Аудит безопасности смарт-контрактов

Адреса смарт контрактов на MainNet Ethereum:

Lottery1ETH: *0x865EE5df064bc1F4A39B95B75e612dD86011d35b*

Lottery10ETH: *0x150dBfC384bA5C13c304EfD2Efee73Cc57cC2C16*

Lottery100ETH: *0xb02Ae0bd0e1431337fCe668d76A6BA4b6eCADD84*

RefStorage: *0x978275D7652a35DC8Df9ce6B62822Aea6A97589D*

Результат Аудита:

В коде проекта Lottery **не было обнаружено** критических уязвимостей и бэкдоров.
Контракты могут быть использованы в обороте реальных средств.

Ограничения, связанные с получением случайных чисел в среде смарт контрактов **допустимы** в связи с относительно небольшими суммами призов.

Владельцы контракта **не могут** приостановить текущий тираж билетов, а также не могут воспрепятствовать рассылке призов и менять ключевые настройки тиражей (кроме реферальной программы).

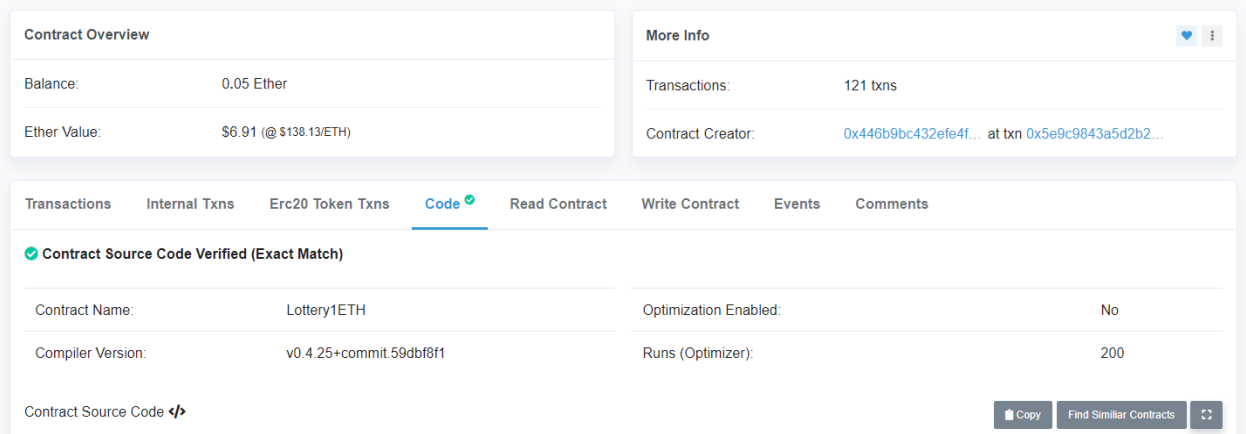
Непосредственно розыгрыш может быть инициирован любым пользователем (отправкой любого количества ETH на адрес лотереи), по прошествии нескольких блоков после приобретения последнего билета в тираже (подробнее смотрите в описании функционала). Далее (если проект не поставлен на паузу), автоматически начинается продажа нового тиража.

Отметим, что одним из минусов данного кода является отсутствие комментирования, что усложняет понимание функционала контрактов.

Выражаем благодарность за помощь в разборе кода разработчикам контрактов, прояснившим некоторые технические решения.

Описание функционала

В данном аудите рассмотрены контракты «Lottery», а также субконтракты «RefStorage», «Storage». Код каждого смарт контракта верифицирован на EtherScan и открыт к просмотру.



Все контракты используются для 3 видов лотерей: тираж 100 билетов, тираж 1 000 билетов и тираж 10 000 билетов. Во всех вариациях код контрактов идентичен за исключением таких параметров как число лотерейных билетов и победителей, призовые суммы, а также некоторые числовые значения в технической реализации получения псевдо источника энтропии (подробнее см. ниже).

