Protocol Audit Report

Azriel

May 8, 2025

Protocol Audit Report

Version 1.0

Cyfrin.io

May 11, 2025

Protocol Audit Report

Azriel

May 8, 2025

Prepared by: Azriel

Table of Contents

* I-3: Modifiers invoked only once can be shoe-horned into the function
* I-4: Empty Block
* I-5: Unused Custom Error
* I-6: Boolean equality is not required.

Protocol Summary

Simulates a school with students, teachers and principals, using upgradeable contracts to simulate "graduation" for students that meet the cut-off score.

Disclaimer

The Azriel(me) team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

In Scope:

```
src/
--LevelOne.sol - Contract that contains the logic for the first school term
--LevelTwo.sol - Contract for the second school term
```

Compatibilities:

Chain: EVM Compatible

Token: USDC

Roles

Principal: In charge of hiring/firing teachers, starting the school session, and upgrading the system at the end of the school session. Will receive 5% of all school fees paid as his wages. can also expel students who break rules.
Teachers: In charge of giving reviews to students at the end of each week. Will share in 35% of all school fees paid as their wages.
Student: Will pay a school fee when enrolling in Hawk High School. Will get a review each week. If they fail to meet the cutoff score at the end of a school session, they will be not graduated to the next level when the Principal upgrades the system.

Executive Summary

For this audit, I used about 6h in total spread across 2 days to conduct the audit and complete the report. I made use of tools like Slither and Aderyn to first conduct static analysis, in addition to manual analysis. Additionally, I used cloc and Solidity Metrics to aid me in the initial scoping phase and understanding of the code base better, although there are only 2 files in scope.

Issues found

Severity	Number of Issues Found
High	7
Medium	0
Low	1
Info	6

Findings

High Findings

[H-1] `LevelOne:graduateAndUpgrade` Does Not Perform Actual Upgrade Or Initialization

Description: `LevelOne:graduateAndUpgrade` currently calls `_authorizeUpgrade(_levelTwo)`, however this makes an internal call to an empty function. This call does not actually upgrade the implementation contract to the `_levelTwo` contract nor call the `LevelTwo:graduate` function to reinitialize the proxy.

Impact: This results in the protocol never being upgraded properly and will always read from the same implementation contract.

Proof of Concept: By adding the following test case, we can confirm that the implementation address stored within the proxy contract does not change even when `LevelOne:graduateAndUpgrade` is called.

```
    function test_upgrade_did_not_upgrade() public schoolInSession {
        bytes32 initialImplementation = vm.load(proxyAddress, ERC1967Utils.IMPLEMENTATION_SL
        address initialImplementationAddress = address(uint160(uint256(initialImplementatio

        levelTwoImplementation = new LevelTwo();
        levelTwoImplementationAddress = address(levelTwoImplementation);

        bytes memory data = abi.encodeCall(LevelTwo.graduate, ());

        vm.prank(principal);
        levelOneProxy.graduateAndUpgrade(levelTwoImplementationAddress, data);

        bytes32 newImplementation = vm.load(proxyAddress, ERC1967Utils.IMPLEMENTATION_SLOT);
        address newImplementationAddress = address(uint160(uint256(newImplementation)));

        assertEq(initialImplementationAddress, newImplementationAddress);
    }
```

Recommended Mitigation: Use `UUPSUpgradeable:upgradeToAndCall` function instead and pass in the data to initialize the proxy with the `_levelTwo` address.

```
--    function graduateAndUpgrade(address _levelTwo, bytes memory) public onlyPrincipal {
++    function graduateAndUpgrade(address _levelTwo, bytes memory data) public onlyPrincipal
        if (_levelTwo == address(0)) {
            revert HH__ZeroAddress();
        }

        uint256 totalTeachers = listOfTeachers.length;

        // @audit-info: teachers should share the 35% right? this is doing each teacher is g
        uint256 payPerTeacher = (bursary * TEACHER_WAGE) / PRECISION;
        uint256 principalPay = (bursary * PRINCIPAL_WAGE) / PRECISION;

--        _authorizeUpgrade(_levelTwo);
++        upgradeToAndCall(_levelTwo, data);

        for (uint256 n = 0; n < totalTeachers; n++) {
            usdc.safeTransfer(listOfTeachers[n], payPerTeacher);
        }
```

```
        usdc.safeTransfer(principal, principalPay);
    }
```

