции assert(), require()и revert().
Вызов revert() происходит в более «легких» случаях (например при вызове if/else), а вызов assert() в более «тяжелых» случаях (например: превышение предельного значения для переменной, условие, которое не должно возникнуть и т.д).
В данном случае будет предпочтительнее использовать revert().
Также использование throw приведет к расходу всего отправленного газа, а использование revert() позволяет вернуть не использованный газ вызывающей стороне.

### 3. Контракт (FlexibleEscrowLib)

```
40                  int nDaysBeforeMoveIn = (int)(self._MoveInDate - nCurrentDate) / (60 * 60 * 24);
```

(Обратить внимание) Код в строке 40 можно заменить на

int nDaysBeforeMoveIn = (int)(self._MoveInDate - nCurrentDate) / (1 days);

(Критично) Выполнение кода в строке 40 без проверки значений переменных может привести к непредсказуемым последствиям

```
78                      int nDaysAfterMoveOut = (int)(nCurrentDate - self._MoveOutDate) / (60 * 60 * 24);
79
80                      if (nDaysAfterMoveOut > ExpireAfterMoveOutDays)
81                      {
82                              int nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (60 * 60 * 24);
83                              require(self._RentPerDay * nPotentialBillableDays <= nActualBalance);
```

(Обратить внимание) Код в строке 78 можно заменить на

int nDaysAfterMoveOut = (int)(nCurrentDate - self._MoveOutDate) / (1 days);

(Обратить внимание) Код в строке 82 можно заменить на

int nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (1 days);

(Критично) Выполнение кода в строках 78 и 82 без проверки значений переменных может привести к непредсказуемым последствиям

```
182                 else if (nCurrentStage == BaseEscrowLib.GetContractStageLiving())
183                 {
184                         nPotentialBillableDays = (int)(nCurrentDate - self._MoveInDate) / (60 * 60 * 24);
```

(Обратить внимание) Код в строке 184 можно заменить на

nPotentialBillableDays = (int)(nCurrentDate - self._MoveInDate) / (1 days);

(Критично) Выполнение кода в строке 184 без проверки значений переменных может привести к непредсказуемым последствиям

```
215                 else if (nCurrentStage == BaseEscrowLib.GetContractStageTermination())
216                 {
217                     nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (60 * 60 * 24);
```

(Обратить внимание) Код в строке 217 можно заменить на

nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (1 days);

(Критично) Выполнение кода в строке 217 без проверки значений переменных может привести к непредсказуемым последствиям

## 4. Контракт (ModerateEscrowLib)

```
39                          int nDaysBeforeMoveIn = (int)(self._MoveInDate - nCurrentDate) / (60 * 60 * 24);
```

(Обратить внимание) Код в строке 39 можно заменить на

int nDaysBeforeMoveIn = (int)(self._MoveInDate - nCurrentDate) / (1 days);

(Критично) Выполнение кода в строке 39 без проверки значений переменных может привести к непредсказуемым последствиям

```
80                          int nDaysAfterMoveOut = (int)(nCurrentDate - self._MoveOutDate) / (60 * 60 * 24);
81
82                          if (nDaysAfterMoveOut > ExpireAfterMoveOutDays)
83                          {
84                                  int nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (60 * 60 * 24);
85                                  require(self._RentPerDay * nPotentialBillableDays <= nActualBalance);
```

(Обратить внимание) Код в строке 80 можно заменить на

int nDaysAfterMoveOut = (int)(nCurrentDate - self._MoveOutDate) / (1 days);

(Обратить внимание) Код в строке 84 можно заменить на

int nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (1 days);

(Критично) Выполнение кода в строках 80 и 84 без проверки значений переменных может привести к непредсказуемым последствиям

```
186                 else if (BaseEscrowLib.GetCurrentStage(self) == BaseEscrowLib.GetContractStageLiving())
187                 {
188                         nPotentialBillableDays = (int)(nCurrentDate - self._MoveInDate) / (60 * 60 * 24);
```

