CYBER3

EXPERT REPORT

on the results of the security audit
of the DAO.casino gambling platform

Table of Contents

1

General provisions

Generally accepted abbreviations

Introduction

This expert report contains the results of the security audit of the gambling platform (hereinafter referred to as “the System”) owned by DAO.casino (hereinafter referred to as “the Company”) with detailed recommendations on improving its current security level.

Abbreviation	Meaning
EVM	Ethereum virtual machine
Gas	Smart contract execution unit
RSA	Cryptographic algorithm with an open key based on the computational complexity of large integers factorization
SDK	Software development kit
PRNG	Pseudo-random number generator
IS	Information security
UI	User interfase

Table 1.2–1. Generally accepted abbreviations

Abstract

The experts conducted the security audit of the System in the period of 19.03. 2018 – 11.04.2018.

In the scope of performed works, the auditors used and examined several attacker models.

The performed works have revealed a few critical vulnerabilities associated both with smart contracts and modules that perform the logic of a bankroller and player. By exploiting the vulnerabilities described below, a bankroller can manipulate with the balance in his/her favor; and a user can remotely execute code on the bankroller’s side.

The overall security level of the System was rated “average.”

Below, you can find the detailed description of the found security imperfections and related security risks.

The recommendation on fixing the vulnerabilities are provided as well.

Scope of the security audit

The scope of performed works included analysis of the security of the Company's external resources listed in Table 1.4. and Table 1.5.:

№	Project
1	dc-dev
2	dc-messaging
3	bankroller-core
4	protocol
5	dclib

Table 1.4. The list of the Company's github projects provided for the security audit

The source code used to conduct the audit is located at github.com

№	Host name/IP-address/Resource
1	dev.dao.casino (an example of a deployed platform)

Table 1.5. The list of the Company's hosts provided for the security audit

2 Security audit principles

IS threats

There are 3 types of IS threats that can affect the Company's information resources: violations of confidentiality, integrity or availability of information.

Confidentiality violation is usually aimed at information disclosure. If a violation is successful, the information becomes known to people, who should not have access to it: unauthorized personnel of the Company, clients, partners, competitors, and third parties.

Integrity violation is aimed at modification or corruption of the information, which can lead to modification of its structure or content, and to complete or partial destruction of the data.

Availability violation (denial-of-service threat) involves data system users' inability to access the information.

The core principle of this IS audit is an estimation of exploitation likelihood of the aforementioned threats affecting the Company's information resources, within the framework of a pre-defined attacker model.

Attacker model

A potential attacker is an individual or a group of individuals, acting either in a collusion or independently, whose intended or unintended actions can carry the aforementioned IS threats, infringe information resources of the System or negatively affect the Company's interests. IS threats are basic threats to confidentiality and integrity of information and the threat of the System denial-of-service.

Attackers can pursue the following goals (and their possible combinations):

- Cause Denial of Service
- Escalate their privileges in the System
- Get unauthorized access to business-critical data

In the scope of work, the auditors used several attacker models.

Tested attacker models

During the security audit, with regard to the White Paper, the following attacker models were tested:

- Player
- Bankroller
- External attacker from the Ethereum network

Since the code of the platform is open, it is considered that an attacker has knowledge of the System.

3

Discovered vulnerabilities

Non-secure function call

SDK security audit

Severity:
high

Likelihood
of exploitation:
low

Overall
impact level:
medium

Description:

Playing activity data is passed from a player to a bankroller as messages. In these messages, the precise name of an action is specified. The parties use the name as the name of the function they should call at the given stage of the game.

Impact:

An attacker can call arbitrary functions (i.e., there is the likelihood of RCE or race condition attacks).

Vulnerable project:

- Bankroller-core

Technical details:

DApp.js line 213:

```
let returns = User.logic[data.func.name].apply(this, args)
```

The data.func.name parameter is not filtered in any way, which makes it possible for an attacker (the Dice game considers an attacker to be a player) to execute arbitrary code on the bankroller's side. When it comes to more complex games, where both a player and a bankroller perform their actions, a bankroller can launch an attack on a player.

Recommendations:

Considering arguments controlled by "the other party" are passed into the function, we recommend the Company to implement strict validation of received messages on the SDK level.

