Hawk High Audit Report

Prepared by: Julián Cabrera Marceglia

Hawk High Audit Report

Prepared by: Julián Cabrera Marceglia

Lead Auditors:

- Julián Cabrera Marceglia (https://github.com/juliancabmar)

Assisting Auditors:

- None

Table of contents

About Julian Cabrera Marceglia

I am a security researcher who want to make the web3 enviroment safer.

Disclaimer

The Julián Cabrera Marceglia team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the 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

Audit Details

The findings described in this document correspond the following:

```
code base: https://github.com/juliancabmar/audit-first_flights-hawk_high

commit hash: b045cb951b0238f7b731bb2fa41abfe3466d5bd0
```

Scope

```
├── src
│   ├── LevelOne.sol
│   └── LevelTwo.sol
```

Protocol Summary

You have been contracted to review the upgradeable contracts for the Hawk High School which will be launched very soon. These contracts utilize the UUPSUpgradeable library from OpenZeppelin. At the end of the school session (4 weeks), the system is upgraded to a new one.

Roles

- `Principal`:
  - hiring/firing teachers
  - start new the school session
  - upgrading the system at the end of the school session
  - expel students who break rules
  - get 5% of the school fees
- `Teachers`:
  - giving reviews to students at the end of each week
  - get 35% of the school fees
- `Student`:
  - pay a school fee when enrolling in Hawk High School
  - 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	3
Medium	3
Low	1
Gas	8
Total	15

Findings

High

[H-1] Missing access control on `LevelOne::initialize` get exposed to MEV attack

Description:
The function who initialize the proxy implementation `LevelOne::initialize` haven't any access control, making possible a MEV attack because an attacker can call `LevelOne::initialize` first.

Impact:
The attacker got the full control of the protocol controling who is the Principal

Proof of Concept:

▶ Detalles

```
User Deploy proxy
     |
     v
User Deploy LevelOne
     |
     |--->**Attacker call LevelOne::initialize()** ---> (Attacker control
the protocol)
     |
     v
User call LevelOne::initialize ---> User Rejected
```

Recommended Mitigation:
Use Ownable library for UUPS of openzeppelin

[H-2] The teachers wage are changed on LevelTwo implementation modiffying the payment standard.

Description:
On LevelTwo implementation, are a not suppoused update on teachers wage from 35% to 40%

▶ Detalles

LevelOne.sol

```
    uint256 public constant TEACHER_WAGE = 35; // 35%
    uint256 public constant PRINCIPAL_WAGE = 5; // 5%
```

LevelTwo.sol

```
@>  uint256 public constant TEACHER_WAGE_L2 = 40; // 40%
    uint256 public constant PRINCIPAL_WAGE_L2 = 5; // 5%
```

Impact:
The payment standard will broken.

Recommended Mitigation:
Change the `TEACHER_WAGE_L2` to "35" on `LevelTwo.sol`, or redefine the payment standard.

[H-3] Teachers wage limit the number of teachers to add.

Description:
The teachers wage is 35% of the bursary, making that more of two teachers can't be added because it pass the 100% of the bursary (3 * 35% = 105%)

Impact:
Limit the protocol to have a maximun of two teachers.

Proof of Concept:
Add the follows to your test suite:

▶ PoC

```
function testCantAddMoreOfTwoTeachers() public {
    // create a new teacher
    address julian = makeAddr("julian");
    // adding three teachers
    vm.startPrank(principal);
    levelOneProxy.addTeacher(alice);
    levelOneProxy.addTeacher(bob);
    levelOneProxy.addTeacher(julian);
    vm.stopPrank();

    // adding a student
    vm.startPrank(clara);
    usdc.approve(address(levelOneProxy), schoolFees);
    levelOneProxy.enroll();
    vm.stopPrank();

    // principal starts session setting 90 for minimun score to pass
    vm.prank(principal);
    levelOneProxy.startSession(90);

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

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

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

Recommended Mitigation:
Standarize the number of teachers to two, or redefine the actual payment structure.

Medium

[M-1] `LevelOne::graduateAndUpgrade` not check if session ends.

Description:
The function `graduateAndUpgrade()` on `LevelOne` contract not make any check about if the session is ended, permit that upgrade the proxy before the 4 weeks long session standard.

Impact:
Students cannot obtain all of their teachers' evaluations, so those who would receive poor evaluations and not meet the cutoff score will still be able to graduate.

Recommended Mitigation:
Add the follows to `LevelOne::graduateAndUpgrade`:

▶ Detalles

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

+   require(block.timestamp >= sessionEnd, "Not session ended yet");

    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);
}
```

[M-2] `LevelOne::graduateAndUpgrade` not check on students reviews.

Description:
The function `graduateAndUpgrade()` on `LevelOne` contract not make any check about if the students have four reviews before make the upgrade

Impact:
Students will can be upgrade without reviews.