[H-2] `LevelOne:graduateAndUpgrade` Can Be Called Before `sessionEnd` Reached

Description: When the `principal` calls `LevelOne:graduateAndUpgrade`, there are no checks to ensure that `sessionEnd` is reached, allowing upgrade to be called at any time.

Impact: `LevelOne:graduateAndUpgrade` can be called even during the school session, violating the invariant which states that "System upgrade cannot take place unless school's `sessionEnd` has reached"

Proof of Concept: The following test passes when added to the test suite

```
    function test_upgrade_without_sessionend() public schoolInSession {
        levelTwoImplementation = new LevelTwo();
        levelTwoImplementationAddress = address(levelTwoImplementation);

        bytes memory data = abi.encodeCall(LevelTwo.graduate, ());

        assertLt(block.timestamp, levelOneProxy.getSessionEnd()); // assert that current ti
        assertTrue(levelOneProxy.getSessionStatus()); // asserts that school session is sti

        vm.prank(principal);
        levelOneProxy.graduateAndUpgrade(levelTwoImplementationAddress, data);
    }
```

Recommended Mitigation: We can ensure this invariant holds by adding the require statement below

```
    function graduateAndUpgrade(address _levelTwo, bytes memory) public onlyPrincipal {
        if (_levelTwo == address(0)) {
            revert HH__ZeroAddress();
        }

++        require(block.timestamp >= sessionEnd, "Session has not ended yet");

        uint256 totalTeachers = listOfTeachers.length;

        // @audit-info: teachers should share the 35% right? this is doing each teacher is g
        uint256 payPerTeacher = (bursary * TEACHER_WAGE) / PRECISION;
        uint256 principalPay = (bursary * PRINCIPAL_WAGE) / PRECISION;

        _authorizeUpgrade(_levelTwo);

        // @audit-info: push vs pull? would it be possible for a revert to occur thus giving
```

```
        // possible if one of the addresses are blacklisted!
        for (uint256 n = 0; n < totalTeachers; n++) {
            usdc.safeTransfer(listOfTeachers[n], payPerTeacher);
        }

        usdc.safeTransfer(principal, principalPay);
    }
```

[H-3] `LevelOne:graduateAndUpgrade` Does Not Filter Students Below Cutoff

Description: Students who do not meet the cutoff score are still upgraded to `LevelTwo` as there are no checks to filter them out.

Impact: This violates the invariant "Any student who doesn't meet the `cutOffScore` should not be upgraded"

Proof of Concept:

```
    function test_students_below_cutoff_not_removed() public schoolInSession {
        vm.warp(block.timestamp + 1 weeks);
        vm.prank(alice);
        levelOneProxy.giveReview(harriet, false);

        assertEq(levelOneProxy.getTotalStudents(), 6); // 6 students present initially
        assertEq(levelOneProxy.studentScore(harriet), 90);
        assertEq(levelOneProxy.cutOffScore(), 100);
        assertLt(levelOneProxy.studentScore(harriet), levelOneProxy.cutOffScore()); // asse

        levelTwoImplementation = new LevelTwo();
        levelTwoImplementationAddress = address(levelTwoImplementation);

        bytes memory data = abi.encodeCall(LevelTwo.graduate, ());

        vm.prank(principal);
        levelOneProxy.graduateAndUpgrade(levelTwoImplementationAddress, data);

        LevelTwo levelTwoProxy = LevelTwo(proxyAddress);

        assertEq(levelTwoProxy.getTotalStudents(), 6); // all 6 students are still present
    }
```

