Algem
Diamond Refactoring

Smart Contract Security Assessment

January 26, 2022

ABSTRACT

Dedaub was commissioned to perform a security audit of the Algem protocol.

The contracts of the repository ilkatel/liquidStaking_audit of the Algem protocol has been audited before, on November 2022 up to commit `0d1aa2e799ee03b116283d4a7a03ab5b9df57a48`. The corresponding report can be found here.

This report focuses exclusively on the Diamond Refactoring of the `LiquidStaking` contract of the Algem protocol. The commit that introduced this refactoring is the `5a357c14c6ac1f1619c0e0e6d6063e7264d578f7`. However, this commit contained a partial implementation of the H1 fix of the main report which was later reverted since it wasn't finalized yet. Thus, the commit on which this audit was based is `8046a411b637c2747160526aaa9a6b0d83ee5607` which is the one after the reversion of the H1 fix.

This report also covers the commits beyond the one on which the audit was based (`8046a411b637c2747160526aaa9a6b0d83ee5607`), which include the fixes of the issues originally reported here as part of the audit. After receiving the new implementations, we re-audited the code with a focus on the changes that resolve the issues. As a result, this report covers the codebase up to the commit `bc50b9aa018e2f5ced49c314f2c6ce9356f8c6e6`.

Issues from the November 2022 main report that were “Dismissed”, and those which were “Acknowledged” but were not fixed or implemented at the time of writing are not reported here again.

The audited contracts are the following:

```
contracts/audit/
├── ArthswapAdapter.sol
├── NFTDistributor.sol
├── ZenlinkAdapter.sol
│
├── LiquidStaking/
│   ├── LiquidStaking.sol
│   ├── LiquidStakingMain.sol
│   ├── LiquidStakingManager.sol
│   ├── LiquidStakingMigration.sol
│   ├── LiquidStakingMisc.sol
│   └── LiquidStakingStorage.sol
│
└── interfaces/
    ├── ILiquidStaking.sol
    ├── ILiquidStakingManager.sol
    └── IZenlinkPair.sol
```

SETTING & CAVEATS

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than the regular use of the protocol. Functional correctness (i.e. issues in "regular use") is a secondary consideration. Typically it can only be covered if we are provided with unambiguous (i.e. full-detail) specifications of what is the expected, correct behaviour. In terms of functional correctness, we often trusted the code's calculations and interactions, in the absence of any other specification. Functional correctness relative to low-level calculations (including units, scaling and quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues affecting the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third-party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third-party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: • User or system funds can be lost when third-party systems misbehave. • DoS, under specific conditions. • Part of the functionality becomes unusable due to a programming error.
LOW	Examples: • Breaking important system invariants but without apparent consequences. • Buggy functionality for trusted users where a workaround exists. • Security issues which may manifest when the system evolves.

Issue resolution includes "dismissed" or "acknowledged" but no action taken, by the client, or "resolved", per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

<table>
<tr><th>ID</th><th>Description</th><th>STATUS</th></tr>
<tr><td>H1</td><td>Lack of access control in LiquidStakingManager functions</td><td><b>RESOLVED</b></td></tr>
<tr><td colspan="3">The pause() and unpause() functions of the LiquidStakingManager contract are not access controlled and can be used by an attacker to pause() and unpause() this contract at will.</td></tr>
</table>

MEDIUM SEVERITY:

<table>
<tr><th>ID</th><th>Description</th><th>STATUS</th></tr>
<tr><td>M1</td><td>Missing roles in LiquidStakingManager contract initialisation</td><td><b>RESOLVED</b></td></tr>
<tr><td colspan="3">The LiquidStakingManager contract has functions addManager() and removeManager() which require the DEFAULT_ADMIN_ROLE to execute. However, this role is not granted to the caller of the initialize() function, making them unusable.</td></tr>
</table>

LOW SEVERITY:

ID	Description	STATUS
L1	Mistakenly reverted fixes and other compile errors	**RESOLVED**

LiquidStaking.sol

- **LoC:22**: The initialize() can be made external.
- **LoC:30**: The setMinStakeAmount() is not visible by LiquidStaking.sol.
- **LoC:40-43**: The initialize() function uses variable _dntUtil which used to be the 2nd argument of the constructor before the Diamond refactoring. However, now this argument was renamed to _utilName, but the code inside the function wasn't updated accordingly.

LiquidStakingMain.sol

- **LoC:194**: The fix which removed the "()" from the use of the updateAll modifier in the withdraw() function was reverted.
- **LoC:232**: The fix which renamed the lastE**t**aTotalBalance to the correct lastE**r**aTotalBalance was reverted.
- **LoC:432**: The _updateUserBalanceInUtility() uses the distr1_5 variable which is undeclared.
- **LoC:493**: The fix which renamed the message of the require() that had a typo was reverted.

LiquidStakingMigration.sol

- **LoC:93**: The migrateStorage() function was made public instead of external with no apparent need for such change.

`NFTDistributor.sol`

- **`LoC:432`**: This line contains the fix of the A6 issue from the main report which was introduced at commit 26c4028. However, it misses the semicolon (`;`) at the end of the line.

