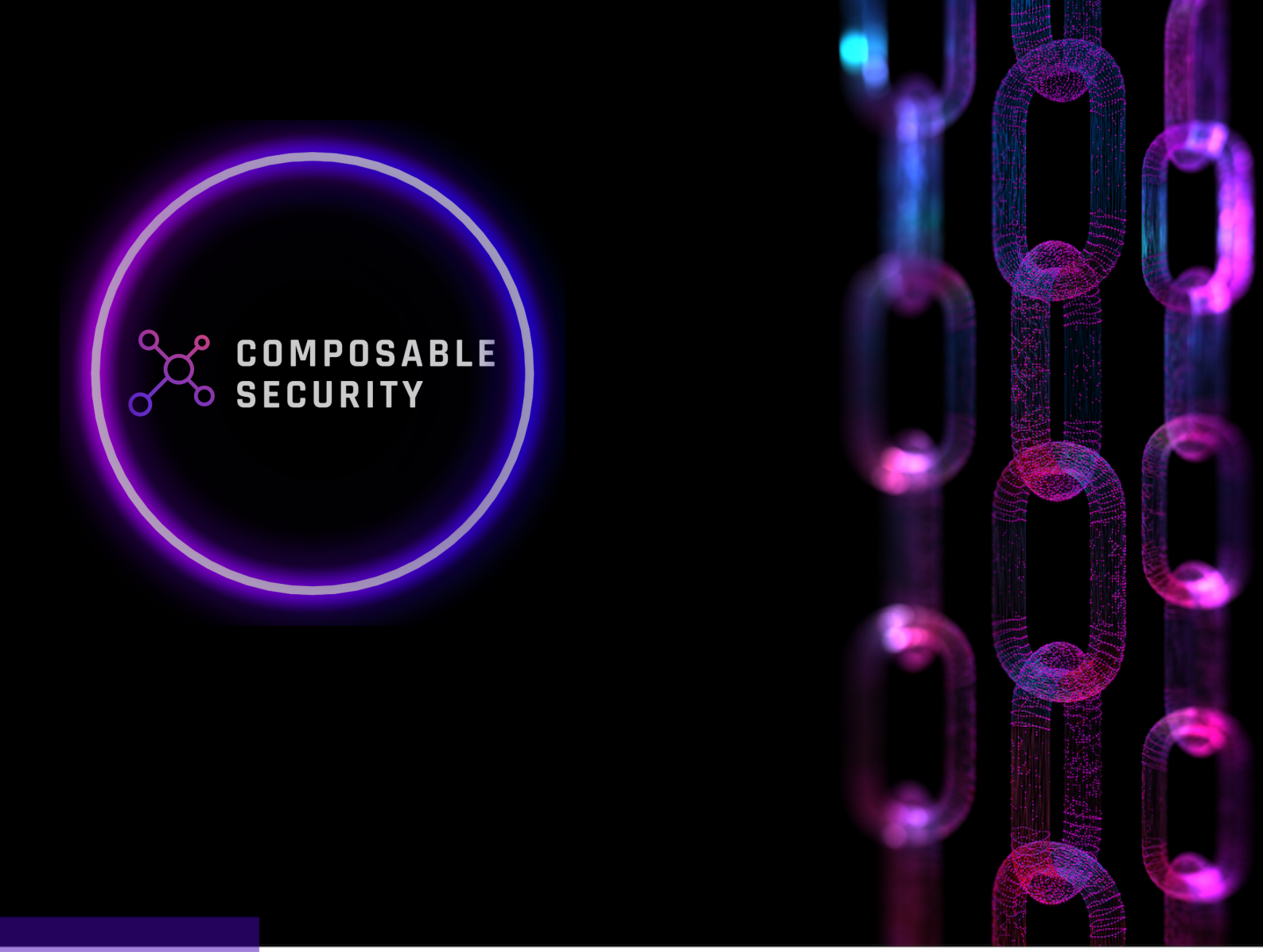


COMPOSABLE SECURITY

# REPORT

Smart contract security review for Arcade Services Inc.

Prepared by: Composable Security
Report ID: ARC-8aa78524
Test time period: 2024-01-24 - 2024-02-12
Retest time period: 2024-03-17 - 2024-03-27
Report date: 2024-03-27
Version: 1.1
Visit: composable-security.com

# Contents

**1. Retest summary (2024-03-27)** **3**
1.1 Results . . . 3
1.2 Scope . . . 4

**2. Current findings status** **5**

**3. Security review summary (2024-02-12)** **8**
3.1 Results . . . 8
3.2 Scope . . . 9

**4. Project details** **10**
4.1 Projects goal . . . 10
4.2 Agreed scope of tests . . . 10
4.3 Threat analysis . . . 11
4.4 Testing methodology . . . 12
4.5 Disclaimer . . . 12

**5. Vulnerabilities** **13**
[ARC-8aa78524-C01] Unvalidated _token parameter . . . 13
[ARC-8aa78524-H01] Lack of fees validation . . . 14
[ARC-8aa78524-H02] Replay attack on signature check . . . 15
[ARC-8aa78524-H03] Lack of possibility to collect vesting funds . . . 16
[ARC-8aa78524-M01] The beneficiary cannot withdraw claimable funds . . . 18
[ARC-8aa78524-M02] The currentMissionId variable is never updated . . . 20
[ARC-8aa78524-M03] Problematic minting functionality design . . . 21
[ARC-8aa78524-M04] Inefficient architecture of setAirdropClaims function . . . 23
[ARC-8aa78524-M05] Early withdrawal requires approval . . . 25
[ARC-8aa78524-M06] DoS via unbound loops and external calls . . . 26
[ARC-8aa78524-M07] Incorrect default mission status . . . 27
[ARC-8aa78524-M08] Incorrect handling of non-standard ERC20 tokens . . . 28
[ARC-8aa78524-M09] Locking reward via multiple calls to closeMission . . . 29
[ARC-8aa78524-M10] Swap fee non-compliant with the documentation . . . 30
[ARC-8aa78524-M11] Excessively powerful owner . . . 31
[ARC-8aa78524-L01] Not following the Checks-Effects-Interactions pattern . . . 33
[ARC-8aa78524-L02] Incorrect assignment of beneficiaries to the vault . . . 34
[ARC-8aa78524-L03] The FundingEnded event is not emitted . . . 35
[ARC-8aa78524-L04] Full control over rewards by MPO . . . 36
[ARC-8aa78524-L05] Costly and unnecessarily complicated vault architecture . . . 37
[ARC-8aa78524-L06] Lack of 2 step ownership transfer . . . 38