Example:

In the config file, a developer sets the type of messages that may be passed between a player and bankroller:

```
msgconfig =
   {
      «roll»:{
         «bet»:»int»,
         «num»:»int»,
         «seed»:»int»,
         «pass»:»bool»
      },
      «turn»:{
         «action»:[
             «right»,
             «left»,
             «up»,
             «down»
         ]
      }
   }
```

When yet another message from a player or bankroller is received, the SDK validates, whether it matches the scheme, and passes the data to the type specified in the config file. For example, if «bet» is of the int type, then the SDK will use the data from the message and turn it into the int type:

```
data[«roll»].push({«bet»:parseint(msg[«roll»][«bet»]}))
```

After that, the data is passed to a developer into a special function “game”. Those games that include passing actions (see tur in msgconfig) should the “enum” construction or alike and validate it accordingly, i.e., check that:

```
msgconfig[‘turn’][‘action’].indexOf(msg[‘turn’][‘action’]) > -1
```

A developer's code should comply with the following pattern:

```
function game(msgType, data) {
   switch(msgType) {
   case 'roll':
      roll(data);
      break;

   case 'turn':
      ...
      [break]

      default:
         ... // the control must not be passed here since
         if a message does not match any of the types, it
         should be discarded
         [break]
      }
   }

   function roll(data){
      ...
   }
```

The control is passed to the sole function to task a developer with validating the order of receiving the other party's messages. For example, until "roll" is called, on cannot call "turn." This should help avoid race conditions.

Status:

- Fixed at PR 45

Notice: Now the only "Game" function is called, but args are still not processed. So, a game developer should carefully process them before using.

Gaming messages are not authenticated

Severity: high

Likelihood of exploitation: average

Overall impact level: high

Description:

Playing activity data is passed from a player to a bankroller as messages with the help of the swarm mechanism of a distributed open system IPFS. However, only those messages related to a smart contract operation have signatures.

Impact:

An attacker can enter a room and send arbitrary actions on behalf of both parties.

Vulnerable project:

- Ds-messaging

Technical details:

Only data passed to a smart contract is signed. Nonetheless, it is necessary to sign all the passed messages with a digital signature.

Recommendations:

- Sign all messages or create a private data transmission channel

Status:

- Fixed at PR 7

No client-side checks for a channel being opened by a bankroller

Severity: high

Likelihood of exploitation: average

Overall impact level: high

Description:

A player sends the data required to open a channel to a bankroller. However, a player does not check whether the channel is opened or not.

Impact:

A bankroller may pretend it has opened a channel instead of actually opening it. When a game is finished a bankroller decides: to open a channel if a player has lost, or do nothing if a player has won.

Vulnerable role:

- Player

Technical details:

In case this vulnerability is successfully exploited, a player will not be able to prove anything if a dispute occurs since the latter should be opened via id channel that has not been created.

Recommendations:

- Implement the check for a channel being opened by a player
- All actions of both parties should be checked as well.
  If anomalous behavior is detected, a player should be offered to open a dispute.

Status:

- Fixed at #e08c8d7c75d3eb34ccf3543441ef30968252c6d6

Note: there are changes in the game protocol here. Now Player is responsible for opening the channel and Bankroller for checking that it really happened.

No duplication check

Severity:
high

Likelihood
of exploitation:
average

Overall
impact level:
high

Description:

A bankroller does not perform duplication check for the seed sent by a player.

Impact:

A player can find a winning seed and send it multiple times.

Vulnerable role:

- Bankroller

Recommendations:

- Use one public key per a game
- Perform duplication check for received seed

Status:

- Fixed at #8ed4f906e357df659bfdcbb78e489dd911f6042e

Player does not validate random numbers

Severity:
high

Likelihood
of exploitation:
average

Overall
impact level:
high

Description:

The parties generate random numbers by the Signidice algorithm. Meanwhile, on the player's side, there is no validation of the RSA signature produced by a bankroller.

Impact:

In such a case, the game results data from a bankroller's perspective can be easily manipulated.

Vulnerable role:

- Player

Technical details:

After a bankroller signs a random number that has been sent along with a bet by a user, a bankroller has to send the signature back to validate it. If the signature is valid, a player may use it for the PRNG to learn game results.

