CERTIK

Security Assessment

Argon Platform

Apr 1st, 2021

Summary

This report has been prepared for Argon Platform smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

Project Summary

Project Name	Argon Platform
Description	Argon a decentralized freelancer platform utilize Argon Token
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/ahmetoznar/ArgonPlatform
Commits	1. ca548a8de3f11cc4a3a10ad6bae86fe420947dd8 2. 9d8b1a7555ea5ee33375eade8859147f00a39564

Audit Summary

Delivery Date	Apr 01, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	11
● Critical	7
● Major	1
● Minor	1
● Informational	2
● Discussion	0

Audit Scope

ID	file	SHA256 Checksum
AFS	ArgonFreelancers.sol	4135ebd954cddeb0daeaa934f7ec353af729b4c4b59c59f29bcee1f0abc0a7f5

Centralization

To bridge the trust gap between the client and users, the client needs to express a sincere attitude. The client has the responsibility to notify users with the following privileges of the `manager`:

- `manager` can send any amount of tokens to the `manager` defined address through `sendInvestorsTokens()`, `getARGONTokenDeployer()`, `getBNB()` and `getAllBNB()` functions in the `ArgonPublicSale.sol` smart contract without any restriction.

- `manager` can set the value of `active` in function `changeActive()` in the `ArgonPublicSale.sol` smart contract without any restriction.

- `employerAddress` can send any amount of tokens to the contract deployer-defined address through `sendDeadline()`, `cancelApprover()` functions in the `ArgonFreelancers.sol` smart contract without any restriction.

- `employerAddress` can update the value of `tokenContractAddress` in function `selectOfferWithToken()` in the `ArgonPublicSale.sol` smart contract without any restriction, which will lead to unknown result to the invocation of transfer() in following functions: `employerReceiveFile()`, `confirmApprover()`,`cancelApprover()` and `sendDeadline()`.

- `employerAddress` can update the value of `approverAddress` in function `selectOffer` and `selectOfferWithToken()` in the `ArgonPublicSale.sol` smart contract without any restriction, which will lead to send any amount of tokens to any address through `sendApproverArgonCoin()` function.

- `employerAddress` can update the value of `freelancerAddress` in function `selectOffer` and `selectOfferWithToken()` in the `ArgonPublicSale.sol` smart contract without any restriction, which will lead to send any amount of tokens to any address through `employerReceiveFile()` and `confirmApprover()` function.

- `ArgonTokenDeployer` can set the value of `approverMinArgonLimit` in function `changeApproverMinArgonLimit()` in the `ArgonFreelancers.sol` smart contract without any restriction.

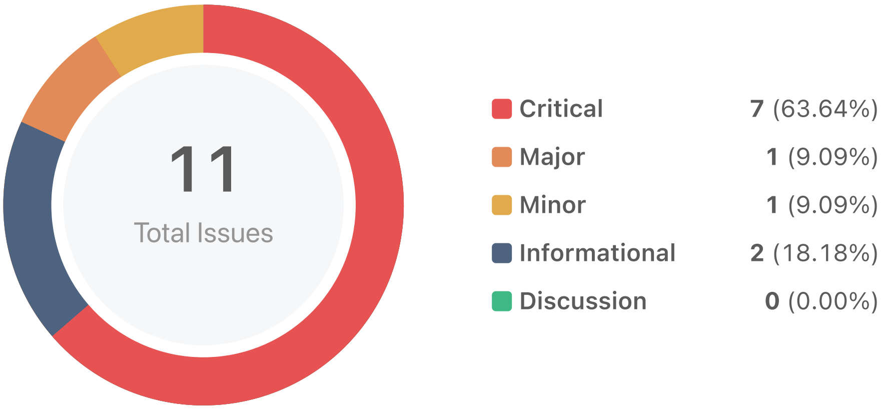
- `ArgonTokenDeployer` can send any amount of tokens to any address through `sendArgonTokenDeployer()` function in the `ArgonFreelancers.sol` smart contract without any restriction.

Besides, the _totalSupply amount of token is minted to a centralized address, which is `0x8CB5b6B0A475e760ed0610AD9cF8403Ec050bc8A`. This may raise the community's concerns about the centralized issue and potential exit scam issue.

As there's an extremely high chance to withdraw assets to any arbitrary address by the above-mentioned roles. The client should inform any sensitive changes of the project, especially about the roles, to the community to improve the trustworthiness of the project. Moreover, any dynamic runtime changes on the protocol should be made through the Timelock mechanism and notified to the community. We also strongly advise the client to adopt Multisig with community selected co-signers and/or DAO mechanism in the project.

The client team invited a community member whose telegraph handle is `@elips7` to manage the private keys of `Approvers` role in contract `ArgonFreelancers.sol` along with the Argon team. This can help to reduce the risk of being accessed by malicious users.