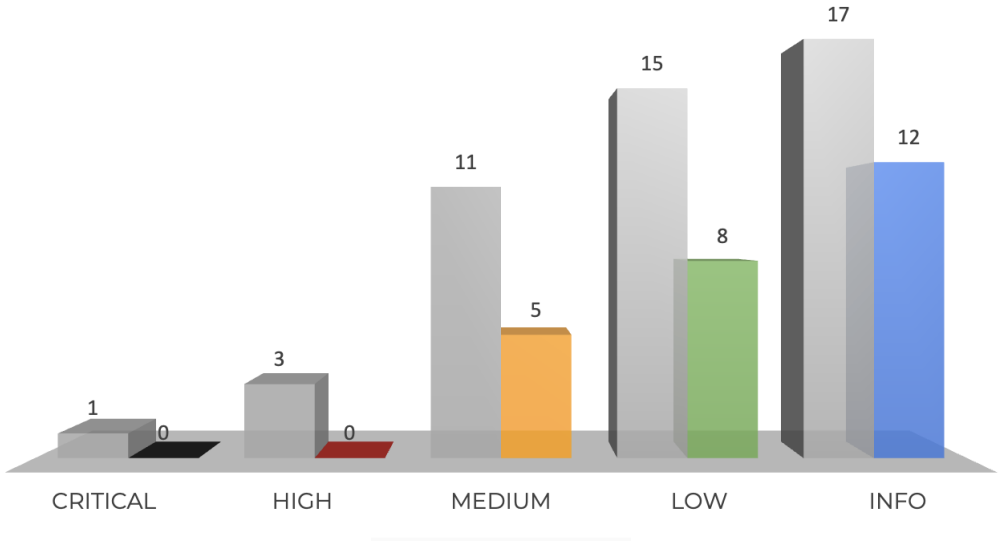
[ARC-8aa78524-L07] Multiple entries with the same investor . . . . . 39
[ARC-8aa78524-L08] No event emitted in important state-changing functions . . . 40
[ARC-8aa78524-L09] The vestingSchedules function lacks appropriate validation . 41
[ARC-8aa78524-L10] Invalid use of transferFrom . . . . . 42
[ARC-8aa78524-L11] Lock of Ether via payable functions . . . . . 43
[ARC-8aa78524-L12] Temporal block of OTF channel . . . . . 43
[ARC-8aa78524-L13] Invalid check for max token supply when swapping . . . . . 45
[ARC-8aa78524-L14] Lack of safeTransfer . . . . . 45
[ARC-8aa78524-L15] Use of dependencies with known vulnerabilities . . . . . 46

**6. Recommendations** **48**
[ARC-8aa78524-R01] Consider using the latest solidity version . . . . . 48
[ARC-8aa78524-R02] Unnecessary expensive claim function . . . . . 48
[ARC-8aa78524-R03] Consider new logic for setVaultAddress function . . . . . 49
[ARC-8aa78524-R04] Follow the layout order from Solidity Style Guide . . . . . 50
[ARC-8aa78524-R05] Remove reduntant checks . . . . . 51
[ARC-8aa78524-R06] Reorder storage variables . . . . . 51
[ARC-8aa78524-R07] Do not duplicate events . . . . . 53
[ARC-8aa78524-R08] Remove reduntant code . . . . . 53
[ARC-8aa78524-R09] Remove unused code . . . . . 54
[ARC-8aa78524-R10] Save values in temporal memory variables . . . . . 55
[ARC-8aa78524-R11] Use ENUMs instead of hardcoded numeric values . . . . . 55
[ARC-8aa78524-R12] Use custom errors . . . . . 56
[ARC-8aa78524-R13] Use hash of role names . . . . . 57
[ARC-8aa78524-R14] Use reward to determine the result of a mission . . . . . 57
[ARC-8aa78524-R15] Keep validation checks consistent . . . . . 58
[ARC-8aa78524-R16] Use packages for external contracts . . . . . 58
[ARC-8aa78524-R17] Insecure upgradeability . . . . . 59

**7. Impact on risk classification** **61**

**8. Long-term best practices** **62**
8.1 Use automated tools to scan your code regularly . . . . . 62
8.2 Perform threat modeling . . . . . 62
8.3 Use Smart Contract Security Verification Standard . . . . . 62
8.4 Discuss audit reports and learn from them . . . . . 62
8.5 Monitor your and similar contracts . . . . . 62

# 1. Retest summary (2024-03-27)




The description of the current status for each retested vulnerability and recommendation has been added in its section.

## 1.1. Results

The **Composable Security** team has performed two iterations of verification whether the vulnerabilities detected during the tests (between 2024-03-17 and 2024-03-27) were removed correctly and no longer appear in the code.

The current status of detected issues is as follows:

- 1 **critical** vulnerabilities have been fully fixed.
- All 3 **high** vulnerabilities have been fully fixed.
- 11 vulnerabilities with a **medium** impact on risk were handled as follows:
  - 6 have been fixed,
  - 5 have been acknowledged.
- 15 vulnerabilities with a **low** impact on risk were handled as follows:
  - 7 have been fixed,
  - 8 have been acknowledged.
- 17 security **recommendations** were handled as follows:
  - 5 have been implemented,
  - 1 has been partially implemented,
  - 11 have not been implemented.

The team decided to fix selected low risk vulnerabilities and implemented most of the recommendations in future iterations.

