Aelin DAO - AELIP-50-52
Security Assessment Findings Report

Date: June 27, 2023
Project: AEL
Version 0.2

Contents

DRAFT

1 Confidentiality statement

This document is the exclusive property of Aelin DAO and Sub7 Security. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both Aelin DAO and Sub7 Security.

2 Disclaimer

A smart contract security audit is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. Sub7 Security prioritized the assessment to identify the weakest security controls an attacker would exploit. Sub7 Security recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls

DRAFT

3 About Sub7

Sub7 is a Web3 Security Agency, offering Smart Contract Auditing Services for blockchain-based projects in the DeFi, Web3 and Metaverse space.

Learn more about us at https://sub7.xyz

4 Project Overview

Aelin is a permissionless multi-chain protocol for capital raises and OTC deals. It allows anyone to raise capital for their project or an OTC deal and to seek out investors.

The Aelin protocol is designed to enable protocols and sponsors to raise money from their communities in two primary ways:

1. *Aelin Direct Deals* are an excellent option for new and established protocols looking to raise capital from investors at pre-established deal terms. Aelin Deals allow for customizable features, such as a defined vesting period and a vesting cliff. Additionally, these deals can also include NFT-gated features, which can be a great way to restrict access to particular NFT holders. Overall, Aelin Deals are a great way to raise capital for new protocols.
2. *Aelin Pools* are best suited for protocols that don't have set deal terms yet, but are gauging investor interest. This type of pool is ideal for sponsors using Aelin to source a future deal, as it allows them to collect data and feedback from investors without committing to anything. Aelin Pools are most similar to SPACs in terms of their purpose and structure.

Aelin is proud to offer many custom features to help protocols and sponsors raise capital.

1. *NFT Gated Pools* - Protocols can decide to gate off their token raise to holders of specific NFT collections. This allows protocols and DAOs to target specific groups of investors, which can improve the process of building and growing their community. Protocols can set maximums purchase amounts for each NFT holder.
2. *Custom Allow List* (100k addresses+) - Protocols may want to provide rewards to a large number of addresses with different allowance amounts to access their upfront Aelin deal. Protocols or large holders can use an off-chain script to generate a CSV file with several hundred thousand addresses and custom amounts based on each wallet's activity level onchain or any logic the protocol desires.
3. *Customizable Vesting Options* - Protocols that use Aelin can customize their vesting options (e.g. cliff, vesting time, etc.) to ensure that their project's objectives are aligned with their investors' incentives.

4. *Predetermined Deal Terms* - Aelin provides protocols with the ability to set a specific, fixed exchange rate for investors, without the worry of fluctuating market conditions affecting the terms of the deal. This makes Aelin more like a large on-chain OTC transaction, which is preferable for protocols that are targeting specific valuations.

DRAFT

5 Executive Summary

Sub7 Security has been engaged to what is formally referred to as a Security Audit of Solidity Smart Contracts, a combination of automated and manual assessments in search for vulnerabilities, bugs, unintended outputs, among others inside deployed Smart Contracts.

The goal of such a Security Audit is to assess project code (with any associated specification, and documentation) and provide our clients with a report of potential security-related issues that should be addressed to improve security posture, decrease attack surface and mitigate risk.

2 (Two) Security Researchers/Consultants were engaged in this activity.

5.1 Scope

Assessment:

- https://github.com/AelinXYZ/aelin/pull/161
- https://github.com/AelinXYZ/aelin/pull/171
- https://github.com/AelinXYZ/aelin/pull/172

Verification: https://github.com/AelinXYZ/aelin/commit/0d94add991cef4cf85a02c5cbfd83c7148b3e51d/contracts