`ArthswapAdapter.sol`

- The `WASTR` address was mistakenly deleted when the fix for H1 of the main audit report was reverted in commit 25e78a.

`ZenlinkAdapter.sol`

- **`LoC:209`**: The `pair.approve()` and the `pair.totalSupply()` functions couldn't be found in `IZenlinkPair.sol`.

L2	Incorrect implementation of `findMedian` function in `LiquidStakingMigration` contract.	**RESOLVED**

The function `findMedian()` in the `LiquidStakingMigration` contract does not implement the median calculation correctly when the array is already in sorted order and is of even length. In this case, the result needs to be averaged between the two middle values.

`LiquidStakingMigration::findMedian()`

```
function findMedian(uint[] memory _arr) private pure returns (uint mean) {
    uint[] memory arr = _arr;
    uint len = arr.length;
    bool swapped = false;
    for (uint i; i < len - 1; i++) {
        for (uint j; j < len - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swapped = true;
                uint s = arr[j + 1];
                arr[j + 1] = arr[j];
                arr[j] = s;
            }
        }

        // Dedaub: The middle values need to be averaged
        //         here as well when len % 2 == 0
        if (!swapped) {
            return arr[len/2];
        }
    }
    if (len % 2 == 0)
        return (arr[len/2] + arr[len/2 - 1])/2;
    return arr[len/2];
}
```

L3	Deleting a selector will never succeed and will cause most of the `LiquidStakingManager` functions to revert	**RESOLVED**

The `deleteSelector()` function of the `LiquidStakingManager` contract, assigns the `address(0)` to the `selectorToAddress` mapping for the provided selector and then calls the private `_deleteSelector()` which erases the rest of the data related to this particular selector.

However, this function needs to retrieve the associated address with this selector in order to delete the rest data from the side of the address. Thus, since the address was deleted earlier, the function gets the `address(0)` as the to-be-deleted address and reverts due to underflow error when calculating the `lastIndex` of the list of the selectors.

As a result, the functions changeSelector() and changeSelector() that call deleteSelector() will eventually revert as well.

LiquidStakingManager::deleteSelector()

```
function deleteSelector(bytes4 selector) public onlyRole(MANAGER) {
    require(selectorToAddress[selector] != address(0),
        "The selector was not set");

    // Dedaub: Here the address of the selector is deleted before the call
    //         to _deleteSelector() which needs the original value
    selectorToAddress[selector] = address(0);

    _deleteSelector(selector);
}
```

LiquidStakingManager::_deleteSelector()

```
function _deleteSelector(bytes4 selector) private {

    // Dedaub: Here the function retrieves the address of the selector
    //         for deleting its related data, but it has already been
    //         zeroed at that point
    address addressFromSelector = selectorToAddress[selector];

    uint256 index = selectorIndex[addressFromSelector][selector];

    // Dedaub: Here the execution will revert due to an underflow error
    //         since the length of the array of the selectors for the
    //         address(0) is zero
    uint256 lastIndex = addressSelectors[addressFromSelector].length - 1;

    bytes4 lastSelector =
        addressSelectors[addressFromSelector][lastIndex];

    addressSelectors[addressFromSelector][index] =
        addressSelectors[addressFromSelector][lastIndex];
    selectorIndex[addressFromSelector][lastSelector] = index;

    addressSelectors[addressFromSelector].pop();

    if (addressSelectors[addressFromSelector].length == 0)
        _deleteAddress(addressFromSelector);
}
```

OTHER / ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

ID	Description	STATUS
A1	Warning for future maintainability regarding the current implementation of the Diamond pattern	**ACKNOWLEDGED**

The current implementation of the Diamond pattern is not the standard implementation since it has some differences.

We raise here a warning for future maintainability for the reason that no use of hashed storage layout is made for ensuring no storage collisions.

However, in the current implementation the facets don't have any storage variable of their own and they all inherit uniformly. This means that this issue doesn't currently occur, but we raise this warning for future upgrades or facet additions that could introduce some storage variables or different inherited contract order which would cause a storage clash.

A2	Possible misplaced `setting()` function in `LiquidStaking` contract	**RESOLVED**
The `LiquidStaking` contract is the proxy contract which provides the entry point into the `Diamond`. It is called by other contracts and performs the required delegate calls to the other facet contracts. However, it also contains a `setting()` function which looks unrelated to the rest of its functionality. Possibly this may need to be moved to a facet instead.		
A3	Redundant contract variable in `LiquidStakingManager` contract	**RESOLVED**
The `LiquidStakingManager` contract has a contract variable named `diamondAddress` which is never initialized nor used.		
A4	No contract implements `IPartnerHandler` interface	**RESOLVED**
The `LiquidStakingMigration` contract has a function `getUserLpTokens()` which uses the `IPartnerHandler` interface when calling `calc()`. However, no contract in the repository implements `IPartnerHandler`. Possibly the adapters need to implement this interface.		
A5	Compiler bugs	**INFO**
The code can be compiled with Solidity 0.8.4. Version 0.8.4, in particular, has some known bugs, which we do not believe affect the correctness of the contracts.		

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Watchdog.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.