## 1.2. Scope

The retest scope included the same contracts, on a different commit in the same repository.

**GitHub repository:** https://github.com/ArcadeDevelopment/arcade-evm-mp-contracts
**CommitID:** bccfd3d338d954462a40ab674af778fea99199c1

**GitHub repository:** https://github.com/ArcadeDevelopment/arcade-evm-token-contracts
**CommitID:** 0a302a799a970690661f7c60e28a79f6f89e66c8

**GitHub repository:** https://github.com/ArcadeDevelopment/arcade-evm-vesting-contracts
**CommitID:** 16689c167840abbf630a0cb58d1d4ec1d7c0f74d

# 2. Current findings status

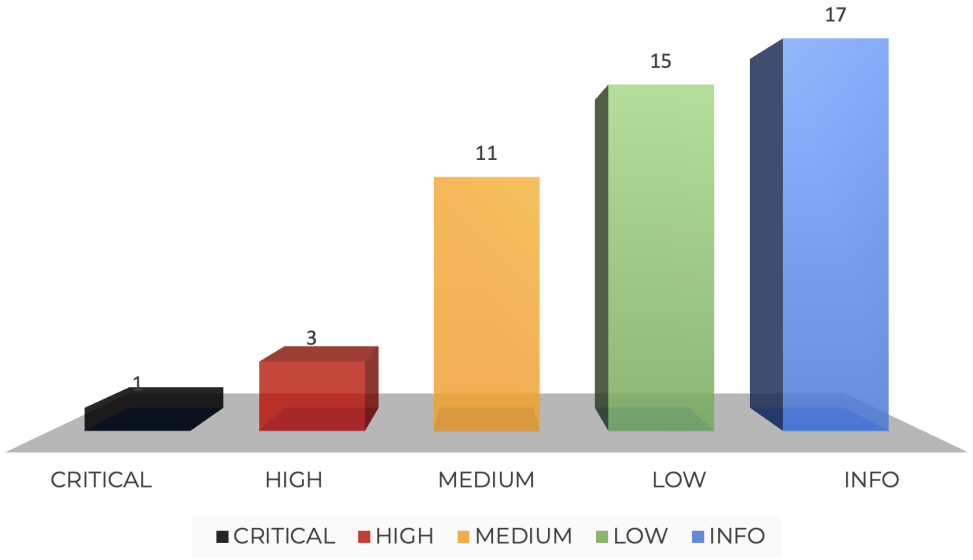
| ID | Severity | Vulnerability | Status |
|---|---|---|---|
| ARC-8aa78524-C01 | CRITICAL | Unvalidated _token parameter | FIXED |
| ARC-8aa78524-H01 | HIGH | Lack of fees validation | FIXED |
| ARC-8aa78524-H02 | HIGH | Replay attack on signature check | FIXED |
| ARC-8aa78524-H03 | HIGH | Lack of possibility to collect vesting funds | FIXED |
| ARC-8aa78524-M01 | MEDIUM | The beneficiary cannot withdraw claimable funds | FIXED |
| ARC-8aa78524-M02 | MEDIUM | The currentMissionId variable is never updated | FIXED |
| ARC-8aa78524-M03 | MEDIUM | Problematic minting functionality design | ACKNOWLEDGED |
| ARC-8aa78524-M04 | MEDIUM | Inefficient architecture of setAirdropClaims function | ACKNOWLEDGED |
| ARC-8aa78524-M05 | MEDIUM | Early withdrawal requires approval | FIXED |
| ARC-8aa78524-M06 | MEDIUM | DoS via unbound loops and external calls | FIXED |
| ARC-8aa78524-M07 | MEDIUM | Incorrect default mission status | ACKNOWLEDGED |
| ARC-8aa78524-M08 | MEDIUM | Incorrect handling of non-standard ERC20 tokens | ACKNOWLEDGED |
| ARC-8aa78524-M09 | MEDIUM | Locking reward via multiple calls to closeMission | FIXED |
| ARC-8aa78524-M10 | MEDIUM | Swap fee non-compliant with the documentation | FIXED |
| ARC-8aa78524-M11 | MEDIUM | Excessively powerful owner | ACKNOWLEDGED |
| ARC-8aa78524-L01 | LOW | Not following the Checks-Effects-Interactions pattern | FIXED |
| ARC-8aa78524-L02 | LOW | Incorrect assignment of beneficiaries to the vault | ACKNOWLEDGED |
| ARC-8aa78524-L03 | LOW | The FundingEnded event is not emitted | FIXED |
| ARC-8aa78524-L04 | LOW | Full control over rewards by MPO | ACKNOWLEDGED |

| | | | |
|---|---|---|---|
| ARC-8aa78524-L05 | LOW | Costly and unnecessarily complicated vault architecture | ACKNOWLEDGED |
| ARC-8aa78524-L06 | LOW | Lack of 2 step ownership transfer | ACKNOWLEDGED |
| ARC-8aa78524-L07 | LOW | Multiple entries with the same investor | ACKNOWLEDGED |
| ARC-8aa78524-L08 | LOW | No event emitted in important state-changing functions | FIXED |
| ARC-8aa78524-L09 | LOW | The vestingSchedules function lacks appropriate validation | ACKNOWLEDGED |
| ARC-8aa78524-L10 | LOW | Invalid use of transferFrom | FIXED |
| ARC-8aa78524-L11 | LOW | Lock of Ether via payable functions | FIXED |
| ARC-8aa78524-L12 | LOW | Temporal block of OTF channel | ACKNOWLEDGED |
| ARC-8aa78524-L13 | LOW | Invalid check for max token supply when swapping | FIXED |
| ARC-8aa78524-L14 | LOW | Lack of safeTransfer | FIXED |
| ARC-8aa78524-L15 | LOW | Use of dependencies with known vulnerabilities | ACKNOWLEDGED |