5.2 Timeline

Assessment: 22 May 2023 to 26 May 2023

Verification: 14 June 2023

DRAFT

5.3 Summary of Findings Identified

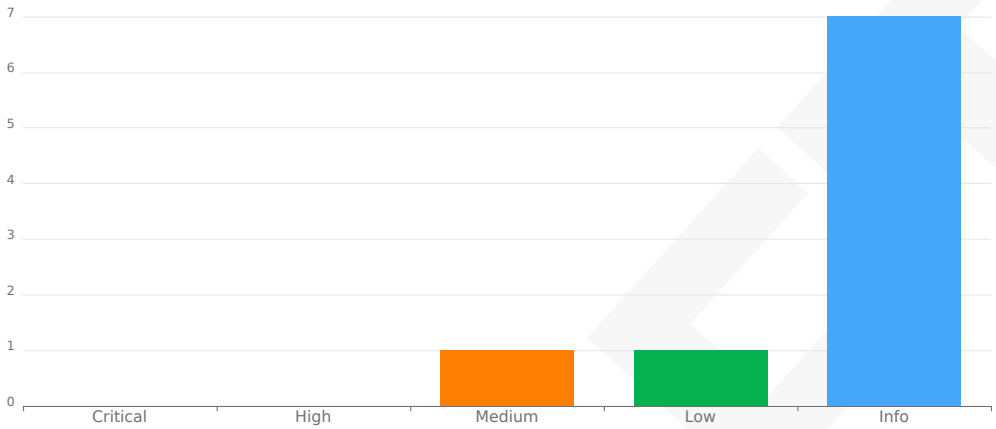

Figure 1: Executive Summary

 # 1 Medium NFTs with unlimited purchase amount should not be blacklisted once used – ***Fixed***

 # 2 Low EIP-721 does not enforce that NFT's tokenIds follow any kind of pattern – ***Acknowledged***

 # 3 Informational Result from external call can be cached outside of for loop – ***Fixed***

 # 4 Informational Can break for loop once maxPurchaseTokenAmount is set to type(uint256).max – ***Fixed***

 # 5 Informational Storage variables can be marked immutable – ***Fixed***

 # 6 Informational Misleading string error message – ***Fixed***

 # 7 Informational ERC20 tokens transfers return value unchecked – ***Acknowledged***

 # 8 Informational Unnecessary operations – ***Fixed***

 # 9 Informational `idRanges` may overlap in initialization – ***Fixed***

5.4 Methodology

Sub7 Security follows a phased assessment approach that includes thorough application profiling, threat analysis, dynamic and manual testing. Security Auditors/Consultants utilize automated security tools, custom scripts, simulation apps, manual testing and validation techniques to scan for, enumerate and uncover findings with potential security risks.

As part of the process of reviewing solidity code, each contract is checked against lists of known smart contract vulnerabilities, which is crafted from various sources like SWC Registry , DeFi threat and previous audit reports.

The assessment included (but was not limited to) reviews on the following attack vectors:

Oracle Attacks | Flash Loan Attacks | Governance Attacks | Access Control Checks on Critical Function | Account Existence Check for low level calls | Arithmetic Over/Under Flows | Assert Violation | Authorization through tx.origin | Bad Source of Randomness | Block Timestamp manipulation | Bypass Contract Size Check | Code With No Effects | Delegatecall | Delegatecall to Untrusted Callee | DoS with (Unexpected) revert | DoS with Block Gas Limit | Logical Issues | Entropy Illusion | Function Selector Abuse | Floating Point and Numerical | Precision | Floating Pragma | Forcibly Sending Ether to a Contract | Function Default Visibility | Hash Collisions With Multiple Variable Length Arguments | Improper Array Deletion | Incorrect interface | Insufficient gas griefing | Unsafe Ownership Transfer | Loop through long arrays | Message call with hardcoded gas amount | Outdated Compiler Version | Precision Loss in Calculations | Price Manipulation | Hiding Malicious Code with External Contract | Public burn() function | Race Conditions / Front Running | Re-entrancy | Requirement Violation | Right-To-Left-Override control character (U+202E) | Shadowing State Variables | Short Address/Parameter Attack | Signature Malleability | Signature Replay Attacks | State Variable Default Visibility | Transaction Order Dependence | Typographical Error | Unchecked Call Return Value | Unencrypted Private Data On-Chain | Unexpected Ether balance | Uninitialized Storage Pointer | Unprotected Ether Withdrawal | Unprotected SELFDESTRUCT Instruction | Unprotected Upgrades | Unused Variable | Use of Deprecated Solidity Functions | Write to Arbitrary Storage Location | Wrong inheritance | Many more…

6 Findings and Risk Analysis

6.1 NFTs with unlimited purchase amount should not be blacklisted once used

Severity: Medium
Status: Fixed

Description

When gating NFT purchases, 'NftCollectionRules.purchaseAmount' is used to determine how many tokens a user can purchase per NFT. If this amount is set to 0, it means that each NFT can purchase an unlimited amount of tokens. However, in 'purchasePoolTokensWithNft', we observe that each NFT can only be used once, even if the collection has unlimited purchase rule.

Users may not be aware of this, as they first purchase a small amount of pool tokens using their "unlimited amount NFT", only to realise that the second call with the same NFT does not work as their NFT can no longer be used, despite it being supposedly allowing unlimited purchase.

