Hawk High Report

Version 1.0

franz-ops

May 22, 2025

Hawk High Report

franz-ops

May 16, 2025

Prepared by: franz-ops Lead Auditors: - franz-ops

Table of Contents

- [L-1] `LevelOne::giveReview` can be called even if the session is not active causing a logic break.
- [L-2] `Graduated` event is never emitted
- [L-3] `LevelOne::graduateAndUpgrade` can occurs even if the session is not ended

Protocol Summary

The Hawk High smart contract system simulates a school environment where: - Students enroll by paying fees (in USDC) and receive weekly reviews from Teachers. - A school session lasts 4 weeks, after which the Principal: - Ensures all students have received 4 reviews. - Pays wages: 5% to the Principal, 35% to Teachers, and 60% stays in the bursary. - Graduates students who meet a minimum score. - Upgrades the contract (UUPS pattern).

It enforces rules like one review per week and no upgrade unless all reviews are complete.

Disclaimer

This assessment was conducted on a best-effort basis within the allocated time frame. While every attempt has been made to identify potential vulnerabilities and assess the security of the Solidity codebase, no guarantees are provided regarding the completeness or accuracy of the findings. This assessment focuses solely on the technical security aspects of the smart contracts and does not constitute an endorsement of the overall product, project, or business model.

Risk Classification

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

Audit Details

There is no extra information about the audit details.

Scope

```
|-- src
|   |-- LevelOne.sol
|   \-- LevelTwo.sol
```

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

Issues found

Severity	Number of Issues Found
High	1
Medium	1
Low	3
Informational	0
Gas	0
Total	5

Findings

High

[H-1] System upgrade can occurs even if some students has no review during session causing Invariant Breaks

Description

The function `LevelOne::graduateAndUpgrade` does not check if all students got the necessary 4 reviews at the end of the session.

This means that a session can end even if no reviews have been submitted, breaking the invariant.

Impact

Session can end even if no reviews have been submitted, breaking the invariant.

Recommended mitigation

Consider adding a check in the function `LevelOne::graduateAndUpgrade` which revert if it finds a student with less then

PoC:

```
    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__NotAllowed();
+           }
+       }

        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);
```

```
24         }
```

Medium

[M-1] Incorrect storage slot assignments

Description

The `LevelTwo` contract has incorrect storage slot assignments for `sessionEnd`, `bursary`, `cutOffScore`, `isTeacher`, `isStudent`, `studentScore`, `listOfStudents`, `listOfTeachers`, and `usdc`. This can lead to unexpected behavior and security vulnerabilities.

The `LevelOne` contract has the following storage layout:

Name	Type	Slot	Offset	Bytes	Contract
principal	address	0	0	20	src/LevelOne.sol:LevelOne
inSession	bool	0	20	1	src/LevelOne.sol:LevelOne
schoolFees	uint256	1	0	32	src/LevelOne.sol:LevelOne
sessionEnd	uint256	2	0	32	src/LevelOne.sol:LevelOne
bursary	uint256	3	0	32	src/LevelOne.sol:LevelOne
cutOffScore	uint256	4	0	32	src/LevelOne.sol:LevelOne
isTeacher	mapping(address => bool)	5	0	32	src/LevelOne.sol:LevelOne
isStudent	mapping(address => bool)	6	0	32	src/LevelOne.sol:LevelOne
studentScore	mapping(address => uint256)	7	0	32	src/LevelOne.sol:LevelOne
reviewCount	mapping(address => uint256)	8	0	32	src/LevelOne.sol:LevelOne
lastReviewTime	mapping(address => uint256)	9	0	32	src/LevelOne.sol:LevelOne
listOfStudents	address[]	10	0	32	src/LevelOne.sol:LevelOne
listOfTeachers	address[]	11	0	32	src/LevelOne.sol:LevelOne
usdc	contract IERC20	12	0	20	src/LevelOne.sol:LevelOne

The `LevelTwo` contract has the following storage layout:

Name	Type	Slot	Offset	Bytes	Contract
principal	address	0	0	20	src/LevelTwo.sol:LevelTwo
inSession	bool	0	20	1	src/LevelTwo.sol:LevelTwo
sessionEnd	uint256	1	0	32	src/LevelTwo.sol:LevelTwo
bursary	uint256	2	0	32	src/LevelTwo.sol:LevelTwo
cutOffScore	uint256	3	0	32	src/LevelTwo.sol:LevelTwo
isTeacher	mapping(address => bool)	4	0	32	src/LevelTwo.sol:LevelTwo
isStudent	mapping(address => bool)	5	0	32	src/LevelTwo.sol:LevelTwo
studentScore	mapping(address => uint256)	6	0	32	src/LevelTwo.sol:LevelTwo
listOfStudents	address[]	7	0	32	src/LevelTwo.sol:LevelTwo
listOfTeachers	address[]	8	0	32	src/LevelTwo.sol:LevelTwo
usdc	contract IERC20	9	0	20	src/LevelTwo.sol:LevelTwo

Impact

It can cause storage collision issues, leading to unexpected behavior and security vulnerabilities.

Recommended mitigation

Consider changing the storage slot of the `LevelTwo` contract to match the one of the `LevelOne` contract.

Low

[L-1] `LevelOne::giveReview` can be called even if the session is not active causing a logic break.

Description

A principal can call the function `LevelOne::giveReview` even if the session is not active. This can lead to unexpected behavior and security vulnerabilities.

Proof of Concepts

In the following PoC we show how the function `LevelOne::giveReview` can be called by principal, even if the session is not active.

Proof of Code:

```
    function test_give_review_before_session_start() public {
        vm.warp(block.timestamp + 1 weeks);
        vm.roll(block.number + 1);
        vm.startPrank(principal);
        levelOneProxy.addTeacher(alice);
        levelOneProxy.addTeacher(bob);
        vm.stopPrank();

        vm.startPrank(clara);
        usdc.approve(address(levelOneProxy), schoolFees);

        levelOneProxy.enroll();
        vm.stopPrank();
        vm.startPrank(alice);
        levelOneProxy.giveReview(clara, false);

        vm.stopPrank();

        assert(levelOneProxy.studentScore(clara) == 90);
        assert(levelOneProxy.getSessionStatus() == false);
    }
```

Impact

Reviews can be given even if the session is not active, breaking the invariant.

Recommended mitigation

Consider adding a check in the function `LevelOne::giveReview` which revert if the session is not active.

PoC:

```
    function giveReview(address _student, bool _isPositive) public
       onlyTeacher {
+       if (!inSession) {
+           revert HH__NotAllowed();
+       }

        if (!isStudent[_student]) {
            revert HH__StudentDoesNotExist();
        }
        require(reviewCount[_student] < 5, "Student review count
           exceeded!!!");

        . . .
```

```
12
13      }
```

[L-2] `Graduated` event is never emitted

Description

The event `Graduated` is missing during the `graduateAndUpgrade` function causing a possible informational miss.

Impact

Informational miss, the event `Graduated` is never emitted.

Recommended mitigation

Consider to add the emit of `Graduated` event at the end of `graduateAndUpgrade` function.

PoC:

```
1  function graduateAndUpgrade(address _levelTwo, bytes memory) public
       onlyPrincipal {
2      if (_levelTwo == address(0)) {
3          revert HH__ZeroAddress();
4      }
5
6      uint256 totalTeachers = listOfTeachers.length;
7
8      uint256 payPerTeacher = (bursary * TEACHER_WAGE) / PRECISION;
9      uint256 principalPay = (bursary * PRINCIPAL_WAGE) / PRECISION;
10
11     _authorizeUpgrade(_levelTwo);
12
13     for (uint256 n = 0; n < totalTeachers; n++) {
14         usdc.safeTransfer(listOfTeachers[n], payPerTeacher);
15     }
16
17     usdc.safeTransfer(principal, principalPay);
18 +   emit Graduated(_levelTwo);
19 }
```

[L-3] `LevelOne::graduateAndUpgrade` can occurs even if the session is not ended

Description

The function `LevelOne::graduateAndUpgrade` does not check if the session is ended.

This means that the upgrade can occurs even if the session is still going, breaking the invariant.

Impact

Upgrade occurs when the session is not ended, breaking the invariant.

Recommended mitigation

Consider adding a check in the function `LevelOne::graduateAndUpgrade` which revert if the session is not ended yet and add the change state of `LevelOne::inSession` to false in order to consider the session ended.

Proof Of Code:

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

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