Recommendations:

- Check RSA signatures

Status:

- Fixed at PR 42

Discovered security weaknesses of the System

Bankroller ties up more funds in a channel than a player (DOS)

Description:

When a channel is opened, a bankroller ties up more funds in a channel than a player.

Impact:

An attacker can create multiple games with a bankroller until the funds are depleted.

Vulnerable role:

- Bankroller

Technical details:

It is clear from an example of the Dice game that a bankroller ties up two times more funds than a player. As the developer claims, the proportion may be even higher for other games. This enables an attacker to create multiple rooms (until a bankroller is out of funds) and linger with interacting with a bankroller. This will result in a bankroller being unable to serve other players. In case of Dice, an attack will require 2 times fewer funds than a bankroller has at its disposal.

Recommendations:

- Even the amount of funds in a channel and check a player's gain, i.e., do not let a player make a bet with the gain exceeding the amount a bankroller's funds

Status:

- After discussion with DAO.casino team we decide that this is busines logic requirement

Insufficient balance validation

Description:

To the moment, the check of a bankroller's and player's balances is implemented in a smart contract only.

Impact:

If bets are not checked at both sides, the gain sum may exceed the funds in a channel.

Vulnerable role:

- Player, Bankroller

Recommendations:

- Check bets on a bankroller's side
- Implement checks that ensure a bankroller plays the bet sent by a player

Status:

- Fixed at PR 42 and PR 10

Using a signature to generate a random number different from the one described in the White Paper

Description:

To generate a random number according to the Signidice algorithm, a bankroller uses the ECDSA signature instead of RSA (as described in the White Paper).

Impact:

A bankroller gets an opportunity to generate a valid but different signature, which may lead to fraudulent actions by a bankroller.

Vulnerable role:

- Player

Recommendations:

- Change the RSA signature; take into consideration the recommendations listed in the par. 3.1.4 and 3.1.5

Status:

- Fixed at PR 42 and PR 10.

Discovered vulnerabilities of the System

Bankroller can manipulate with RSA keys

Severity:
high

Likelihood
of exploitation:
average

Overall
impact level:
high

Description:

To save the Memory, a smart contract stores a fingerprint of a Bankroller's public key only: fingerprint = keccak256 (N, E).

Impact:

If the described type of data storing is used, a bankroller has an opportunity to manipulate the "public key – private key" pair, without changing the fingerprint. This means that it is possible to manipulate with the signature of the Signidice algorithm (obtaining random numbers).

Vulnerable project:

- Protocol. OneStepGame.sol:149

Technical details:

The keccak256 hashing function implemented by EVM has the following feature:

```
keccak256(«123», «123») == keccak256(«12», «3123»)
```

That is why claiming the keys are N and e equal «0xDEAD» and «0xBEEF», correspondingly, a bankroller can recalculate its key in relation to new N and e («0xDE» and «0xADBEEF», correspondingly) and generate a winning signature for the Signidice algorithm.

Recommendations:

- Use Merkle Tree to store a public key:

```
fingerprint = keccak256(keccak256(N),keccak256(e))
```

Status:

- Fixed at PR 20

Mistype resulting in a postponed dispute opening


Severity: average


Likelihood of exploitation: average


Overall impact level: average

Description:

Instead of subtraction, the logic of time-checking of dispute opening uses division.

Impact:

It is possible to open a dispute much later than required.

Vulnerable project:

- Protocol. OneStepGame.sol:301

Technical details:

There is the following check in the dispute opening function:

```
if (c.endBlock.div(block.number) < safeTime) {
     c.endBlock = c.endBlock.add(safeTime)
}
```

Possibly, it was supposed to use the sub operator in spite of div (the checks of the kind in other functions and add operations in the code prove this).

Recommendations:

- Change div to sub.

Status:

- Fixed before PR 20

Roundoff error


Severity: average


Likelihood of exploitation: average


Overall impact level: average

Description:

The smart contract can operate only with integers. In certain circumstances, may result in missing least significant digits of the obtained number.

Impact:

If such operations deal with fund charging (e.g., tokens), the parties may get an incomplete sum.