(Обратить внимание) Код в строке 188 можно заменить на

nPotentialBillableDays = (int)(nCurrentDate - self._MoveInDate) / (1 days);

(Критично) Выполнение кода в строке 188 без проверки значений переменных может привести к непредсказуемым последствиям

```
223                 {
224                         nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (60 * 60 * 24);
```

(Обратить внимание) Код в строке 224 можно заменить на

nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (1 days);

(Критично) Выполнение кода в строке 224 без проверки значений переменных может привести к непредсказуемым последствиям

## 5. Контракт (StrictEscrowLib)

```
40                          int nDaysBeforeMoveIn = (int)(self._MoveInDate - nCurrentDate) / (60 * 60 * 24);
```

(Обратить внимание) Код  строке 40 можно заменить на

int nDaysBeforeMoveIn = (int)(self._MoveInDate - nCurrentDate) / (1 days);

(Критично) Выполнение кода в строке 40 без проверки значений переменных может привести к непредсказуемым последствиям

```
71                          int nDaysAfterMoveOut = (int)(nCurrentDate - self._MoveOutDate) / (60 * 60 * 24);
72
73                          if (nDaysAfterMoveOut > ExpireAfterMoveOutDays)
74                          {
75                                  int nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (60 * 60 * 24);
```

(Обратить внимание) Код в строке 71 можно заменить на

int nDaysAfterMoveOut = (int)(nCurrentDate - self._MoveOutDate) / (1 days);

(Обратить внимание) Код в строке 75 можно заменить на

int nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (1 days);

(Критично) Выполнение кода в строках 71 и 75 без проверки значений переменных может привести к непредсказуемым последствиям

```
177                 else if (nCurrentStage == BaseEscrowLib.GetContractStageLiving())
178                 {
179                         nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (60 * 60 * 24);
```

(Обратить внимание) Код в строке 179 можно заменить на

nPotentialBillableDays = (int)(nCurrentDate - self._MoveInDate) / (1 days);

(Критично) Выполнение кода в строке 179 без проверки значений переменных может привести к непредсказуемым последствиям

```
190                 else if (nCurrentStage == BaseEscrowLib.GetContractStageTermination())
191                 {
192                         nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (60 * 60 * 24);
```

(Обратить внимание) Код в строке 192 можно заменить на

nPotentialBillableDays = (int)(self._MoveOutDate - self._MoveInDate) / (1 days);

(Критично) Выполнение кода в строке 192 без проверки значений переменных может привести к непредсказуемым последствиям

6. **Контракт (StayBitContractFactory)**

```
49                  require(supportedTokens[tokenId]._ContractFeeBal >= amount);
50                  supportedTokens[tokenId]._ContractFeeBal -= amount;
```

Код в строке 50 является безопасным, т.к. в строке 49 производится проверка на допустимость значений переменных.

```
129             contracts[keccak256(Guid)]._Balance = contracts[keccak256(Guid)]._tokenApi.balanceOf(this) - startBalance - CalculateCreat
```

(Критично) Выполнение кода в строке 129 без проверки значений переменных может привести к непредсказуемым последствиям

```
186                 else{
187                         revert();
188                         return;
189                 }
```

```
217                 else{
218                         revert();
219                         return;
220                 }
```

```
272                 else{
273                         revert();
274                         return;
275                 }
```

(Обратить внимание) Вызов оператора return в строках 188, 219, 274 никогда не выполнится, т.к. всегда выполнение кода в строках 187, 218, 273 приведет к выходу из функции и завершению транзакции.

```
288                     if (contracts[keccak256(Guid)]._landlBal > 0)
289                     {
290                             uint landlBal = uint(contracts[keccak256(Guid)]._landlBal);
291                             contracts[keccak256(Guid)]._landlBal = 0;
292                             contracts[keccak256(Guid)]._tokenApi.transfer(contracts[keccak256(Guid)]._landlord, landlBal);
293                             contracts[keccak256(Guid)]._Balance -= landlBal;
294                     }
295
296                     if (contracts[keccak256(Guid)]._tenantBal > 0)
297                     {
298                             uint tenantBal = uint(contracts[keccak256(Guid)]._tenantBal);
299                             contracts[keccak256(Guid)]._tenantBal = 0;
300                             contracts[keccak256(Guid)]._tokenApi.transfer(contracts[keccak256(Guid)]._tenant, tenantBal);
301                             contracts[keccak256(Guid)]._Balance -= tenantBal;
302                     }
```