> *Примечание: в данном контракте НЕ используется библиотека безопасных вычислений SafeMath, предотвращающая ошибки в вычислениях в смарт контрактах. Исходя из контекста вычислений возможных случаев переливания в контракте нет.*

> *Примечание: в данном контракте мог быть потенциальный бэкдор в рассылке призов, но для отправления ETH используется send, а не transfer. Это исключает возможность владельца противостоять рассылке средств.*

Характеристики лотереи:

Стоимость билета во всех контрактах единая и неизменяемая: 0.01 ETH. (строка 204). В одной транзакции возможно купить только 1 билет.

```
203
204     uint256 constant public PRICE = 0.01 ether;
205
```

Примечание: если отправляемая сумма меньше стоимости билета, транзакция будет отклонена, если выше – будет автоматически возвращена сдача (строки 261-263).

```
260
261         if (msg.value > PRICE) {
262             msg.sender.transfer(msg.value - PRICE);
263         }
264
```

Примечание: лотерейный билет в проекте нельзя купить с другого смарт контракта, что исключает массовую автозакупку. Для исключения данной возможности задействован специальный модификатор notFromContract (строка 227).

```
226
227     modifier notFromContract() {
228         address addr = msg.sender;
229         uint256 size;
230         assembly { size := extcodesize(addr) }
231         require(size <= 0);
232         _;
233     }
234
```

Определение победителей и розыгрыш призов происходит после продажи всех билетов в цикле (100, 1000, 10000 билетов), далее цикл начинается заново (во время покупки 1 билета в следующем цикле).

Существует 3 вида победителей: *Silver, Gold, Brilliant*.

Количество победителей / сумма выигрыша (строки 215-218):

	100 билетов	1000 билетов	10000 билетов
Silver	10 / 0.02 ETH	20 / 0.1 ETH	40 / 0.5 ETH
Gold	2 / 0.05 ETH	5 / 0.2 ETH	10 / 1.0 ETH
Brilliant	1 / 0.50 ETH	1 / 5.0 ETH	1 / 50.0 ETH

```
215
216     uint256[] silver    = [10, 0.02 ether];
217     uint256[] gold      = [2,  0.05 ether];
218     uint256[] brilliant = [1,  0.50 ether];
219
```

> *Примечание: каждый розыгрыш проводится независимо среди всех лотерейных билетов, следовательно, один и тот же билет теоретически может быть победителем нескольких призов.*

Реферальная программа:

В проекте существует единая реферальная система, реализованная с помощью отдельного контракта RefStorage. Система вынесена в отдельный контракт для того, чтобы указанный реферер закреплялся во всех видах лотерей данного проекта.

Реферер указывается в поле Data при покупке билета. Для обработки данных из поля Data в коде присутствует стандартный метод bytesToAddress (строки 357-362).

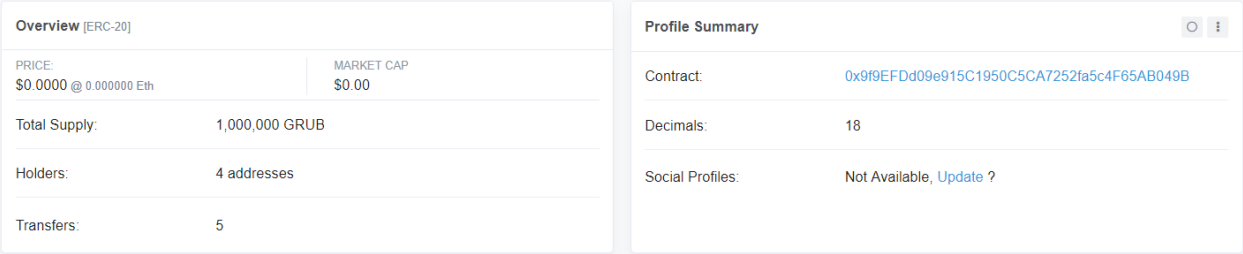
> *Примечание: ключевые параметры реферальной программы в контракте могут быть изменены владельцем. (строки 142-152)*

```
119
120     uint256 public prize = 0.00005 ether;
121     uint256 public interval = 100;
122
```

Бонус рефереру и участнику лотереи:
Переменная prize (строка 120): по умолчанию установлена как 0.00005

токенов проекта (50000000000000 без учета знаков после запятой). (актуальную информацию можно посмотреть во вкладке read contract в контракте RefStorage: *0x978275D7652a35DC8Df9ce6B62822Aea6A97589D*)