Unlimited purchase is only valid to the point in which all tokens are purchased in the same sitting. We believe this should not be the case as an unlimited purchase rule NFT should be usable even across different calls.

Location

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinPool.sol#L253-L254

Recommendation

Consider blacklisting an NFT after it has been used, only for collections that are not unlimited purchases.

```
function purchasePoolTokensWithNft{
    ...
    for(uint256 i; i < nftPurchaseListLength; ++i){
        ...
        for (uint256 j; j < tokenIdsLength; ++j){
            if((NftCheck.supports721(collectionAddress)){
                ...
+               if(nftCollectionRules.purchaseAmount != 0){
+                   nftId\[collectionAddress\]\[tokenIds\[j\]\] = true;
+               }
-               nftId\[collectionAddress\]\[tokenIds\[j\]\] = true;
                emit BlacklistNFT(collectionAddress, tokenIds\[j\]);
                ...
            }
            ...
```

```
16            }
17            ...
18        }
19        ...
20    }
```

Client Comments

None

6.2 EIP-721 does not enforce that NFT's tokenIds follow any kind of pattern

Severity: Low
Status: Acknowledged

Description

Generally when working with NFTs, we should not assume that the tokenIds follow any kind of pattern, such as the incremental pattern, or that token starts counting from 'tokenId = 0'. From the EIP-721 docs https://eips.ethereum.org/EIPS/eip-721,

> While some ERC-721 smart contracts may find it convenient to start with ID 0 and simply increment by one for each new NFT, callers SHALL NOT assume that ID numbers have any specific pattern to them, and MUST treat the ID as a "black box".

The design behind NFT gating via token ranges seems to premise on the idea that 'tokenIds' follow a certain pattern, which is incorrect. The current design limits 'nftCollectionRules' to 10 ranges. 10 ranges appear to be sufficient if the NFT used has the incremental pattern. The protocol might not be able to support an NFT that does not implement such pattern, as each "range" may be sufficient to cover only 1 'tokenIds'. Note that the span of acceptable 'tokenIds' is 'uint256'.

While the current implementation does not cause any loss to the protocol, it can potentially affect the kind of NFTs that protocol can support, hence we believe low severity to be appropriate.

Location

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinPool.sol#LL249C18-L249C18

Recommendation

There may not be a way to gate the variety of NFTs in a general manner using their 'tokenIds'. Giving support to only NFTs which follows the incremental pattern may be the most rationale approach here. We are reporting this issue as Aelin may find the report useful.

Client Comments

None

6.3 Result from external call can be cached outside of for loop

Severity: Informational
Status: Fixed

Description

`NftCheck.supports721(collectionAddress)` is called on each iteration in the nested for loop in AelinPool.purchaseDealTokensWithNft and AelinNFTGating.purchaseDealTokensWithNft. Since `collectionAddress` is the same of each iteration, the result can be cached out side of the for loop to save gas.

Location

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinPool.sol#L244

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/libraries/AelinNftGating.sol#L138

Recommendation

Consider caching `NftCheck.supports721(collectionAddress)` outside of the nested for loop in `purchaseDealTokensWithNft`.

Client Comments

None

6.4 Can break for loop once maxPurchaseTokenAmount is set to type(uint256).max

Severity: Informational
Status: Fixed

Description

In AelinPool.purchaseDealTokensWithNft and AelinNFTGating.purchaseDealTokensWithNft there are 3 places where `maxPurchaseTokenAmount` is set to `type(uint256).max`. However, this happens in a for loop and the loop does not `break` there meaning that redundant iterations might be made wasting redundant gas for the user.

Location

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinPool.sol#L266-L283

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/libraries/AelinNftGating.sol#L164-L177

Recommendation

Consider adding a `break`; after `maxPurchaseTokenAmount` is set to `type(uint256).max`.

Client Comments

None

6.5 Storage variables can be marked immutable

Severity: Informational
Status: Fixed

Description

The `aelinToken`, `oldAelinToken` and `aelinTreasury` in AelinTokenSwapper are only set in the contract's constructor and therefore can be marked `immutable` to save gas (1x SLOAD opcode) each time they are used.

Location

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinTokenSwapper.sol#L17-L19

Recommendation

Consider marking the aforementioned storage variables as immutable.

Client Comments

None

6.6 Misleading string error message

Severity: Informational
Status: Fixed

Description

In AelinVestingToken.transferVestingShare it is checked that the `_shareAmount_` *ot transfer is not greater than or _equal to* the whole vesting schedule amount. The revert message in the following check however says that the `\_shareAmount` is strictly greater than the `schedule.share`:

```
require(\_shareAmount < schedule.share, "amout gt than current share");
```

Location

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinVestingToken.sol#L28

Recommendation

Consider either using <= instead of < in the `require` condition or change `gt` to `gte` in the error message.