(Критично) Выполнение кода в строках 293 и 301 без проверки значений переменных может привести к непредсказуемым последствиям

(Критично) В функции SendTokens() из контракта при рассылке токенов арендатору и арендодателю не учитывается число десятичных знаков токена.

## 7. Контракт (MyToken)

```
86      function transferOwnership(address _newOwner) public onlyOwner {
87          newOwner = _newOwner;
88      }
89      function acceptOwnership() public {
90          require(msg.sender == newOwner);
91          OwnershipTransferred(owner, newOwner);
92          owner = newOwner;
93          newOwner = address(0);
94      }
```

(Критично) Перед выполнением присвоения нового значения адреса, нет проверки значения этого адреса на 0. В случае введения неверного значения возможно возникновения ситуации потери контроля над управлением контрактом.

(Критично) Функцию в строке 89 может вызвать любой пользователь и произвести изменение владельца контракта. В случае некорректного

значения переменной newOwner, например, сразу после установки контракта в сеть, вызов функции любым пользователем приведет к потере возможности управлением контрактом.

```
108         uint public _totalSupply;
```

(Обратить внимание) В строке 108, либо надо назвать переменную totalSupply и тогда убрать вызов функции totalSupply() из строки 136, либо сделать переменную _totalSupply внутренней:
uint private _totalSupply;

```
154         function transfer(address to, uint tokens) public returns (bool success) {
155             balances[msg.sender] = balances[msg.sender].sub(tokens);
156             balances[to] = balances[to].add(tokens);
157             Transfer(msg.sender, to, tokens);
158                     collectTransferFee(to, tokens);
159             return true;
160         }
```

(Критично) В стоках 155 и 156 изменение числа токенов происходит без проверки значения этих токенов на допустимость. Вызов данной функции с не корректными значениями приведет к непредсказуемым последствиям. Необходимо осуществлять проверку на допустимость значений.
require(tokens <= _balances[msg.sender]);
require(to != address(0));

```
187         function transferFrom(address from, address to, uint tokens) public returns (bool success) {
188             balances[from] = balances[from].sub(tokens);
189             allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
190             balances[to] = balances[to].add(tokens);
191             Transfer(from, to, tokens);
192                     collectTransferFee(to, tokens);
193             return true;
194         }
```

(Критично) В строках 188, 189 и 190 изменение числа токенов происходит без проверки значения этих токенов на допустимость. Вызов данной функции с не корректными значениями приведет к непредсказуемым последствиям. Необходимо осуществлять проверку на допустимость значений.
require(tokens <= _balances[from]);
require(to != address(0));
require(tokens <= _allowed[from][msg.sender]);

```
171         function approve(address spender, uint tokens) public returns (bool success) {
172             allowed[msg.sender][spender] = tokens;
173             Approval(msg.sender, spender, tokens);
174             return true;
175         }
```

(Критично) Перед выполнением кода в строке 172 не сделана проверка на корректность введенного значения адреса.
Необходимо осуществлять проверку на допустимость значения:
require(spender != address(0));

```
211         function approveAndCall(address spender, uint tokens, bytes data) public returns (bool success) {
212             allowed[msg.sender][spender] = tokens;
213             Approval(msg.sender, spender, tokens);
214             ApproveAndCallFallBack(spender).receiveApproval(msg.sender, tokens, this, data);
215             return true;
216         }
```