> *Примечание: в коде (строка 120) приз указан в форме 0.00005 ether что является формой написания числа 50000000000000, так как приставка ether в Solidity добавляет 18 нолей после запятой (именно столько знаков после запятой у токена проекта).*



> *Примечание: адрес контракта бонусного токена GOLD RUBLE (GRUB) – 0x9f9EFDd09e915C1950C5CA7252fa5c4F65AB049B (строка 139).*

```
137
138     constructor() public {
139         token = IERC20(address(0x9f9EFDd09e915C1950C5CA7252fa5c4F65AB049B));
140     }
141
```

Данная сумма выдается раз в определенный интервал по количеству купленных билетов, после указания реферера, а также выдается каждому победителю типа Gold.
Переменная interval (строка 121): по умолчанию установлена как 100 билетов. (актуальную информацию можно посмотреть во вкладке read contract в контракте RefStorage: *0x978275D7652a35DC8Df9ce6B62822Aea6A97589D*)
Также переменная interval задает минимальный порог купленных билетов, позволяющий участнику стать рефererом проекта.

В коде контракта RefStorage реализовано 3 ключевых метода (все они доступны для вызова только ограниченному списку адресов, по логике кода – контрактам лотерей (строки 133–136):

NewTicket (154–164) – отметка о покупке нового билета, в случае если с момента указания реферера куплено 100 билетов, отправляется бонус как пользователю, так и рефереру. Бонус высылается каждые 100 билетов, при условии если на балансе контракта достаточно токенов.

```
153
154     function newTicket() external restricted {
155         players[tx.origin].tickets++;
156         if (players[tx.origin].referrer != address(0) && (players[tx.origin].tickets - players[tx.origin].checkpoint) % interval == 0) {
157             if (token.balanceOf(address(this)) >= prize * 2) {
158                 token.transfer(tx.origin, prize);
159                 emit BonusSent(tx.origin, prize);
160                 token.transfer(players[tx.origin].referrer, prize);
161                 emit BonusSent(players[tx.origin].referrer, prize);
162             }
163         }
164     }
165
```

AddReferrer (166–173) – закрепление реферера.

> *Примечание: реферер указывается единоразово и не может быть изменен в будущем.*

```
165
166     function addReferrer(address referrer) external restricted {
167         if (players[tx.origin].referrer == address(0) && players[referrer].tickets >= interval && referrer != tx.origin) {
168             players[tx.origin].referrer = referrer;
169             players[tx.origin].checkpoint = players[tx.origin].tickets;
170
171             emit ReferrerAdded(tx.origin, referrer);
172         }
173     }
174
```

SendBonus (175–181) – отправка бонуса пользователю, по логике кода – победителю типа Gold.

```
174
175     function sendBonus(address winner) external restricted {
176         if (token.balanceOf(address(this)) >= prize) {
177             token.transfer(winner, prize);
178
179             emit BonusSent(winner, prize);
180         }
181     }
182
```

Контракт Storage:

Для функционирования системы отбора победителей Gold в каждом виде лотереи присутствует идентичный контракт «Storage» (строки 58–112). Данный контракт представляет собой временное хранилище данных об актуальных лидерах по количеству купленных билетов. Каждый цикл во время розыгрыша данный контракт редеплоится (строка 331). Данное техническое решение реализовано в контракте в виду того, что менее затратно создавать новое хранилище для данных, нежели очищать прежнее.

В данном контракте реализовано два метода:
Purchase (70–91) – метод доступный для вызова только для основного контракта лотереи. Логика функционирования данного метода сохраняет за пользователем количество купленных билетов в данном цикле, а также при надобности стирает предыдущие записи.

```
69
70      function purchase(address addr) public {
71          require(msg.sender == game);
72
73          amount[addr]++;
74          if (amount[addr] > 1) {
75              level[amount[addr]].push(addr);
76              if (amount[addr] > 2) {
77                  for (uint256 i = 0; i < level[amount[addr] - 1].length; i++) {
78                      if (level[amount[addr] - 1][i] == addr) {
79                          delete level[amount[addr] - 1][i];
80                          break;
81                      }
82                  }
83              } else if (amount[addr] == 2) {
84                  count++;
85              }
86              if (amount[addr] > maximum) {
87                  maximum = amount[addr];
88              }
89          }
90
91      }
92
```