| ID | Severity | Recommendation | Status |
|---|---|---|---|
| ARC-8aa78524-R01 | INFO | Consider using the latest solidity version | NOT IMPLEMENTED |
| ARC-8aa78524-R02 | INFO | Unnecessary expensive claim function | NOT IMPLEMENTED |
| ARC-8aa78524-R03 | INFO | Consider new logic for setVaultAddress function | NOT IMPLEMENTED |
| ARC-8aa78524-R04 | INFO | Follow the layout order from Solidity Style Guide | NOT IMPLEMENTED |
| ARC-8aa78524-R05 | INFO | Remove reduntant checks | NOT IMPLEMENTED |
| ARC-8aa78524-R06 | INFO | Reorder storage variables | NOT IMPLEMENTED |
| ARC-8aa78524-R07 | INFO | Do not duplicate events | IMPLEMENTED |
| ARC-8aa78524-R08 | INFO | Remove reduntant code | NOT IMPLEMENTED |
| ARC-8aa78524-R09 | INFO | Remove unused code | IMPLEMENTED |
| ARC-8aa78524-R10 | INFO | Save values in temporal memory variables | NOT IMPLEMENTED |
| ARC-8aa78524-R11 | INFO | Use ENUMs instead of hardcoded numeric values | NOT IMPLEMENTED |

| | | | |
|---|---|---|---|
| ARC-8aa78524-R12 | INFO | Use custom errors | NOT IMPLEMENTED |
| ARC-8aa78524-R13 | INFO | Use hash of role names | IMPLEMENTED |
| ARC-8aa78524-R14 | INFO | Use reward to determine the result of a mission | NOT IMPLEMENTED |
| ARC-8aa78524-R15 | INFO | Keep validation checks consistent | IMPLEMENTED |
| ARC-8aa78524-R16 | INFO | Use packages for external contracts | NOT IMPLEMENTED |
| ARC-8aa78524-R17 | INFO | Insecure upgradeability | IMPLEMENTED |

# 3. Security review summary (2024-02-12)




## 3.1. Results

The **Arcade Services Inc.** engaged Composable Security to review security of **Arcade Mission Pools, Vesting and Token**. Composable Security conducted this assessment over 2 person-weeks with 2 engineers.

The summary of findings is as follows:

- **1** vulnerability with a **critical** impact on risk was identified. Its potential consequence is:
  - Stealing funds of other MPCs.
- **3** vulnerabilities with a **high** impact on risk were identified. Their potential consequences are:
  - Theft of contributors investments.
  - Signature reuse to call any other protected function on the same mission pool contract.
  - Impossibility to withdraw tokens from further schedules.
- **11** vulnerabilities with a **medium** impact on risk were identified.
- **15** vulnerabilities with a **low** impact on risk were identified.
- **17 recommendations** have been proposed that can improve overall security and help implement best practices.
- The most important issues detected concern architecture and business logic.
- The project uses outdated versions of dependencies. The review of non-contract dependencies was outside the scope of testing, but upgrades to the latest versions should be considered as they may contain vulnerabilities. See details here: ARC-8aa78524-L15.

- The system architecture is not optimal. It contains many redundant files and does not align with best smart contract development practices. Contracts with identical uses are separate files instead of being reused. This unnecessarily increases the code base and will cause problems in maintaining it. See details here: ARC-8aa78524-L05.
- The system is highly centralized. Although this is consistent with the team's assumption, it should be taken into account that management accounts are often the target of attackers. Their privileged operations should be limited and occur at such a pace that the user has a chance to observe and act on them. By doing so, users and funds will be better protected. See details here: ARC-8aa78524-M11, ARC-8aa78524-L04.
- Many important vulnerabilities result from incorrect and not extensively tested implementation. Not necessarily from the possibility of carrying out an attack.

Composable Security recommends that **Arcade Services Inc.** complete the following:

- Address all reported issues.
- Consider architecture changes in line with the report's advice that will make the whole system more transparent, efficient and easier to manage.
- Extend unit tests with scenarios that cover detected vulnerabilities where possible.
- Consider whether the detected vulnerabilities may exist in other places (or ongoing projects) that have not been detected during engagement.
- Review dependencies and upgrade to the latest versions.
- Many vulnerabilities have been detected and some of them will require major changes in logic and architecture. It is recommended to consider performing another security review.

## 3.2. Scope

The scope of the tests included selected contracts from the following repository.

**GitHub repository:** https://github.com/ArcadeDevelopment/arcade-evm-mp-contracts
**CommitID:** 2b05208d2acc89793ba77f9be9beeb76cc766e95

**GitHub repository:** https://github.com/ArcadeDevelopment/arcade-evm-token-contracts
**CommitID:** 39578d8dbd28c92ca88ac6a903d926d43a95a9e5

**GitHub repository:** https://github.com/ArcadeDevelopment/arcade-evm-vesting-contracts
**CommitID:** 68e1b410d267262254585f3bb550f6e672b03ee1

The detailed scope of tests can be found in Agreed scope of tests.

# 4. Project details

## 4.1. Projects goal

The Composable Security team focused during this audit on the following:

- Perform a tailored threat analysis.
- Ensure that smart contract code is written according to security best practices.
- Identify security issues and potential threats both for **Arcade Services Inc.** and their users.
- The secondary goal is to improve code clarity and optimize code where possible.

## 4.2. Agreed scope of tests

The subjects of the test were selected contracts from the **Arcade Services Inc.** repository.