(Критично) Перед выполнением кода в строке 212 не сделана проверка на корректность введенного значения адреса.
Необходимо осуществлять проверку на допустимость значения:
require(spender != address(0));

```
237           function faucetWithdrawToken(uint tokens)
238           {
239                   require(tokens <= faucetAllowance);
240                   require((faucetWithdrawals[msg.sender] + tokens) <= faucetAllowance);
241                   balances[owner] = balances[owner].sub(tokens);
242                   balances[msg.sender] = balances[msg.sender].add(tokens);
243                   faucetWithdrawals[msg.sender] = faucetWithdrawals[msg.sender].add(tokens);
244           }
```

(Критично) В стоках 241, 242,243 изменение числа токенов происходит без проверки значения этих токенов на допустимость. Вызов данной функции с не корректными значениями приведет к непредсказуемым последствиям.
Необходимо осуществлять проверку на допустимость значений.
require(tokens <= _balances[owner]);

```
255         function collectTransferFee(address from, uint256 tokens) internal {
256                 if (from != owner)
257                 {
258                         uint256 fee = tokens.mul(transferFeeNumerator).div(transferFeeDenominator);
259                         balances[from] = balances[from].sub(fee);
260                         balances[owner] = balances[owner].add(fee);
261                         Transfer(from, owner, fee);
262                 }
263         }
```

(Критично) В стоках 258, 259, 260 изменение числа токенов происходит без проверки значения этих токенов на допустимость. Вызов данной функции с не корректными значениями приведет к непредсказуемым последствиям. Необходимо осуществлять проверку на допустимость значений.
require(transferFeeDonominator > 0);
require(fee <= _balances[from]);

```
266         function checkTransferFee(uint256 _value) public constant returns (uint){
267             return _value.mul(transferFeeNumerator).div(transferFeeDenominator);
268           }
```

(Критично) В строке 267 необходимо произвести проверку значения переменной на допустимость. Вызов данной функции с не корректными значениями приведет к непредсказуемым последствиям.
require(transferFeeDonominator > 0);

```
249         function faucetAllowanceOf(address tokenOwner) public constant returns (uint balance) {
250             return (faucetAllowance.sub(faucetWithdrawals[tokenOwner]));
251         }
```

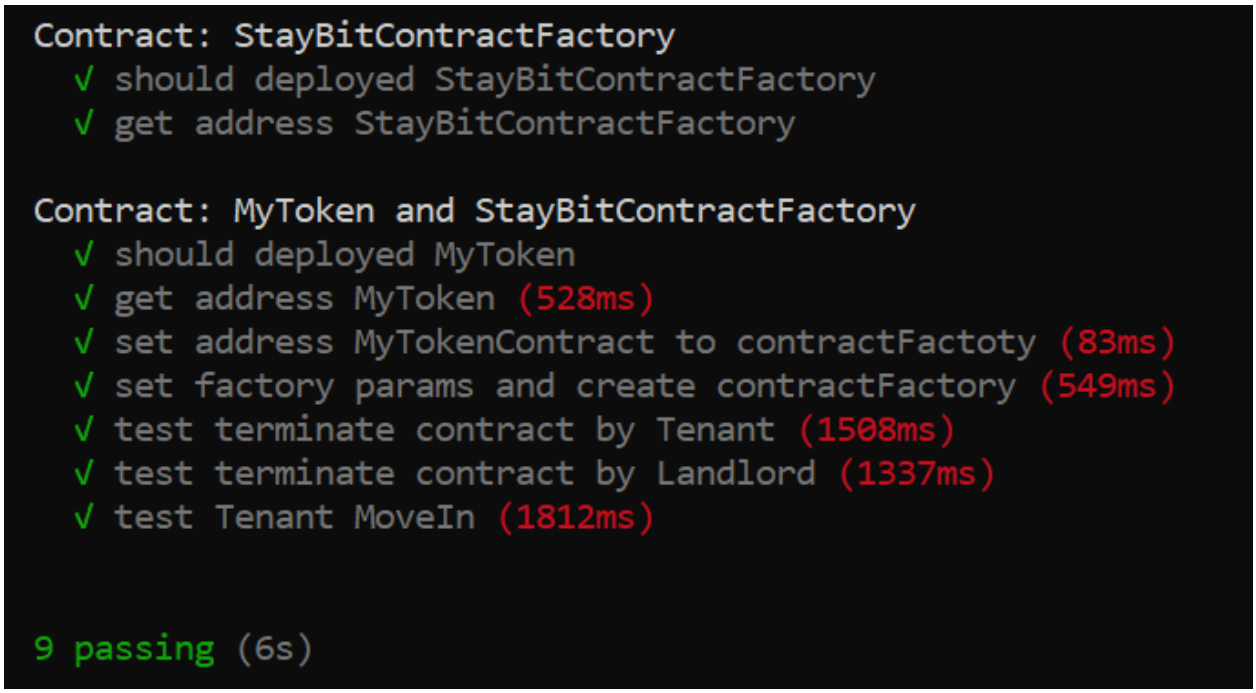
(Критично) Перед выполнением кода в строке 250 необходимо произвести проверку значения переменной на допустимость. Вызов данной функции с не корректными значениями приведет к непредсказуемым последствиям.
require(faucetAllowance >= faucetWithdrawals[tokenOwner]);