Draw (93–110) – инфофункция, передающая лидеров по купленным билетам в текущем цикле. К данному методу обращается основной контракт проекта во время рассылки призов.

```
92
93      function draw(uint256 goldenWinners) public view returns(address[] addresses) {
94
95          addresses = new address[](goldenWinners);
96          uint256 winnersCount;
97
98          for (uint256 i = maximum; i >= 2; i--) {
99              for (uint256 j = 0; j < level[i].length; j++) {
100                 if (level[i][j] != address(0)) {
101                     addresses[winnersCount] = level[i][j];
102                     winnersCount++;
103                     if (winnersCount == goldenWinners) {
104                         return;
105                     }
106                 }
107             }
108         }
109
110     }
111
112 }
113
```

Основной контракт лотереи:

Функция конструктора контракта (235-241) вызывается единоразово при деплое контракта в сеть, в коде устанавливаются переменные для хранения адресов контрактов Storage, RefStorage, псевдотокены LotteryTicket и WinnerTicket, а также обновляется счетчик проведенных розыгрышей gameCount.

> *Примечание: во вкладке Read Contract на EtherScan всегда можно увидеть сколько тиражей билетов уже розыграны посмотрев счетчик gameCount.*

```
234
235     constructor(address RS_Addr) public {
236         x = new Storage();
237         LT = new LotteryTicket();
238         WT = new WinnerTicket();
239         RS = RefStorage(RS_Addr);
240         gameCount++;
241     }
242
```

Покупка билета осуществляется с помощью FallBack функции, автоматически вызываемой при отправлении на адрес контракта ETH (строки 243–279). Для корректного функционирования в коде прописано множество проверок и условий, для проведения розыгрыша, установки реферера, возвращения сдачи, покупки билета.

```
242
243     function() public payable notFromContract {
244
245         if (players.length == 0 && paused) {
246             revert();
247         }
248
249         if (players.length == limit) {
250             drawing();
251
252             if (players.length == 0 && paused || msg.value < PRICE) {
253                 msg.sender.transfer(msg.value);
254                 return;
255             }
256
257         }
258
259         require(msg.value >= PRICE);
260
261         if (msg.value > PRICE) {
262             msg.sender.transfer(msg.value - PRICE);
263         }
264
265         if (msg.data.length != 0) {
266             RS.addReferrer(bytesToAddress(bytes(msg.data)));
267         }
268
269         players.push(msg.sender);
270         x.purchase(msg.sender);
271         RS.newTicket();
272         LT.emitEvent(msg.sender);
273         emit NewPlayer(msg.sender, gameCount);
274
275         if (players.length == limit) {
276             drawing();
277         }
278
279     }
280
```

Принцип розыгрыша:

Механизм розыгрыша действует в две транзакции: установка будущего отсчетного блока и непосредственно розыгрыш.

Весь ключевой функционал находится во внутренней функции Drawing (строки 281–342), которая автоматически вызывается при любой отправке ETH (даже 0), когда все билеты в тираже выкуплены.

```
280
281     function drawing() internal {
282
```

Другими словами, чтобы инициировать розыгрыш призов после выкупа всего тиража, необходимо выслать любое количество ETH на адрес контракта (даже 0). Если будет выслано более 0.01 ETH автоматически будет куплен первый билет в следующем цикле.

Во время покупки последнего билета в цикле (100, 1000 или 10000), устанавливается определенный будущий отсчетный блок эфириума: +10 блоков для первого контракта, +20 для второго и +40 для третьего (строка 286).
Именно цепочка хешей блоков (10, 20 или 40) независимо определяет победителей.

```
284
285         if (block.number >= futureblock + 240) {
286             futureblock = block.number + 10;
287             return;
288         }
289
```

Следующая транзакция должна пройти в период начиная с отсчетного блока до 250 блоков с предыдущей транзакции. Все транзакции по проведению рассылки призов или покупке билетов в период ожидания отсчетного блока будут отклонены (строка 283).

```
282
283         require(block.number > futureblock, "Awaiting for a future block");
284
```

Примечание: в сети эфириума есть возможность получить данные только о последних 256 блоков, для всех остальных же запрашиваемый хеш будет заранее известен – 0. Данное ограничение учтено в коде лотереи, и в случае превышения 250 блоков отсчетный блок будет установлен заново (строки 288-290).