**GitHub repository:** https://github.com/ArcadeDevelopment/arcade-evm-mp-contracts
**CommitID:** 2b05208d2acc89793ba77f9be9beeb76cc766e95

Files in scope:

```
contracts/.
└── contracts/./src
    ├── contracts/./src/ArcadeMissionPools.sol
    └── contracts/./src/lib
        ├── contracts/./src/lib/DistributionsUtility.sol
        ├── contracts/./src/lib/MissionEvents.sol
        ├── contracts/./src/lib/MissionStructs.sol
        └── contracts/./src/lib/Signatory.sol
```

**GitHub repository:** https://github.com/ArcadeDevelopment/arcade-evm-token-contracts
**CommitID:** 39578d8dbd28c92ca88ac6a903d926d43a95a9e5

Files in scope:

```
contracts/.
└── contracts/./src
    ├── contracts/./src/ArcadeSwap.sol
    ├── contracts/./src/interfaces
    │   ├── contracts/./src/interfaces/IArcade.sol
    │   └── contracts/./src/interfaces/IXArcade.sol
    └── contracts/./src/tokens
        ├── contracts/./src/tokens/ArcadeToken.sol
```

```
        └── contracts/./src/tokens/XArcadeToken.sol
```

**GitHub repository:** https://github.com/ArcadeDevelopment/arcade-evm-vesting-contracts
**CommitID:** 68e1b410d267262254585f3bb550f6e672b03ee1

Files in scope:

```
contracts/.
├── contracts/./airdrop
│   ├── contracts/./airdrop/AirdropTokenClaim.sol
│   └── contracts/./airdrop/SolanaAirdropTokenClaim.sol
├── contracts/./interfaces
│   └── contracts/./interfaces/ITokenVault.sol
├── contracts/./vaults
│   ├── contracts/./vaults/AdvisoryVault.sol
│   ├── contracts/./vaults/ArcadeTeamVault.sol
│   ├── contracts/./vaults/CommunityVault.sol
│   ├── contracts/./vaults/GeneralReserveVault.sol
│   ├── contracts/./vaults/SolanaClaimVault.sol
│   ├── contracts/./vaults/TokenDistributionVault.sol
│   └── contracts/./vaults/base
│       └── contracts/./vaults/base/BaseTokenVault.sol
└── contracts/./vesting
    ├── contracts/./vesting/VestingTokenClaim.sol
    └── contracts/./vesting/VestingTokenClaimCore.sol
```

**Documentation:** The following sources were used to get acquainted with the idea of the project: ArcadeWhitepaper, Arcade-Litepaper-v.01.15.

## 4.3. Threat analysis

This section summarizes the potential threats that were identified during initial threat modeling performed before the audit. The tests were focused, but not limited to, finding security issues that could be exploited to achieve these threats.

Potential attackers goals:

- Theft of Arcade or XArcade token.
- Unlimited mint of Arcade or XArcade token.
- Lock users' funds in the Vault or Mission Pool.
- Theft of rewards from Mission Pool.
- Bypass vesting limits.

- Block the contract, so that others cannot use it.

Potential scenarios to achieve the indicated attacker's goals:

- Vesting a larger number of tokens through re-entrancy.
- Skipping the fee when swapping between Arcade and XArcade.
- Participating in multiple mission pools with the same tokens.
- Influence or bypass the business logic of the system.
- Take advantage of arithmetic errors.
- Privilege escalation through incorrect access control to functions or badly written modifiers.
- Existence of known vulnerabilities (e.g., front-running, re-entrancy).
- Design issues.
- Excessive power, too much in relation to the declared one.
- Unintentional loss of the ability to govern the system.
- Poor security against taking over the managing account.
- Private key compromise, rug-pull.
- Withdrawal of more funds than expected.

## 4.4. Testing methodology

Smart contract security review was performed using the following methods:

- Q&A sessions with the **Arcade Services Inc.** development team to thoroughly understand intentions and assumptions of the project.
- Initial threat modeling to identify key areas and focus on covering the most relevant scenarios based on real threats.
- Automatic tests using slither.
- Custom scripts (e.g. unit tests) to verify scenarios from initial threat modeling.
- **Manual review of the code.**

## 4.5. Disclaimer

Smart contract security review **IS NOT A SECURITY WARRANTY**.

During the tests, the Composable Security team makes every effort to detect any occurring problems and help to address them. However, it is not allowed to treat the report as a security certificate and assume that the project does not contain any vulnerabilities. Securing smart contract platforms is a multi-stage process, starting from threat modeling, through development based on best practices, security reviews and formal verification, ending with constant monitoring and incident response.

***Therefore, we encourage the implementation of security mechanisms at all stages of development and maintenance.***

# 5. Vulnerabilities

## [ARC-8aa78524-C01] Unvalidated _token parameter

**CRITICAL** **FIXED**

**Retest (2024-03-18)**

The vulnerability has been removed as recommended.

### Affected files

- ArcadeMissionPools.sol#L331

### Description

The Mission Pool Contributor (MPC) can use any token to fund mission (e.g. a malicious one) and withdraw the legitimate mission's token using `earlyWithdrawalFromMission` function.

### Attack scenario

The attackers might take the following steps in turn:

① Deploy a malicious token X.
② Mint a large number of X tokens to the attacker's address (as many as the `poolAmount` of the mission).
③ Call `fundMission` function and pass token X as a parameter.
④ The contract pulls the X token from the attacker and increases their investment amount.
⑤ Call `earlyWithdrawalFromMission` with the whole investment amount.
⑥ The contract transfers out the token assigned to the mission.

**Result of the attack:** Stealing funds of other MPCs.

**Recommendation**

- Remove `_token` parameter.
- Take the token from the `Mission` struct.

### References

1. SCSVS G5: Access control

# [ARC-8aa78524-H01] Lack of fees validation

HIGH FIXED

**Retest (2024-03-26)**

The vulnerability has been removed as recommended.

- The sum of mission pools fees is verified.
- The `BASE_EARLY_FEE` is limited as documented.
- The swap fee rates in ArcadeSwap are correctly limited.
- The platform fee description has been added to the whitepaper.

## Affected files

- ArcadeMissionPools.sol#L216
- ArcadeMissionPools.sol#L288
- ArcadeMissionPools.sol#L290
- ArcadeMissionPools.sol#L959
- ArcadeSwap.sol#L251

## Description

The fees assigned to mission are never validated whether they sum up to 100%. That can lead to the investments theft. For example, if `lenderRewardPoints` is 100000, representing 1000%, it would send not only the whole reward to lenders, but also part of investments.