## 8. Контракт (Ownable)

К этому контракту замечаний нет. Код контракта написан безопасно.

### *Тестирование смарт-контрактов*

Для проверки работоспособности DApp, проверки логики работы и совместной работы всех смарт-контрактов были созданы тесты с помощью фреймворка Truffle. Результаты прогона тестов для основных жизненных стадий DApp приведены на скриншоте.




Исходный код тестов доступен в сети Интернет по адресу:

https://github.com/vpomo/AuditRentalContracts

В исходный код смарт-контрактов изменения не вносились, за исключением включения возможности изменения текущей даты при проведении тестирования.

Для этого были внесены изменения в исходный код контракта BaseEscrowLib. Была закомментирована строка 93 и раскомментирована строка 90.

```
89          //DEBUG or TESTNET
90          //bool private constant EnableSimulatedCurrentDate = true;
91
92          //RELEASE
93          bool private constant EnableSimulatedCurrentDate = false;
94
```

Также при проведении тестов было принято, что число десятичных знаков в токене равно 0.

Успешное проведение тестов говорит о том, что в целом исходный код контракта работоспособен. Результаты выполнения функций соответствуют заявленной логике работы.

## Вывод

1. Основная логика работы всех смарт-контрактов работоспособна. Это подтверждает проведение успешных тестов для проверки работоспособности основных жизненных стадий DApp. В исходный код смарт-контрактов изменения не вносились, за исключением включения возможности изменения текущей даты при проведении тестирования. Исходный код тестов доступен по адресу: https://github.com/vpomo/AuditRentalContracts
2. Радует факт того, что все важные функции, касающиеся жизненного цикла по сдаче в аренду недвижимости защищены от выполнения пользователями не являющимися участниками сделки.
3. В процессе аудита было найдено много критических замечаний. В основном они связаны с отсутствием использования безопасной математики SafeMath и неверной логикой работы некоторых функций.
4. Ни в одном из представленных к изучению контрактов не используется актуальная версия Solidity – 0.4.25.
5. Наименования переменных и констант не по codestyle: https://solidity.readthedocs.io/en/v0.4.25/style-guide.html
6. Создателями DApp планируется интеграция с токенами для использования их в качестве расчетной единицы при проведении финансовых сделок. Но в существующей логике контрактов не предусмотрена работа с десятичными разрядами токенов. Этот факт может привести к финансовым потерям. Например, если число десятичных знаков токена равно 18, то вместо ожидаемой суммы придет сумма в 10 в 18-ой степени раз меньшая.
7. Рекомендуется в соответствии с принципами объекто-ориентированного программирования – SOLID (Single responsibility, Open-closed, Liskov substitution, Interface segregation и Dependency inversion) разбить их на отдельные самостоятельные сущности и установить каждый по отдельности. Установка контрактов (не всех), как отдельных сущностей позволит в процессе дальнейшей работы с DApp менять отдельные модули DApp без нарушения работы всего приложения.
8. Перед началом промышленной эксплуатации этого DApp необходимо существенно переработать весь исходный код смарт-контрактов.