Далее происходит определение победителей и рассылка призов в следующем порядке:

1) Для победителей типа Silver используется простой принцип определения победителя: один хеш блока определяет одного победителя (строки 292-297).
Логика не требует усложнений ввиду того, что награда за данную позицию составляет от 0.02 до 0.5 ETH, что является небольшой суммой, следовательно, манипулировать хешами блоков при помощи больших майнерских мощностей становится экономически невыгодно.

```
291
292         for (uint256 i = 0; i < silver[0]; i++) {
293             address winner = players[uint((blockhash(futureblock - 1 - i))) % players.length];
294             winner.send(silver[1]);
295             WT.emitEvent(winner);
296             emit SilverWinner(winner, silver[1], gameCount);
297         }
298
```

2) Победители типа Gold известны заранее: ими являются те пользователи, которые приобрели наибольшее количество билетов в данном цикле. (строки 299-313)
Запрос о победителях делается в субконтракт Storage (строка 306).

Примечание: из двух пользователей, купивших одинаковое количество билетов наивысшее место занимает тот, чья транзакция по покупке последнего билета была первой в сети эфириума.

```
298
299         uint256 goldenWinners = gold[0];
300         uint256 goldenPrize = gold[1];
301         if (x.count() < gold[0]) {
302             goldenWinners = x.count();
303             goldenPrize = gold[0] * gold[1] / x.count();
304         }
305         if (goldenWinners != 0) {
306             address[] memory addresses = x.draw(goldenWinners);
307             for (uint256 k = 0; k < addresses.length; k++) {
308                 addresses[k].send(goldenPrize);
309                 RS.sendBonus(addresses[k]);
310                 WT.emitEvent(addresses[k]);
311                 emit GoldenWinner(addresses[k], goldenPrize, gameCount);
312             }
313         }
314
```

3) Для победителей типа Brilliant используется усложнённая логика определения победителя: на исход розыгрыша влияет цепочка независимых блоков (строки 315–327).
Для тиража 100 билетов используется 7 блоков, для 1000 – 10 блоков, для 10000 – 14 блоков (строка 315).
Процесс подбора победителя следующий: с каждым задействованным блоком эфириума выборка лотерейных билетов уменьшается в 2 раза (строки 319–321).

```
314
315         uint256 laps = 7;
316         uint256 winnerIdx;
317         uint256 indexes = players.length * 1e18;
318         for (uint256 j = 0; j < laps; j++) {
319             uint256 change = (indexes) / (2 ** (j+1));
320             if (uint(blockhash(futureblock - j)) % 2 == 0) {
321                 winnerIdx += change;
322             }
323         }
324         winnerIdx = winnerIdx / 1e18;
325         players[winnerIdx].send(brilliant[1]);
326         WT.emitEvent(players[winnerIdx]);
327         emit BrilliantWinner(players[winnerIdx], brilliant[1], gameCount);
328
```

Пример первого вида лотереи, поэтапное определение победителя с использованием цепочки из 7 блоков:
100 – 50 – 25 – 12.5 – 6.25 – 3.125 – 1.5625 – 1.

Каждому лотерейному билету соответствует единственная неповторимая комбинация хешей блоков. Точное определение победного билета происходит за счет использования в расчетах множителя 1e18 (строки 317, 324).

Следовательно, чтобы выиграл определенный билет в тираже 10 000 билетов, необходимо чтобы 14 блоков подряд хеш соответствовал необходимому для выигрыша данного билета, если хотя бы один хеш будет изменен то победителем станет другой билет.

Далее происходит обновление цикла (строки 329–332).

```
328
329         players.length = 0;
330         futureblock = 0;
331         x = new Storage();
332         gameCount++;
333
```

Также восполняется стоимость определения победителей и рассылки призов отправителю транзакции (строки 290, 334–336). Другими словами, все расходы на отправку транзакции покупки первого билета оплачиваются с баланса смарт контракта, а пользователь также приобретает билет.

```
333
334         uint256 txCost = tx.gasprice * (gas - gasleft());
335         msg.sender.send(txCost);
336         emit txCostRefunded(msg.sender, txCost);
337
```