The code also contains inconsistencies with the documentation:

- The `BASE_EARLY_FEE` is not limited to 15% (1500 basis points) which limit has been specified in the white paper ( ArcadeMissionPools.sol#L216, ArcadeMissionPools.sol#L959). The early withdrawal fee set in `createMission` function is never compared with the `BASE_EARLY_FEE`, which means that `BASE_EARLY_FEE` is never used.
- The platform fee specified in the `createMission` function is not limited, and it is not mentioned in the white paper.
- The swap fee rate in ArcadeSwap contract should be max 5% (500 basis points) according to the white paper, but it is not enforced in the contracts.

## Attack scenario

The attackers might take the following steps in turn:

① The MPO creates a mission with legitimate fee values, but the `_lenderRewardPoints` is set to 100000 (1000%).

② The MPC fund the mission.
③ The MPO adds a lender.
④ The MPO closes mission with reward equal to X tokens.
  - The mission pool contract pulls X tokens from the MPO.
  - The contract calculates the amount sent to lender. It is part of reward X times the fee percent which is set to 1000% (ten times the X tokens).
  - The contract sends 10 times X tokens to the lender, essentially sending the funds of MPC.

**Result of the attack:** Theft of MPCs investments.

**Recommendation**

- Add a mission pool fee verification which forces mission pool fees to add up to 100% for a set or update of any fee.
- Validate fees when they are set or updated and enforce business limits communicated in the white paper.
- Remove the platform fee deducted when the mission is closed as it was not communicated in the white paper or update the white paper.

### References

1. SCSVS G7: Arithmetic

## [ARC-8aa78524-H02] Replay attack on signature check

HIGH FIXED

**Retest (2024-03-18)**

The vulnerability has been removed. The team decided not to use EIP712, but has built a message that includes all important params to be signed.

### Affected files

- ArcadeMissionPools.sol#L257
- ArcadeMissionPools.sol#L337
- ArcadeMissionPools.sol#L391
- ArcadeMissionPools.sol#L451
- ArcadeMissionPools.sol#L582
- ArcadeMissionPools.sol#L662
- ArcadeMissionPools.sol#L702

- ArcadeMissionPools.sol#L818
- ArcadeMissionPools.sol#L852

### Description

Selected functions (see the Affected files section) include the signature check of `_hashHex` param, contract address, and sender's address. It is assumed that this is the signature generated off-chain to approve the call. However, the `_hashHex` parameter is never compared to any function parameters and not invalidated.

Any user that gets the correct signature for any call, can reuse it for other calls (e.g. fund other mission using the same signature and `_hashHex` as in legitimate function call.

### Attack scenario

The attackers might take the following steps in turn:

① Request a legitimate call to `fundMission` to fund 100 tokens via the off-chain component.
② The off-chain components verifies the call and signs it for the attacker.
③ Make the legitimate call with the correct signature.
④ Make other calls to the `fundMission` function with the same signature which is accepted.

**Result of the attack:** Signature reuse to call any other protected function on the same mission pool contract.

**Recommendation**

- Generate the hash based on the parameters and a nonce.
- Add domain separator to the signed data - see EIP712.
- Set the signed and executed hash as used on-chain to protect from replaying the same function call again.

### References

1. EIP-712: Typed structured data hashing and signing
2. EIP712.sol
3. SCSVS G5: Access control

## [ARC-8aa78524-H03] Lack of possibility to collect vesting funds

HIGH FIXED

**Retest (2024-03-25)**

The vulnerability has been removed as recommended.

## Affected files

- VestingTokenClaim.sol#L78

## Description

If the beneficiary calls the `claim` function that has more than one element in the `vestingSchedules`, the function iterates through all of their schedules. It is enough that the funds from one vesting are claimed in full, and the remaining funds (from other schedules) will be blocked. That's due to the occurring revert, which checks whether the funds have already been fully claimed from any schedule.

```
70  function claim() external {
71      VestingSchedule[] storage beneficiaryVestingSchedules = vestingSchedules[msg.
            sender];
72
73      require(beneficiaryVestingSchedules.length > 0, "No vesting schedules found")
            ;
74
75      for(uint256 i = 0; i < beneficiaryVestingSchedules.length; i++) {
76          VestingSchedule storage vestingSchedule = beneficiaryVestingSchedules[i];
77
78          require(vestingSchedule.claimedAmount < vestingSchedule.totalAmount, "
                Nothing to claim");
79
80          uint256 claimableAmount = _getClaimableAmount(vestingSchedule);
81          require(claimableAmount > 0, "Nothing to claim");
82
83          vestingSchedule.claimedAmount += claimableAmount;
84          vault.release(msg.sender, claimableAmount);
85
86          emit Claimed(msg.sender, claimableAmount);
87      }
88  }
```

## Vulnerable scenario

The following scenario might occur:

① One address have 2 schedules assigned, as they were participating in both private and

public sale.

② The **first schedule** values: {`cliff 1000`, `duration 1000`, `totalAmount 50000`, `claimedAmount 0`, `startTime 0`, `BeneficiaryType private`}

③ The **second schedule** values: {`cliff 2000`, `duration 1000`, `totalAmount 50000`, `claimedAmount 0`, `startTime 0`, `BeneficiaryType public`}

④ The 2000 seconds pass, so the address can use the `claim` function to receive a `totalAmount` equal to 50000 from the first schedule (as `cliff` and `duration` passed) and 0 tokens from the second schedule, because only the `cliff` time has passed and not the `duration`.

⑤ The **first schedule** changed values: {`cliff 1000`,`duration 1000`,`totalAmount 50000`, `claimedAmount 50000`,`startTime 0`,`BeneficiaryType private`}

⑥ During the second `claim` after another 1000 seconds, the `for` loop checks the first schedule first and due to the check in line 78, the function reverts because `claimedAmount == totalAmount`.