Vulnerable project:

- Protocol. OneStepGame.sol:374

Technical details:

In the function of channel closing, there are the following computational procedures:

```
uint _forReward = c.totalBet.div(100).mul(2);
```

It should be noted that the first performed operation is division (2 least significant digits are missed), and multiplication is the second one.

Recommendations:

- To narrow down the number of potential and actual roundoff errors, perform mathematical operations (SafeMath libraries) in the following order:
  1. mul
  2. add | sub
  3. div

Status:

- Fixed before PR 20

Improper balance checking

Severity:
low

Likelihood
of exploitation:
low

Overall
impact level:
low

Description:

When a channel status is updated, the balance is checked. However, in the current implementation, the check provides no relevant results.

Impact:

When a vulnerability enabling manipulation of balance is being exploited, a proper check might have made it either more difficult or impossible the successful exploitation.

Vulnerable project:

- Protocol. OneStepGame.sol:180

Technical details:

There is the following check in the function of updating the status of a channel:

```
require(_playerBalance.add(_bankrollerBalance) <=
c.playerBalance.add(c.bankrollerBalance))
```

In this case, it is necessary to use ==, but as the developer states, the client side of the platform may return inaccurate values, which disrupts the logic of operation with smart contract. Nonetheless the <= group of symbols commits a bigger computational margin (up to_playerBalance.add (_bankrollerBalance) == 0).

Recommendations:

- Use a special JS library on the client side of the platform to get more accurate figures (accordingly, change the operator to ==)

Status:

- Fixed at PR 20

Discovered security weaknesses of the System

Low dispute opening time

Description:

A smart contract enables operations with the update Channel and opendispute functions in the certain operational timeframe. The timeframe, however, is too narrow.

Impact:

Either of the parties may be unable to interact with a smart contract in time.

Vulnerable project:

- Protocol. OneStepGame.sol:34

Technical details:

SafeTime set as 50 (blocks), which is equal to 12 minutes.

Recommendations:

- Increase the time up to 1 hour or set parties' trust relations

Status:

- Fixed at PR 20

Description:

There is redundant code in a smart contract.

Impact:

Redundant code provokes the excessive use of Gas.

Vulnerable project:

- Protocol. Refferer.sol:33

Technical details:

The setService function has the following checks:

```
function setService(address _player, address _operator,
address _referrer) public {
      require(msg.sender == _operator);
      require(_operator != address(0));
...
```

The last check in the code above is of no use, since its functions are performed by the previous one.

Recommendations:

- Remove redundant code

Status:

- Fixed at PR 20

5

Bankroller and player execute untrusted code

Other recommendations

Description:

The games on the platform are implemented by third-party developers.

Impact:

It is possible that the platform would be used to spread malicious code.

Vulnerable roles:

- Bankroller, Player

Technical details:

In its nature, game code is an extension of a bankroller's, player's and platform's smart contract code. Therefore, it is necessary to hold code review to avoid putting malicious games into the platform.

Recommendations:

- Implement the mechanism of rating developers and games.
  Add the "Auditor" role. Users with this role assigned to them will be reputationally and financially responsible for potential incidents with the games they reviewed

Status:

- Taken into account

UI security improvement

To improve the security of the future platform UI, it is recommended to implement the following mechanisms (headers):

- X-Frame-Options: DENY Strict-Transport-Security:
  max age=31536000; includeSubDomains;
- Configure CSP (Content-Security-Policy)

Status:

- Taken into account

6 Conclusion

As a result of the works performed at the audit stage, numerous vulnerabilities were discovered. These vulnerabilities enable a bankroller or player to play unfair pool. It has been demonstrated that the used method of transferring data between the parties is non-secure. Thus, the overall security level of the System may be rated “average.”

We’ve recommended the Company to take the recommendations described in this document into account and to apply them before the deployment of the System in the Ethereum network.

It is also advised to analyze the performance and security of the dispute mechanism since at the audit stage its full capacity was utilized by the smart contract only.

After the audit document was submitted, the Company took all recommendations into account and eliminated the System vulnerabilities in the shortest time. After the necessary changes have been made, the current security level of the updated System may be rated «above average».

7 License

Appendix 1.
Security audit. Background information.