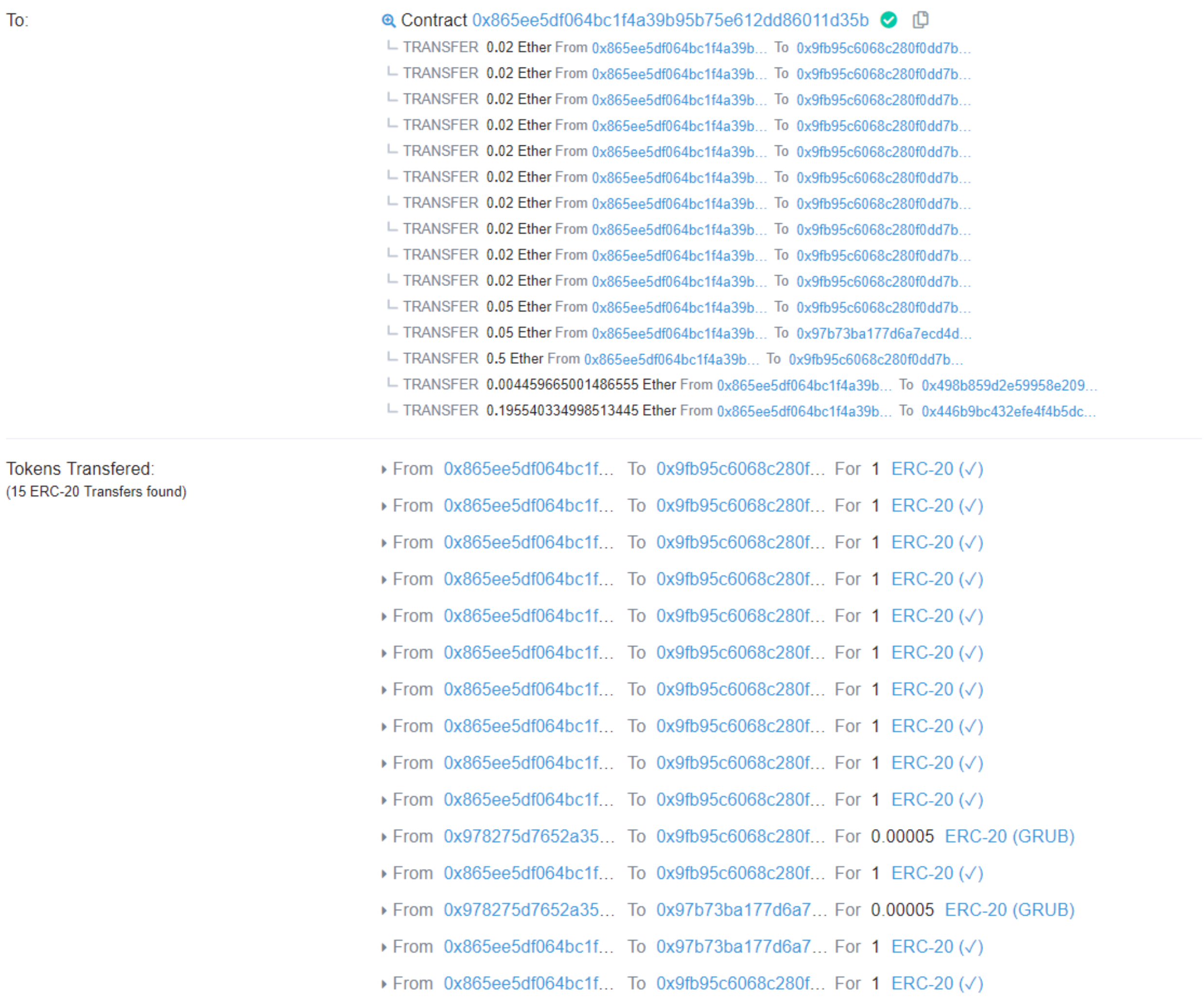
Остаток отправляется владельцу проекта (строки 338–340).

```
337
338         uint256 fee = address(this).balance - msg.value;
339         owner.send(fee);
340         emit FeePayed(owner, fee);
341
```

После успешного выполнения метода drawing происходит стандартная покупка первого билета в новом цикле.