Proof of Concept:
Add the follows to your test suite:

▶ PoC

```
function testSystemCanBeUpgradedWithoutFourReviewsPerStudent() public {
    // adding a teacher
    vm.prank(principal);
    levelOneProxy.addTeacher(bob);
    // adding a student
    vm.startPrank(clara);
    usdc.approve(address(levelOneProxy), schoolFees);
    levelOneProxy.enroll();
    vm.stopPrank();
    // advance the time one week
```

```
        vm.warp(block.timestamp + 1 weeks);

        // review the student
        vm.prank(bob);
        levelOneProxy.giveReview(clara, true);

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

        bytes memory data = abi.encodeCall(LevelTwo.graduate, ());
        // Apply the upgrade
        vm.prank(principal);
        levelOneProxy.graduateAndUpgrade(levelTwoImplementationAddress, data);
    }
```

Recommended Mitigation:

▶ Fix

```

        ////////////////////////////////
        /////                      /////
        /////        ERRORS        /////
        /////                      /////
        ////////////////////////////////
        error HH__NotPrincipal();
        error HH__NotTeacher();
        error HH__ZeroAddress();
        error HH__TeacherExists();
        error HH__StudentExists();
        error HH__TeacherDoesNotExist();
        error HH__StudentDoesNotExist();
        error HH__AlreadyInSession();
        error HH__ZeroValue();
        error HH__HawkHighFeesNotPaid();
        error HH__NotAllowed();
    +   error HH__InsuficientReviews(address student, uint256 reviewsNum);
```

Add the follows to `LevelOne::giveReview`:

```

    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 be given once per week");

        // where `false` is a bad review and true is a good review
        if (!review) {
            studentScore[_student] -= 10;
        }

        // Update last review time
        lastReviewTime[_student] = block.timestamp;

+       reviewCount[_student]++;
        emit ReviewGiven(_student, review, studentScore[_student]);
    }
```

Add the follows to `LevelOne::graduateAndUpgrade`:

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

+       uint256 studentLength = listOfStudents.length;
+       for (uint256 n = 0; n < studentLength; n++) {
+           if (reviewCount[listOfStudents[n]] < 4) {
+               revert HH__InsuficientReviews(listOfStudents[n],
    reviewCount[listOfStudents[n]]);
+           }
+       }

        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);
    }
```

[M-3] `LevelOne::graduateAndUpgrade` not check on students cuttoff score.

Description:
The function `graduateAndUpgrade()` on `LevelOne` contract not make any check about if the students match the cuttoff score before make the upgrade

Impact:
Students will can be upgrade without have the sufficient score.

Proof of Concept:
Add the follows to your test suite:

▶ PoC

```
function testAStudentWhoNotMeetCutOffScoreCanBeUpgraded() public {
    uint256 cutOffScore = 90;
    // adding a teacher
    vm.prank(principal);
    levelOneProxy.addTeacher(bob);
    // adding a student
    vm.startPrank(clara);
    usdc.approve(address(levelOneProxy), schoolFees);
    levelOneProxy.enroll();
    vm.stopPrank();

    // principal starts session setting 90 for minimun score to pass
    vm.prank(principal);
    levelOneProxy.startSession(cutOffScore);

    // advance the time one week
    vm.warp(block.timestamp + 1 weeks);
    // review the student
    vm.prank(bob);
    levelOneProxy.giveReview(clara, false);
    // now his score is 90

    // advance the time one week
    vm.warp(block.timestamp + 1 weeks);
    // review the student
    vm.prank(bob);
    levelOneProxy.giveReview(clara, false);
    // now his score is 80

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

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

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

    assertEq(clara, LevelTwo(proxyAddress).getListOfStudents()[0]);
    console2.log("Cut off score: ", cutOffScore);
    console2.log("Clara final score: ",
LevelTwo(proxyAddress).studentScore(clara));
}
```

Recommended Mitigation:

▶ Fix

```
    /////////////////////////////////
    /////                      /////
    /////        ERRORS        /////
    /////                      /////
    /////////////////////////////////
    error HH__NotPrincipal();
    error HH__NotTeacher();
    error HH__ZeroAddress();
    error HH__TeacherExists();
    error HH__StudentExists();
    error HH__TeacherDoesNotExist();
    error HH__StudentDoesNotExist();
    error HH__AlreadyInSession();
    error HH__ZeroValue();
    error HH__HawkHighFeesNotPaid();
    error HH__NotAllowed();
+   error HH__InsuficientScore(address student, uint256 score);
```

Add the follows to `LevelOne::graduateAndUpgrade`:

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

+   uint256 studentLength = listOfStudents.length;
+   for (uint256 n = 0; n < studentLength; n++) {
+       if (studentScore[listOfStudents[n]] < cutOffScore) {
+           revert HH__InsuficientScore(listOfStudents[n],
studentScore[listOfStudents[n]]);
+       }
+   }

    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);
}
```

[L-1] The cutoff score can be upper than 100

Description:
On `LevelOne::startSession` function the accept a `_cutOffScore` parameter above of 100

Impact:
No student can be upgrade because the cutoff score match will've impossible.