Recommended Mitigation: There are many ways to include this logic, depending on the required logic. The following shows the most straight forward way which is to call `expel` the students that did not meet the cutoff score, then call upgrade. Other ways that do not use `expel` would also work if `expel` is not the desired outcome.

```
    function graduateAndUpgrade(address _levelTwo, bytes memory) public onlyPrincipal {
        if (_levelTwo == address(0)) {
            revert HH__ZeroAddress();
        }

++        for (uint256 n = 0; n < listOfStudents.length; n++) {
++            address student = listOfStudents[n];
++            if (studentScore[student] < cutOffScore) {
++                expel(student);
++            }
++        }

        uint256 totalTeachers = listOfTeachers.length;

        uint256 payPerTeacher = (bursary * TEACHER_WAGE) / PRECISION;
        uint256 principalPay = (bursary * PRINCIPAL_WAGE) / PRECISION;

        _authorizeUpgrade(_levelTwo);

        for (uint256 n = 0; n < totalTeachers; n++) {
            usdc.safeTransfer(listOfTeachers[n], payPerTeacher);
        }

        usdc.safeTransfer(principal, principalPay);
    }
```

[H-4] Payment Disrbursement When `LevelOne:graduateAndUpgrade` Called Is Incorrect

Description: When wages are paid out during the call to `LevelOne:graduateAndUpgrade`, the structure is supposed to follow the invariant:

```
* `principal` gets 5% of `bursary`
* `teachers` share of 35% of bursary
* remaining 60% should reflect in the bursary after upgrade
```

However this is not the case as teachers are each receiving 35% of the bursary, rather than collectively sharing 35%.

Impact: This violates the invariant given and would also cause the contract to run out of funds if there are 3 teachers or more.

Proof of Concept: The following test shows that the remaining funds within the contract is less than expected, which should be 60% of the initial bursary before wages are paid.

```
    function test_payment_structure_wrong() public schoolInSession {
        levelTwoImplementation = new LevelTwo();
```

```
        levelTwoImplementationAddress = address(levelTwoImplementation);

        bytes memory data = abi.encodeCall(LevelTwo.graduate, ());

        uint256 initialBursary = schoolFees * 6;
        assertEq(usdc.balanceOf(proxyAddress), initialBursary);

        vm.prank(principal);
        levelOneProxy.graduateAndUpgrade(levelTwoImplementationAddress, data);

        // 5% of 30k = 1500 -> principal
        // 35% of 30k = 10500 -> shared by teachers
        // 60% of 30k = 18000 -> remaining in bursary
        uint256 bursaryRemaining = 60 * initialBursary / 100;
        assertLt(usdc.balanceOf(proxyAddress), bursaryRemaining);
    }
```

Recommended Mitigation: The `payPerTeacher` variable should be further divided by the number of teachers in the school as shown below:

```
    function graduateAndUpgrade(address _levelTwo, bytes memory) public onlyPrincipal {
        if (_levelTwo == address(0)) {
            revert HH__ZeroAddress();
        }

        uint256 totalTeachers = listOfTeachers.length;

--        uint256 payPerTeacher = (bursary * TEACHER_WAGE) / PRECISION;
++        uint256 payPerTeacher = (bursary * TEACHER_WAGE) / (PRECISION * listOfTeachers.le
        uint256 principalPay = (bursary * PRINCIPAL_WAGE) / PRECISION;

        _authorizeUpgrade(_levelTwo);

        for (uint256 n = 0; n < totalTeachers; n++) {
            usdc.safeTransfer(listOfTeachers[n], payPerTeacher);
        }

        usdc.safeTransfer(principal, principalPay);
    }
```

[H-5] `LevelOne:graduateAndUpgrade` Lacks Review Count Check

Description: When the `principal` calls `LevelOne:graduateAndUpgrade`, there are no checks to ensure that all students have gotten 4 reviews.