Пример транзакции:

Пример тестовой транзакции розыгрыша призов в тираже 100 билетов можно увидеть тут: https://etherscan.io/tx/0x3be1c7d6a475125927e643ec61bdd7a7bec1948cb5f47295ec3772bbe43b3bbf



To: Contract 0x865ee5df064bc1f4a39b95b75e612dd86011d35b

└ TRANSFER 0.02 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.02 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.02 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.02 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.02 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.02 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.02 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.02 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.02 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.02 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.05 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.05 Ether From 0x865ee5df064bc1f4a39b... To 0x97b73ba177d6a7ecd4d...
└ TRANSFER 0.5 Ether From 0x865ee5df064bc1f4a39b... To 0x9fb95c6068c280f0dd7b...
└ TRANSFER 0.004459665001486555 Ether From 0x865ee5df064bc1f4a39b... To 0x498b859d2e59958e209...
└ TRANSFER 0.195540334998513445 Ether From 0x865ee5df064bc1f4a39b... To 0x446b9bc432efe4f4b5dc...

Tokens Transfered:
(15 ERC-20 Transfers found)

From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)
From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)
From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)
From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)
From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)
From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)
From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)
From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)
From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)
From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)
From	0x978275d7652a35...	To	0x9fb95c6068c280f...	For	0.00005	ERC-20 (GRUB)
From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)
From	0x978275d7652a35...	To	0x97b73ba177d6a7...	For	0.00005	ERC-20 (GRUB)
From	0x865ee5df064bc1f...	To	0x97b73ba177d6a7...	For	1	ERC-20 (✓)
From	0x865ee5df064bc1f...	To	0x9fb95c6068c280f...	For	1	ERC-20 (✓)

Как видно из транзакции: в первую очередь были отправлены призы типа Silver (10 раз по 0.02 ETH), далее 2 приза по 0.05 ETH для победителей типа Gold, 0.5 ETH для Brilliant, далее была восстановлена стоимость транзакции

для отправителя ~ 0.00446 ETH, и остаток отправляется на кошелек разработчика ~ 0.1955 ETH.

> *Примечание: в данном тираже практически все билеты были выкуплены с одного адреса, и только второй приз Gold был выдан другому адресу.*

Прочее:

В контракте реализована функция паузы (строки 214, 245-247, 251, 344-350). В случае постановки лотереи на паузу работа контракта прекращается только после окончания текущего розыгрыша, т.е. пауза предотвращает только покупку первого билета и не может вмешаться в ход текущего цикла.

В коде присутствует функция вывода токенов ERC20 с контракта (судя по всему для очистки контракта от баунти и рекламных кампаний) (строки 352-355).

```
351
352     function withdrawERC20(address ERC20Token, address recipient) external onlyOwner {
353         uint256 amount = IERC20(ERC20Token).balanceOf(address(this));
354         IERC20(ERC20Token).transfer(recipient, amount);
355     }
356
```

Инфофункции доступные во вкладке read на etherscan:
AmountOfPlayers - количество проданных билетов в текущем цикле.
ReferrerOf - реферер пользователя (если такой имеется).
TicketsOf - общее количество когда-либо купленных билетов пользователем.

Также в коде присутствует стандартный контракт Ownable (36-56) для имплементации права владения контрактом, краткий интерфейс стандарта ERC20 (3-6).

В контрактах реализована система ивентов (220-225) для передачи информации о событиях в блокчейне во внешнюю среду.

```
219
220     event NewPlayer(address indexed addr, uint256 indexed gameCount);
221     event SilverWinner(address indexed addr, uint256 prize, uint256 indexed gameCount);
222     event GoldenWinner(address indexed addr, uint256 prize, uint256 indexed gameCount);
223     event BrilliantWinner(address indexed addr, uint256 prize, uint256 indexed gameCount);
224     event txCostRefunded(address indexed addr, uint256 amount);
225     event FeePayed(address indexed owner, uint256 amount);
226
```

В коде вызов ивента происходит с использованием маркера emit.

Имплементированы два псевдотокена для вызова специальных ивентов (8-34) LotteryTicket и WinnerTicket покупателям лотерейного билета и победителям соответственно. Вызов данных ивентов отображается на EtherScan как отсылка токена ERC20 с соответствующим названием. Данные псевдотокены не несут реальной ценности и не влияют на функционал контрактов.