Recommended Mitigation:
Add the following to `LevelOne::startSession`:

```
function startSession(uint256 _cutOffScore) public onlyPrincipal
notYetInSession {
+   require(_cutOffScore <= 100, "Cuttoff Score can't be above of 100");
    sessionEnd = block.timestamp + 4 weeks;
    inSession = true;
    cutOffScore = _cutOffScore;

    emit SchoolInSession(block.timestamp, sessionEnd);
}
```

Gas

[G-1] `LevelOne::removeTeacher` Ineficient loop.

Description:
The `LevelOne::removeTeacher` have a for loop where storage is read many times

▶ PoC

```
function removeTeacher(address _teacher) public onlyPrincipal {
    if (_teacher == address(0)) {
        revert HH__ZeroAddress();
    }

    if (!isTeacher[_teacher]) {
        revert HH__TeacherDoesNotExist();
    }
    uint256 teacherLength = listOfTeachers.length;
@>  for (uint256 n = 0; n < teacherLength; n++) {
        if (listOfTeachers[n] == _teacher) {
            listOfTeachers[n] = listOfTeachers[teacherLength - 1];
            listOfTeachers.pop();
            break;
        }
    }

    isTeacher[_teacher] = false;

    emit TeacherRemoved(_teacher);
}
```

Recommended Mitigation:
Use local memory array [] instead

```
function removeTeacher(address _teacher) public onlyPrincipal {
    if (_teacher == address(0)) {
        revert HH__ZeroAddress();
    }

    if (!isTeacher[_teacher]) {
        revert HH__TeacherDoesNotExist();
    }
+   address[] memory mlistOfTeachers = listOfTeachers;
-   uint256 teacherLength = listOfTeachers.length;
+   uint256 teacherLength = mlistOfTeachers.length;
    for (uint256 n = 0; n < teacherLength; n++) {
-       if (listOfTeachers[n] == _teacher) {
-           listOfTeachers[n] = listOfTeachers[teacherLength - 1];
-           listOfTeachers.pop();
+       if (mlistOfTeachers[n] == _teacher) {
+           mlistOfTeachers[n] = mlistOfTeachers[teacherLength - 1];
+           mlistOfTeachers.pop();
            break;
        }
    }

+   listOfTeachers = mlistOfTeachers;

    isTeacher[_teacher] = false;

    emit TeacherRemoved(_teacher);
}
```

[G-2] `LevelOne::expel` Ineficient loop.

Description:
The `LevelOne::expel` have a for loop where storage is read many times

▶ PoC

```
function expel(address _student) public onlyPrincipal {
    if (inSession == false) {
        revert();
    }
    if (_student == address(0)) {
        revert HH__ZeroAddress();
    }

    if (!isStudent[_student]) {
        revert HH__StudentDoesNotExist();
    }
    uint256 studentLength = listOfStudents.length;
@>  for (uint256 n = 0; n < studentLength; n++) {
        if (listOfStudents[n] == _student) {
            listOfStudents[n] = listOfStudents[studentLength - 1];
            listOfStudents.pop();
            break;
        }
    }

    isStudent[_student] = false;

    emit Expelled(_student);
}
```

Recommended Mitigation:

Use local memory array [] instead

```diff

function expel(address _student) public onlyPrincipal {
    if (inSession == false) {
        revert();
    }
    if (_student == address(0)) {
        revert HH__ZeroAddress();
    }

    if (!isStudent[_student]) {
        revert HH__StudentDoesNotExist();
    }
+   address[] memory mlistOfStudents = listOfStudents;
-   uint256 studentLength = listOfStudents.length;
+   uint256 studentLength = mlistOfStudents.length;
    for (uint256 n = 0; n < studentLength; n++) {
-       if (listOfStudents[n] == _student) {
-           listOfStudents[n] = listOfStudents[studentLength - 1];
-           listOfStudents.pop();
-           break;
-       }
+       if (mlistOfStudents[n] == _student) {
+           mlistOfStudents[n] = mlistOfStudents[studentLength - 1];
+           mlistOfStudents.pop();
+           break;
+       }
    }

+   listOfStudents = mlistOfStudents;

    isStudent[_student] = false;

    emit Expelled(_student);
}
```

[G-3] Other Gas improvements.

Description and Mitigation:

`LevelOne::principal`: initialized once, make it immutable.

`LevelOne::inSession`: initialized once, make it immutable.

`LevelOne::schoolFees`: initialized once, make it immutable.

`LevelOne::usdc`: initialized once, make it immutable.

`LevelOne::reviewCount`: is never initialized, remove it.

`LevelOne::reviewTime`: initialized in declaration and never changed, make it constant.