**Result of the attack:** Impossibility to withdraw tokens from further schedules.

**Recommendation**

- Remove for loop. Deploy a `vestingTokenClaim` contract for each vault and set one vesting schedule for each beneficiary.
- Alternatively, remove the `require` statement from line 78 and let the `for` loop execute further schedules.

### References

1. SCSVS G4: Business logic

## [ARC-8aa78524-M01] The beneficiary cannot withdraw claimable funds

MEDIUM FIXED

**Retest (2024-03-18)**

The vulnerability has been removed as recommended. The `for` loop has been kept but the code is executed only if the claimable amount is greater than 0.

### Affected files

- VestingTokenClaim.sol#L81

## Description

If the beneficiary calls the `claim` function that has more than one element in the `vestingSchedules`, the function iterates through all of their schedules.

If the first schedule in the sequence does not have any claimable amount, a revert will occur. This way, even if the next schedule allows for a partial or even full claim, the user will not be able to withdraw their tokens within the scheduled time.

```
70  function claim() external {
71      VestingSchedule[] storage beneficiaryVestingSchedules = vestingSchedules[msg.
            sender];
72
73      require(beneficiaryVestingSchedules.length > 0, "No vesting schedules found")
            ;
74
75      for(uint256 i = 0; i < beneficiaryVestingSchedules.length; i++) {
76          VestingSchedule storage vestingSchedule = beneficiaryVestingSchedules[i];
77
78          require(vestingSchedule.claimedAmount < vestingSchedule.totalAmount, "
                Nothing to claim");
79
80          uint256 claimableAmount = _getClaimableAmount(vestingSchedule);
81          require(claimableAmount > 0, "Nothing to claim");
82
83          vestingSchedule.claimedAmount += claimableAmount;
84          vault.release(msg.sender, claimableAmount);
85
86          emit Claimed(msg.sender, claimableAmount);
87      }
88  }
```

## Vulnerable scenario

The following scenario might occur:

① One address have 2 schedules assigned, as they were participating in both private and public sale.
② The **first schedule** values: {`cliff` 5000, `duration` 1000, `totalAmount` 50000, `claimedAmount` 0, `startTime` 0, `BeneficiaryType` `private`}
③ The **second schedule** values: {`cliff` 1000, `duration` 1000, `totalAmount` 50000, `claimedAmount` 0, `startTime` 0, `BeneficiaryType` `public`}
④ The 2000 seconds pass, so the `totalAmount` from the second schedule should be claimable as `cliff` and `duration` passed.

⑤ However, the **first schedule** will still have `claimableAmount` = 0 and because of the check in line 81 it will not allow for claim the `totalAmount` from the second schedule as it should.

**Result of the attack:** The beneficiary will not be able to claim their funds according to the schedule.

**Recommendation**

- Remove for loop. Deploy a `vestingTokenClaim` contract for each vault and set one vesting schedule for each beneficiary.
- Alternatively, instead of `require`, use an `if` statement that updates the `claimedAmount` and makes a transfer, while in the case where there is nothing to claim, move to the next element in the loop.

### References

1. SCSVS G4: Business logic

## [ARC-8aa78524-M02] The currentMissionId variable is never updated

MEDIUM FIXED

**Retest (2024-03-18)**

The `currentMissionId` storage variable is correctly updated. However, for more code clarity, it can be removed and the `missionIdCounter.current()` can be used instead.

### Affected files

- ArcadeMissionPools.sol#L288
- ArcadeMissionPools.sol#L297

### Description

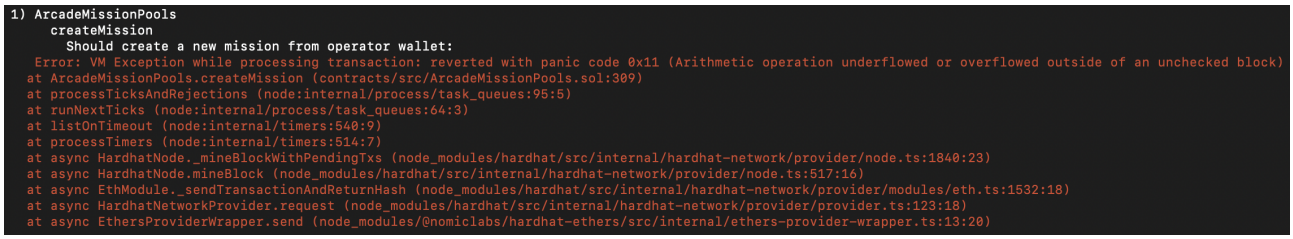
The `ArcadeMissionPools` contract uses `currentMissionId` variable to set the details of mission in mappings. However, it is never updated. The only updated variable is the `missionIdCounter`, but it is never assigned to `currentMissionId` after increase.

Additionally, the `createMission` decreases the value of `currentMissionId` variable to return the index of the mission pool.

```
308 missionIdCounter.increment();
```

```
309 return currentMissionId - 1;
```

That leads to underflow error and does not allow MPOs to deploy any mission pool as show on the picture below.




**Result of the attack:** The original source code does not allow to deploy any mission pool.

**Note:** This vulnerability, if not patched properly, may lead to a critical vulnerability. If the `currentMissionId` variable is only decreased (to avoid underflow error) it would lead to overwriting the details of mission pool with index 0.

