Block-Rank

Smart Contract Security Audit

Prepared by ShellBoxes

May 18th, 2022 - July 4th, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	BlockRank Ltd
Version	1.0
Classification	Public

Scope

The Block-Rank Contract in the Block-Rank Repository

Repo	Commit Hash
https://github.com/blockrank/reach-contracts	2a40c75098a376c30b50f53ae72d7a99136c574f

Files	MD5 Hash
blockrank/delegated-airdrop.rsh	2966f75dc435b8bf8e8c22b734a58293

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

BlockRank Ltd engaged ShellBoxes to conduct a security assessment on the Block-Rank beginning on May 18th, 2022 and ending July 4th, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 Disclaimer

The Block-rank project has been terminated.Hence, the following project didn't undergo our re-audit process.The report herewith contains the findings and results of the initial audit report as is.

1.2 About BlockRank Ltd

Block-rank mission is to develop a new layer of accountability in anonymous web3 ecosystems. Transaction network security, airdrop management, and more.

Issuer	BlockRank Ltd
Website	https://block-rank.com
Type	Reach Smart Contract
Audit Method	Whitebox

1.3 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's

scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.3.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Block-Rank implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 1 high-severity, 2 medium-severity, 1 low-severity, 1 informational-severity, 1 undetermined-severity vulnerabilities.

Vulnerabilities	Severity	Status
Centralization Risk	HIGH	Not Fixed
The Admin And The Delegate Can Steal In The pay API	MEDIUM	Not Fixed
The Mnemomic Of The Payout Manager Is Exposed	MEDIUM	Not Fixed
Redundant checks	LOW	Not Fixed
Missmatch Between The Documentation And The Code	INFORMATIONAL	Not Fixed
The Architecture Used Is Not Secure By Design	UNDETERMINED	Not Fixed

3 Finding Details

A delegated-airdrop.rsh

A.1 Centralization Risk [HIGH]

Description:

The admin and the delegate account can send any amount from the contract to any user. This represents a significant centralization risk due to the amount of power that these entities have. This can be comprehensible knowing that the admin will fund the contract, however from a user perspective, he should trust the admin.

Code:

Listing 1: delegated-airdrop.rsh

```
141 .api(
142   adminAPI.pay,
143   (who, howMuch) => {
144     const authorized = this == Delegate || this == Admin;
145     check(authorized, UNAUTHORIZED);
146     check(
147       who !== Delegate && who !== Admin,
148       "Stealing is naughty and disallowed"
149     );
150     check(howMuch > 0 && howMuch <= maxPayout, "Invalid user allocation");
151     check(howMuch <= balance(token), LOW_CTC_BALANCE);
152   },
153   (_, _) => [0, [0, token]],
154   (whom, howMuch, notify) => {
155     const authorized = this == Delegate || this == Admin;
156     check(authorized, UNAUTHORIZED);
157     check(howMuch > 0 && howMuch <= maxPayout, "Invalid user allocation");
158     check(howMuch <= balance(token), LOW_CTC_BALANCE);
```

```
159
160       transfer(howMuch, token).to(whom);
161       events.payout(whom, howMuch);
162       notify(null);
163
164       return [keepGoing];
165     }
166   );
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

This issue requires a change in the architecture to be remediated, the suggested architecture is stated in the issue A4.

Status - Not Fixed

A.2 The Admin And The Delegate Can Steal In The pay API [MEDIUM]

Description:

The pay API contains a restriction that is supposed to prevent the admin and the delegate from withdrawing the airdrop. However, the admin/delegate can withdraw funds to a different wallet, therefore bypassing this restriction.

Code:

Listing 2: delegated-airdrop.rsh

```
141   .api(
```

```
142   adminAPI.pay,
143   (who, howMuch) => {
144     const authorized = this == Delegate || this == Admin;
145     check(authorized, UNAUTHORIZED);
146     check(
147       who !== Delegate && who !== Admin,
148       "Stealing is naughty and disallowed"
149     );
150     check(howMuch > 0 && howMuch <= maxPayout, "Invalid user allocation");
151     check(howMuch <= balance(token), LOW_CTC_BALANCE);
152   },
153   (_, _) => [0, [0, token]],
154   (whom, howMuch, notify) => {
155     const authorized = this == Delegate || this == Admin;
156     check(authorized, UNAUTHORIZED);
157     check(howMuch > 0 && howMuch <= maxPayout, "Invalid user allocation");
158     check(howMuch <= balance(token), LOW_CTC_BALANCE);
159
160     transfer(howMuch, token).to(whom);
161     events.payout(whom, howMuch);
162     notify(null);
163
164     return [keepGoing];
165   }
166 );
```

Risk Level:

Likelihood – 2

Impact - 4

Recommendation:

This issue requires a change in the architecture to be remediated, the suggested architecture is stated in the issue A4.

Status - Not Fixed

A.3 Redundant checks [LOW]

Description:

The local part of the fund API contains several checks to prevent integer overflows. However, the checks that exist in L118, L121 and L123 are redundant and can be reduced to one check.

Code:

Listing 3: delegated-airdrop.rsh