Impact: This directly violates the invariant "Students must have gotten all reviews before system upgrade. System upgrade should not occur if any student

has not gotten 4 reviews (one for each week)"

Proof of Concept: Since `reviewCount` is a private mapping, I changed it to a public mapping temporarily so that I can access the count value. By adding the following test code, we have shown that upgrading is possible even when students have less than 4 (in this case 0) reviews.

```
    function test_upgrade_without_4_reviews() public schoolInSession {
        levelTwoImplementation = new LevelTwo();
        levelTwoImplementationAddress = address(levelTwoImplementation);

        bytes memory data = abi.encodeCall(LevelTwo.graduate, ());

        assertEq(levelOneProxy.reviewCount(harriet), 0);

        vm.prank(principal);
        levelOneProxy.graduateAndUpgrade(levelTwoImplementationAddress, data);
    }
```

Recommended Mitigation: By including the extra check below, we can ensure that all students have had 4 reviews before allowing the upgrade.

```
    function graduateAndUpgrade(address _levelTwo, bytes memory) public onlyPrincipal {
        if (_levelTwo == address(0)) {
            revert HH__ZeroAddress();
        }

++        for (uint256 n = 0; n < listOfStudents.length; n++) {
++            if (reviewCount[listOfStudents[n]] < 4) {
++                revert HH__NotEnoughReviews(); // adding new revert for this error
++            }
++        }

        uint256 totalTeachers = listOfTeachers.length;

        // @audit-info: teachers should share the 35% right? this is doing each teacher is g
        uint256 payPerTeacher = (bursary * TEACHER_WAGE) / PRECISION;
        uint256 principalPay = (bursary * PRINCIPAL_WAGE) / PRECISION;

        _authorizeUpgrade(_levelTwo);

        // @audit-info: push vs pull? would it be possible for a revert to occur thus giving
        // possible if one of the addresses are blacklisted!
        for (uint256 n = 0; n < totalTeachers; n++) {
            usdc.safeTransfer(listOfTeachers[n], payPerTeacher);
        }

        usdc.safeTransfer(principal, principalPay);
```

```
}
```

[H-6] `LevelOne:giveReview` Does Not Increment `reviewCount`

Description: When teachers call `LevelOne:giveReview` for a student, the count of reviews is not being incremented.

Impact: This is extremely important as the invariant: "Students must have gotten all reviews before system upgrade. System upgrade should not occur if any student has not gotten 4 reviews (one for each week)" directly relies on the count being inaccurate. If the count is wrong, the invariant would not hold.

Proof of Concept: Since `reviewCount` is a private mapping, I changed it to a public mapping temporarily so that I can access the count value. By adding the following test code, we have shown that `reviewCount` is not being incremented.

```
function test_student_reviewCount_update() public schoolInSession {
    assertEq(levelOneProxy.reviewCount(harriet), 0);

    vm.warp(block.timestamp + 1 weeks);
    vm.prank(alice);
    levelOneProxy.giveReview(harriet, false);
    assertEq(levelOneProxy.reviewCount(harriet), 0);

    vm.warp(block.timestamp + 1 weeks);
    vm.prank(alice);
    levelOneProxy.giveReview(harriet, false);
    assertEq(levelOneProxy.reviewCount(harriet), 0);
}
```

Recommended Mitigation:

```
function giveReview(address _student, bool review) public onlyTeacher {
    if (!isStudent[_student]) {
        revert HH__StudentDoesNotExist();
    }
    require(reviewCount[_student] < 5, "Student review count exceeded!!!");
    require(block.timestamp >= lastReviewTime[_student] + reviewTime, "Reviews can only

++      reviewCount[_student]++;
}
```

[H-7] Mismatch In Storage Layout Of `LevelOne` And `LevelTwo`

Description: The storage layout of the `LevelTwo` contract differs from `LevelOne` as the variables present are not exactly the same. There are many missing variables in `LevelTwo` that are present in `LevelOne`.