Client Comments

None

6.7 ERC20 tokens transfers return value unchecked

Severity: Informational
Status: Acknowledged

Description

According to EIP20: “Callers MUST handle **`false`** from `returns (bool success)`. Callers MUST NOT assume that **`false`** is never returned!“. Although AelinTokenSwapper only works with known tokens, it is recommended to always follow the EIP specifications.

Location

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinTokenSwapper.sol#L33

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinTokenSwapper.sol#L42

Recommendation

Consider using the SafeERC20 library from OpenZeppelin.

Client Comments

None

6.8 Unnecessary operations

Severity: Informational
Status: Fixed

Description

In AelinNFTGating.initialize the `\_data.hasNftList` flag is set to `true` when the Aelin deal is NFT gated. The flag is also set to `false` in case there are no provided `\_nftCollectionRules`. The latter is redundant since `false` is the default value of `hasNftList`.

In Aelin.sol the inherited Ownable contract contructor is explicitely called. This is redundant as it gets called automatically.

In AelinTokenSwapper.depositTokens and .swap, the contract checks if the callers have enough balance before performing a .transferFrom. These checks are redundant and a waste of gas as the ERC20 token contracts already take care of such validation.

In AelinTokenSwapper the `receiver` indexed parameter of the `TokenDeposited` event is redundant as it always has the value of `address(this)` (the AelinTokenSwapper contract address).

In AelinPool.initialize and AelinNFTGating.initialize it is checked twice whether the first `colletionAddress` from the `\_nftCollectionRules` array supports the given NFT standard (ERC721 or ERC1155).

Location

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/libraries/AelinNftGating.sol#L89-L91

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/Aelin.sol#L16

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinTokenSwapper.sol#L32

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinTokenSwapper.sol#L41

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinTokenSwapper.sol#L48

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinPool.sol#L127-L129

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/Aelin.sol#L30

Recommendation

Consider removing the following lines in AelinNFTGating.initialize:

```
  else {
    \_data.hasNftList = false;
}
```

Consider removing the call to Ownable() in Aelin.sol:

```
constructor(address \_initialHolder) ERC20("Aelin", "AELIN") Ownable() {
```

Consider removing the following checks in AelinTokenSwapper.depositTokens and .swap:

```
function depositTokens() external {
    ...
    if (IERC20(aelinToken).balanceOf(msg.sender) < TOKEN\_SUPPLY) revert BalanceTooLow();

function swap(uint256 \_amount) external {
    ...
    if (IERC20(oldAelinToken).balanceOf(msg.sender) < \_amount) revert BalanceTooLow();
```

Consider removing the `receiver` parameter from the `TokenDeposited` event in AelinTokenSwapper:

```
event TokenDeposited(address indexed sender, address indexed receiver, uint256 amount);
```

Consider performing the following checks in AelinPool.initialize and AelinNFTGating.initialize only `if (i != 0):`

```
require(NftCheck.supports721(\_nftCollectionRules\[i\].collectionAddress), "can only
    contain 721");

require(NftCheck.supports1155(\_nftCollectionRules\[i\].collectionAddress), "can only
    contain 1155");
```

Consider removint the following check in Aelin.setAuthorizedMinter:

```
if (\_minter == address(0)) revert InvalidAddress();
```

Client Comments

None

6.9 idRanges may overlap in initialization

Severity: Informational
Status: Fixed

Description

When initializing ‘nftCollectionRules.idRanges‘, we sanitise the input of idRanges with ‘nftCollectionRules.idRanges.begin <= ‘nftCollectionRules.idRanges.end‘. This is a neccessary check due to the function ‘isTokenIdInRange()‘. We can consider further sanitising the input to prevent overlapping ranges.

Location

https://github.com/AelinXYZ/aelin/blob/d6b84f249973388656c9a3d33277c228736ed1fe/contracts/AelinPool.sol#L135-L138

Recommendation

Consider further sanitising input such that ranges follow an order. For eg, [1,10], [12,20], [20,50] is valid, but [1,10], [5,20] is not valid and [12,20], [1,10] is not valid.

We can do this by checking that the head of a range is bigger than the tail of the previous range, i.e. ‘nftCollectionRules[i].idRanges[j].begin >= nftCollectionRules[i].idRanges[j-1].end‘.

Client Comments

None

DRAFT