Findings



ID	Title	Category	Severity	Status
AFS-1	Centralized Risk	Control Flow	Critical	Partially Resolved
AFS-2	Storage Manipulation in `view` function	Language Specific	Major	Resolved
AFS-3	Division Before Multiplication	Mathematical Operations	Informational	Partially Resolved
AFS-4	Centralized Risk	Control Flow	Critical	Partially Resolved
AFS-5	Missing Emit Events	Logical Issue	Informational	Resolved
AFS-6	Lack of Sanity Check	Volatile Code	Minor	Resolved
AFS-7	Centralized Risk	Control Flow	Critical	Partially Resolved
AFS-8	Centralized Risk	Control Flow	Critical	Partially Resolved
AFS-9	Centralized Risk	Control Flow	Critical	Partially Resolved
AFS-10	Potential Reentrancy Vulnerability	Logical Issue	Critical	Resolved
AFS-11	Token Minted to Centralized Address	Logical Issue	Critical	Acknowledged

AFS-1 | Centralized Risk

Category	Severity	Location	Status
Control Flow	● Critical	ArgonFreelancers.sol: 646, 860, 862, 872, 874, 613	Partially Resolved

Description

Following functions can send any amount of tokens to the contract deployer-defined address `employerAddress` without any restriction.

- `sendDeadline()`
- `cancelApprover()`

Recommendation

We advise the client to notify contract users right after the contract deployment

Alleviation

The client adopted a centralized role `Approver` to restrict the access to the aforementioned sensitive functions in commit `9d8b1a7555ea5ee33375eade8859147f00a39564`. The client team also invited a community member whose telegraph handle is `@elips7` to manage the private keys of `Approvers` along with the Argon team. This can help to reduce the risk of being accessed by malicious users.

AFS-2 | Storage Manipulation in `view` function

Category	Severity	Location	Status
Language Specific	● Major	ArgonFreelancers.sol: 352~353	✓ Resolved

Description

There should not be any storage variable manipulation in the `view` function

Recommendation

We advise the client to consider changing `storage` into `memory` for `data` in L352 and use a local variable to store the value of `result`

Alleviation

The client heeded the advice and update the function by changing `storage` into `memory` for `data` in the commit `9d8b1a7555ea5ee33375eade8859147f00a39564`

AFS-3 | Division Before Multiplication

Category	Severity	Location	Status
Mathematical Operations	● Informational	ArgonFreelancers.sol: 358	Partially Resolved

Description

Mathematical operations in the aforementioned function perform divisions before multiplications. Performing multiplication before division can sometimes avoid loss of precision.

Recommendation

We recommend applying multiplications before divisions if integer overflow would not happen in functions.

Alleviation

The client heeded the advice and rearranged the order of mathematical operations to make sure that the multiplication operation happens before the division operation in the commit `9d8b1a7555ea5ee33375eade8859147f00a39564`. However, the value of multiplier is updated from `1000` in the commit `ca548a8de3f11cc4a3a10ad6bae86fe420947dd8` into `100` in the commit `9d8b1a7555ea5ee33375eade8859147f00a39564`. This is notified to the client.

AFS-4 | Centralized Risk

Category	Severity	Location	Status
Control Flow	● Critical	ArgonFreelancers.sol: 242, 252, 258	Partially Resolved

Description

`ArgonTokenDeployer` is an important role that can only be set by contract deployer in the contract. The ArgonTokenDeployer address can operate on following functions:

- `changeApproverMinArgonLimit()`
- `sendArgonTokenDeployer()`

Recommendation

We advise the client to carefully manage the ArgonTokenDeployer account's private key and avoid any potential risks of being hacked. We also advise the client to adopt Multisig, Timelock, and/or DAO in the project to manage sensitive role, `ArgonTokenDeployer` in this case.

Alleviation

The client adopted a centralized role `Approver` to restrict the access to the aforementioned sensitive functions in commit `9d8b1a7555ea5ee33375eade8859147f00a39564`. The client team also invited a community member whose telegraph handle is `@elips7` to manage the private keys of `Approvers` along with the Argon team. This can help to reduce the risk of being accessed by malicious users.

AFS-5 | Missing Emit Events

Category	Severity	Location	Status
Logical Issue	● Informational	ArgonFreelancers.sol: 252, 258	✓ Resolved

Description

Function that affect the status of sensitive variables should be able to emit events as notifications to customers:

- `changeApproverMinArgonLimit()`
- `sendArgonTokenDeployer()`

Recommendation

Consider adding events for sensitive actions, and emit it in the function like below:

```
event ChangeApproverMinArgonLimit(address indexed user, uint _approverMinArgonLimit);

function changeApproverMinArgonLimit(uint _value) public {
    require(msg.sender == ArgonTokenDeployer);
    approverMinArgonLimit = _value;
    emit ChangeApproverMinArgonLimit(msg.sender, approverMinArgonLimit);
}
```

Alleviation

The client heeded the advice and added emit events to the aforementioned functions in the commit `9d8b1a7555ea5ee33375eade8859147f00a39564`

AFS-6 | Lack of Sanity Check

Category	Severity	Location	Status
Volatile Code	● Minor	ArgonFreelancers.sol: 651, 646	Resolved

Description

The assigned value to `_employerAddress`, `_t` should be verified as non zero value to prevent being mistakenly assigned as `address(0)` in the constructor of the contract.