```
111  .api(
112    adminAPI.fund,
113    (amt) => {
114      check(this == Admin, UNAUTHORIZED);
115
116      const bal = balance(token);
117      check(bal < minBalance, "Contract balance exceeds minimum");
118      check(bal < UInt.max - amt, BALANCE_LIMIT);
119
120      const newBal = bal + amt;
121      check(newBal <= UInt.max, BALANCE_LIMIT);
122      check(amt > 0 && amt >= minBalance, MIN_BALANCE);
123      check(amt < UInt.max - bal, "Invalid amount");
124    },
```

Risk Level:

Likelihood – 2

Impact - 2

Recommendation:

It is recommended to remove the redundant checks and leave only one that prevents integer overflows.

Status - Not Fixed

A.4 The Architecture Used Is Not Secure By Design [UNDETERMINED]

Description:

The current architecture is not secure by design, causing several risks, mainly centralization issues but also the fact that the majority of the actions are triggered by the admin or the delegate open a large surface of attacks.

Recommendation:

It is recommended to change the architecture to contain the following APIs:
- An API where the admin will initialize the contract.
- An API where the airdrop creator will submit the Merkle Root to the contract.
- An API where the airdrop creator will fund the contract.
- An API where the airdrop creator will change the status of airdrop to be OPEN. Thus, anyone can claim his tokens.
- An API where the user can withdraw his amount of the airdrop after passing the Merkle Proof to ensure that he is whitelisted.
- An API where the admin can close the contract and return the rest of the funds to the airdrop creator.

Note that the admin should not be able to change the Merkle root, to ensure that the admin will not alter the list of recipients.

Status - Not Fixed

B lambda/src/index.ts

B.1 The Mnemomic Of The Payout Manager Is Exposed [MEDIUM]

Description:

The mnemonic of the payout manager is used to perform actions in the DApp, this can lead to the mnemonic getting exposed to multiple parties if the server is hacked.

Code:

Listing 4: lambda/src/index.ts

```
32 if (!mnm) {
33 // Check for (funded!) account which will call contract API
34 return errorHandler(400, `Payout Manager account is not configured`);
35 }
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

Consider using a one time generated signatures for the payout manager instead of the mnemonic to perform actions in the DApp, therefore even if someone has managed to steal the signature, he will be able to use it since it is expired.

Status - Not Fixed

B.2 Missmatch Between The Documentation And The Code [INFORMATIONAL]

Description:

The code in the lambda function accepts the passphrase and the adminAddress as parameters, but in the lambda/src/index.ts it accepts only the recipient, ctcInfo and tokenDecimals. In addition to that, the commands to run the tests in the documentation are incorrect, the command should be npm run airdrop-action instead of npm run airdrop:action.

Code:

Listing 5: lambda/src/index.ts

```
8  type Data = {
9    ctcInfo: string | number;
10   recipients: [addr: string, amt: number][];
11   tokenDecimals: number;};
```

Listing 6: README.md

```
58 interface Data {
59   ctcInfo: string | number;
60   recipients: [ addr: string, amt: number ][];
61   adminAddress: string;
62   passphrase: string;
63   tokenDecimals?: number;}
```

Recommendation:

It is recommended to adapt the documentation to match accurately what exists in the code.

Status - Not Fixed

4 Best Practices

BP.1 Errors While Running The Tests

Description:

There are multiple errors encountered while running the tests. As a best practice, it is recommended to address these errors to make the process simpler. We also recommend testing the edges cases including calling the close API with the delegate account.

5 Tests

Results:

```
* Automated Delegate Drop
        * Connected to ALGO
        * Cost: 20 UNO
:key: Checking for Mnemonic ...
:key: Found Mnemonic!
* Create AirDrop Contract
        * Connected to ALGO
Deploying Aidrop contract ...
Contract deployed at 98237545
        * Admin paid 10 #98174026
Admin done!
 Contract deployed at 98237545
Run AirDrop (ctc 98237545)
Connected to ALGO
Attaching Admin account ...
Beginning airdrop ...
0x86656c0f4bbe490f449ee10aa5c9fe3cf4862a2be6341ff0ae0d89a80ae2c60b
received 10
Airdrop complete! Funding for second run ...
```

```
Fund AirDrop (ctc 98237545)
        * Connected to ALGO
Attaching Admin account ...
        * Amount Due: 10 VAR
Funding airdrop contract ...
Added 10 UNO to contract!
Airdrop funded! ...
Run AirDrop (ctc 98237545)
Connected to ALGO
Attaching Admin account ...
Beginning airdrop ...
Airdrop complete! Closing contract ...
Close AirDrop Ctc (98237545)
Connected to ALGO
Attaching Admin account ...
Closing airdrop contract ...
0x86656c0f4bbe490f449ee10aa5c9fe3cf4862a2be6341ff0ae0d89a80ae2c60b
received 10
Contract closed! Exiting ...
```

6 Conclusion

We examined the design and implementation of Block-Rank in this audit and found several issues of various severities. We advise BlockRank Ltd team to implement the recommendations contained in all 6 of our findings to further enhance the code's security. It is of utmost priority to start by addressing the most severe exploit discovered by the auditors then followed by the remaining exploits, and finally we will be conducting a re-audit following the implementation of the remediation plan contained in this report.

We would much appreciate any constructive feedback or suggestions regarding our methodology, audit findings, or potential scope gaps in this report.

7 Disclaimer

Shellboxes reports should not be construed as "endorsements" or "disapprovals" of particular teams or projects. These reports do not reflect the economics or value of any "product" or "asset" produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

For a Contract Audit, contact us at contact@shellboxes.com