Impact: Since the `LevelTwo` contract only defines functions to read variables,

this makes the contract unusable as the values returned would be wrong, since the proxy would follow the layout of the `LevelOne` contract when it was first initialised.

Proof of Concept: NA

Recommended Mitigation: Change the layout of `LevelTwo` to exactly follow that of `LevelOne`

```
    address principal;
    bool inSession;
++    uint256 schoolFees;
    uint256 public sessionEnd;
    uint256 public bursary;
    uint256 public cutOffScore;
    mapping(address => bool) public isTeacher;
    mapping(address => bool) public isStudent;
    mapping(address => uint256) public studentScore;
++    mapping(address => uint256) public reviewCount;
++    mapping(address => uint256) private lastReviewTime;
    address[] listOfStudents;
    address[] listOfTeachers;
```

Low Findings

[L-1] `LevelOne:initialize` Not Protected

Description: Attackers are able to call `LevelOne:initialize` directly on the implementation contract to initialize the logic contract with their own values or address that they control.

Impact: Attackers able to initialize `principal` to an address they control. This would give them free reign over the logic contract, however its impact is limited as the sensitive functions they would be able to exploit would not affect the memory region within the proxy contract.

Proof of Concept: NA

Recommended Mitigation: Create a constructor function such as below within the `LevelOne` contract.

```
    constructor() {
        _disableInitializers();
    }
```

Informational / Gas Findings

I-1: `public` functions not used internally could be marked `external`

Instead of marking a function as `public`, consider marking it as `external` if it is not used internally.

8 Found Instances

- Found in src/LevelOne.sol Line: 120

```
function initialize(address _principal, uint256 _schoolFees, address _usdcAddress)
```

- Found in src/LevelOne.sol Line: 201

```
function addTeacher(address _teacher) public onlyPrincipal notYetInSession {
```

- Found in src/LevelOne.sol Line: 220

```
function removeTeacher(address _teacher) public onlyPrincipal {
```

- Found in src/LevelOne.sol Line: 243

```
function expel(address _student) public onlyPrincipal {
```

- Found in src/LevelOne.sol Line: 269

```
function startSession(uint256 _cutOffScore) public onlyPrincipal notYetInSession {
```

- Found in src/LevelOne.sol Line: 277

```
function giveReview(address _student, bool review) public onlyTeacher {
```

- Found in src/LevelOne.sol Line: 295

```
function graduateAndUpgrade(address _levelTwo, bytes memory) public onlyPrincipal {
```

- Found in src/LevelTwo.sol Line: 28

```
function graduate() public reinitializer(2) {}
```

I-2: Empty `require()` / `revert()` statements

Use descriptive reason strings or custom errors for revert paths.

1 Found Instances

- Found in src/LevelOne.sol Line: 245

```
            revert();
```

I-3: Modifiers invoked only once can be shoe-horned into the function

1 Found Instances

- Found in src/LevelOne.sol Line: 101

```
modifier onlyTeacher() {
```

I-4: Empty Block

Consider removing empty blocks.

2 Found Instances

- Found in src/LevelOne.sol Line: 314

```
function _authorizeUpgrade(address newImplementation) internal override onlyPrincip
```

- Found in src/LevelTwo.sol Line: 28

```
function graduate() public reinitializer(2) {}
```

I-5: Unused Custom Error

it is recommended that the definition be removed when custom error is unused

1 Found Instances

- Found in src/LevelOne.sol Line: 86

```
error HH__HawkHighFeesNotPaid();
```

I-6: Boolean equality is not required.

If `x` is a boolean, there is no need to do `if(x == true)` or `if(x == false)`.
Just use `if(x)` and `if(!x)` respectively.

2 Found Instances

- Found in src/LevelOne.sol Line: 109

```
        if (inSession == true) {
```

- Found in src/LevelOne.sol Line: 244

```
        if (inSession == false) {
```