Recommendation

We advise client to check that the addresses are not zero by adding following checks in the constructor of the contract as well as functions.

```
require(_employerAddress != address(0), "employerAddress's address must not be address(0)");
require(_t != address(0), "deployedFromContract contract's address must not be address(0)");
```

Alleviation

The client heeded the advice and added the aforementioned input checks to the constructor of the contract in the commit `9d8b1a7555ea5ee33375eade8859147f00a39564`

AFS-7 | Centralized Risk

Category	Severity	Location	Status
Control Flow	● Critical	ArgonFreelancers.sol: 626, 809	Partially Resolved

Description

Address `tokenContractAddress` can be updated to any address by calling `selectOfferWithToken()` by `employerAddress`, which can lead to the unknown result to the invocation of `transfer()` in following functions:

- `employerReceiveFile()`
- `confirmApprover()`
- `cancelApprover()`
- `sendDeadline()`

Recommendation

We advise the client to carefully manage `employerAddress`s private key and avoid any potential risk of being hacked. We also advise the client to adopt Timelock, Multisig, and/or DAO in the project to manage sensitive role access

Alleviation

The client adopted a centralized role `Approver` to restrict the access to the aforementioned sensitive functions in commit `9d8b1a7555ea5ee33375eade8859147f00a39564`. The client team also invited a community member whose telegraph handle is `@elips7` to manage the private keys of `Approvers` along with the Argon team. This can help to reduce the risk of being accessed by malicious users.

AFS-8 | Centralized Risk

Category	Severity	Location	Status
Control Flow	● Critical	ArgonFreelancers.sol: 620, 576	Partially Resolved

Description

Address `approverAddress` can be updated to any address by calling `selectOffer()` or `selectOfferWithToken()` by `employerAddress`, which can lead to sending tokens to any address through `sendApproverArgonCoin()` function.

Recommendation

We advise the client to carefully manage `approverAddress`'s private key and avoid any potential risks of being hacked. We also advise the client to adopt Multisig, Timelock, and/or DAO in the project to manage sensitive role access.

Alleviation

The client adopted a centralized role `Approver` to restrict the access to the aforementioned sensitive functions in commit `9d8b1a7555ea5ee33375eade8859147f00a39564`. The client team also invited a community member whose telegraph handle is `@elips7` to manage the private keys of `Approvers` along with the Argon team. This can help to reduce the risk of being accessed by malicious users.

AFS-9 | Centralized Risk

Category	Severity	Location	Status
Control Flow	● Critical	ArgonFreelancers.sol: 614	Partially Resolved

Description

Address `freelancerAddress` can be updated to any address by calling `selectOffer()` or `selectOfferWithToken()` by `employerAddress`, which can lead to the unknown result in calling `transfer()` in `employerReceiveFile()` and `confirmApprover` function.

Recommendation

We advise the client to carefully manage `freelancerAddress`'s private key and avoid any potential risks of being hacked. We also advise the client to adopt Multisig, Timelock, and/or DAO in the project to manage sensitive role access.

Alleviation

The client adopted a centralized role `Approver` to restrict the access to the aforementioned sensitive functions in commit `9d8b1a7555ea5ee33375eade8859147f00a39564`. The client team also invited a community member whose telegraph handle is `@elips7` to manage the private keys of `Approvers` along with the Argon team. This can help to reduce the risk of being accessed by malicious users.

AFS-10 | Potential Reentrancy Vulnerability

Category	Severity	Location	Status
Logical Issue	● Critical	ArgonFreelancers.sol: 869(WorkContract), 729(WorkContract), 724(WorkContract), 699(WorkContract), 576(MainContract)	✓ Resolved

Description

Following functions are exposed to reentrancy attacks where the attacker is able to reentrant functions multiple times in a single transaction.

- `sendDeadline()`
- `updateOffer()`
- `createOffer()`
- `deleteOffer()`
- `sendApproverArgonCoin()`

Recommendation

We advise client to consider to adopt following `nonReentrant` modifier on above mentioned functions:

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/ReentrancyGuard.sol

Alleviation

The client heeded the advice and adopted the `nonReentrant` modifier to prevent any potential reentrancy issues in the commit `9d8b1a7555ea5ee33375eade8859147f00a39564`

AFS-11 | Token Minted to Centralized Address

Category	Severity	Location	Status
Logical Issue	● Critical	ArgonFreelancers.sol: 94	ⓘ Acknowledged

Description

The `_totalSupply` amount of token is minted to a centralized address, which is `0x8CB5b6B0A475e760ed0610AD9cF8403Ec050bc8A`. This may raise community's concerns about the centralized issue.

Recommendation

We advise the client to carefully manage the 0x8CB5b6B0A475e760ed0610AD9cF8403Ec050bc8A account's private key and avoid any potential risks of being hacked. We also advise the client to adopt Multisig, Timelock, and/or DAO in the project to manage this specific account in this case.

Alleviation

N/A

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

CERTIK