Security level analysis

To analyze the System security level, it is necessary to measure severity and likelihood of the detected vulnerabilities exploitation. The likelihood of exploitation is measured, according to the ease of vulnerability exploitation and the accessibility of a vulnerability.

Vulnerability analysis

The “Severity” property of a vulnerability describes possible results of this vulnerability exploitation, regarding confidentiality, integrity, and availability of information processed on a vulnerable resource. Severity levels are described in the Table A-1.

Severity level	Confidentiality violation	Integrity violation	Availability violation
None	Does not happen	Does not happen	Does not happen
Low	Obtaining access to a noncritical information by an attacker through privilege escalation	Integrity violation of a noncritical information by an attacker with basic user rights in the System	Short-time denial-of-service of a mission-critical application
Medium	Confidentiality violation of sensitive data by an attacker with basic user rights in the System	Integrity violation of sensitive data by an attacker with basic user rights in the System	Denial of service of a mission-critical application or a short-time denial of service of the System
High	Confidentiality violation of critical information by an attacker with administrator rights in the System	Integrity violation of critical information by an attacker with administrator rights in the System	Denial of Service of the System

Table A–1. Severity levels

Ease of vulnerability exploitation

The “Ease of exploitation” property of a vulnerability defines what hardware and software, time and computing resources, and professional skills are required to exploit a vulnerability (Table A-2).

Level	Description
Low	Vulnerability exploitation requires high computing powers, significant time resources, developing new software, configuration analysis of the System, determination and testing possible ways and conditions of successful exploitation of this vulnerability.
Medium	Vulnerability exploitation requires high-performance computing, extensive time resources, special hardware and software, and analysis of a violated system configuration. An attacker does not have to have deep knowledge of the system or professional skills to perform an attack.
High	Vulnerability exploitation does not require the use of any special hardware or software, high-performance computing, time resources or any professional skills to perform an attack.

Table A–2. Ease of vulnerability exploitation levels

Vulnerability availability

The “Availability” property of a vulnerability defines what user classes have access to a vulnerable resource (Table A-3).

Level	Description
Low	Privileged users
Medium	Registered users
High	All users

Table A–3. Availability

Likelihood of exploitation

The likelihood of exploitation is calculated according to “ease of exploitation” and “availability” levels (Table A–4).

Likelihood of exploitation		Ease of exploitation ↓		
		Low	Medium	High
Accessibility →	Low	Low	Low	Medium
	Medium	Low	Medium	High
	High	Medium	High	High

Table A–4. Likelihood of exploitation

Vulnerability impact

Vulnerability impact (for one of the existing threats) is measured, according to vulnerability severity (for one of the existing threats) and the likelihood of exploitation of a vulnerability (Table A-5).

Vulnerability impact		Likelihood of exploitation ↓		
		Low	Medium	High
Vulnerability severity →	Low	Low	Low	Medium
	Medium	Low	Medium	High
	High	Medium	High	High

Table A–5. Vulnerability impact levels

Appendix 2. The list of detected vulnerabilities and security weaknesses

The vulnerabilities and security weaknesses detected during this security audit are listed in Table B-1.

Security audit		
Vulnerability	**Overall impact level**	**See paragraph**
Gaming messages are not authenticated	High	3.1.2.
No client-side checks for a channel being opened by a bankroller	High	3.1.3.
No duplication check	High	3.1.4.
Player does not validate random numbers	High	3.1.5.
Non-secure function call	Average	3.1.1.

Security weakness	**See paragraph**
Bankroller ties up more funds in a channel than a player (DOS)	3.2.1.
Insufficient balance validation	3.2.2.
Using a signature to generate a random number different from the one described in the White Paper	3.2.3.

Smart contract code audit		
Vulnerability	**Overall impact level**	**See paragraph**
Bankroller can manipulate with RSA keys	High	4.1.1.
Mistype resulting in a postponed dispute opening	Average	4.1.2.
Roundoff error	Average	4.1.3.
Improper balance checking	Low	4.1.4.

Security weakness	**See paragraph**
Low dispute opening time	4.2.1.
Redundant code	4.2.2.

Table B-1. Detected vulnerabilities and security weaknesses of the System

CYBER3