**Recommendation**

Remove the `currentMissionId` variable and take the current mission id from `missionIdCounter` variable.

```
308 missionIdCounter.increment();
309 return missionIdCounter.current() - 1;
```

### References

1. SCSVS G4: Business Logic

## [ARC-8aa78524-M03] Problematic minting functionality design

MEDIUM ACKNOWLEDGED

**Retest (2024-03-18)**

No recommendation has been implemented.
Team response:
*The updateVaultsMinted will be called by the contract owner after tokens have been minted to the various vaults required. As there is a need to use Arcade token for the initial liquidity on exchanges/token launchers, we set it up to mint the tokens.*

*We have outlined the amounts to mint on each contract to assure the values do not exceed the max supply of 800,000,000 Arcade and xArcade tokens in circulation.*

## Affected files

- ArcadeToken.sol#L81
- ArcadeToken.sol#L131
- ArcadeToken.sol#L149
- XArcadeToken.sol#L97
- XArcadeToken.sol#L147
- XArcadeToken.sol#L165

## Description

The Arcade team plans to mint `XArcadeToken` to the vaults and `ArcadeToken` for initial liquidity in token sales. The current implementation of the minting functionality not only allows for abuse by the owner but also allows for mistakes to be made that may completely block the correct operation of the system. The main problems are:

1. Even though `MAX_SUPPLY` should be `800000000*10**18` tokens, they can be minted twice as many because both `ArcadeToken` and `XArcadeToken` are mintable (by the Owner and by `ArcadeSwap`, the contract selected by the Owner).
2. When swapping `ArcadeToken` for `XArcadeToken` (or the opposite) via `ArcadeSwap`, it is verified whether `arcadeSupply + xArcadeSupply` are less than `MAX_SUPPLY`. Due to the problem mentioned in 1, it is possible that the swap functionality will stop working and will revert every call.
3. The administrator's task is to mint the appropriate amounts into Vaults and then call the `updateVaultsMinted` function, which blocks the further possibility of minting into vaults. However, it does not have to ever call this function, leaving the possibility of minting to any address. Calling this function prematurely (e.g. several transactions go to the mempool and this one is selected first) may also result in the inability to mint tokens to vaults.
4. Since `xArcadeToken` is based on the LayerZero OFTV2 solution, this token can be transferred to another chain. The `totalSupply` will decrease by burning tokens as they are transferred, allowing for further mint as `circulatingSupply` is not taken into account.

The entire process can be significantly simplified and most of the current threats can be mitigated.

**Result of the attack:** It may not be possible to swap `ArcadeToken` for `XArcadeToken` via `ArcadeSwap`.

**Recommendation**

- Instead of implementing minting to vaults in both tokens, handle the entire process in one selected token (e.g. `xArcadeToken`).
- If the total value to be minted into the vaults is known in advance, an appropriate limit should be applied to it. If not, make `mintToVault` one-time use and not require a separate call to disable it.
- When minting and setting limits, take into account the cross-chain `burn` resulting from OFTV2 implementation and the change in circulating supply as it might allow for extra an mint as owner.

### References

1. SCSVS G1: Architecture, design and threat modeling

## [ARC-8aa78524-M04] Inefficient architecture of setAirdropClaims function

MEDIUM ACKNOWLEDGED

**Retest (2024-03-18)**

No recommendation has been implemented.
Team response:
*Overwriting values is preferred in case of improper address sent in airdrop activation and to avoid users sending multiple addresses to claim from.*

### Affected files

- AirdropTokenClaim.sol#L66
- SolanaAirdropTokenClaim.sol#L59

### Description

The present architecture of the `setAirdropClaims` function exhibits significant vulnerabilities and inefficiencies, rendering it not only expensive in terms of transaction costs, but also prone to numerous potential errors.

```
66        function setAirdropClaims(
67        address[] memory _beneficiaries,
68        uint256[] memory _amounts,
69        uint256 _airdropId
```

```
70      ) external onlyOwner {
71          require(
72              _beneficiaries.length == _amounts.length,
73              "Beneficiaries and amounts length mismatch"
74          );
75
76          for (uint256 i = 0; i < _beneficiaries.length; i++) {
77              _setAirdropClaim(_beneficiaries[i], _amounts[i], _airdropId);
78              emit AirdropClaimSet(_beneficiaries[i], _amounts[i], _airdropId);
79          }
80      }
```

```
88      function _setAirdropClaim(address _beneficiary, uint256 _amount, uint256
            _airdropId) internal {
89          airdropClaims[_airdropId][_beneficiary] = AirdropClaim({
90              beneficiary: _beneficiary,
91              amount: _amount,
92              claimed: false,
93              airdropId: _airdropId
94          });
95      }
```

Key issues include the risk of overwriting beneficiary data if an address is duplicated within the input array, as well as the possibility of double claims arising when initial values are incorrect and subsequent attempts are made to fix these entries. Given the common scenario where airdrops are distributed across multiple addresses, the execution of this function is likely to be exceptionally gas-intensive, thereby exacerbating its cost implications.

## Vulnerable scenario

The vulnerable scenario can occur in the following way:

① The beneficiary is put on the list twice because they performed the required task/action twice.
② The owner calls the `setAirdropClaims` function with the values from the generated list.
③ During the first `for` loop iteration, the `_amount` is set for the beneficiary address.
④ During the next `for` loop iteration, the `_amount` is overwritten for the same beneficiary address.

**Result of the attack:** Airdrop claims amounts instead of adding up, are overwritten.

**Recommendation**

- Given the centralized nature of this solution, it becomes unnecessary to manage `setAirdropClaims` directly on the blockchain. Instead, use a Merkle tree for orchestrating Airdrop Claims with off-chain validation mechanisms.
- Validate off-chain:
  - The cumulative amounts designated cannot surpass 100%,
  - Each beneficiary's address should be uniquely represented within the array,
  - The address(0) should be excluded as a beneficiary,
  - The amounts with 0 value should be excluded,
  - The length of beneficiaries and amounts arrays should be the equal,
  - Each airdrop should have its own unique identifier that integrity cannot be disturbed.
- Additionally, explicitly set if it should be active or inactive.
- The on-chain validation should be only for the verification of a beneficiary's eligibility to claim their airdrop, which should be efficiently executed via confirmation against the Merkle tree.

### References

1. SCSVS G1: Architecture, design and threat modeling

## [ARC-8aa78524-M05] Early withdrawal requires approval

MEDIUM FIXED

**Retest (2024-03-18)**

The vulnerability has been removed as recommended.

### Affected files

- ArcadeMissionPools.sol#L589

### Description

The `earlyWithdrawalFromMission` function requires the user to approve the mission pools contract to withdraw the funds even though it is not needed to transfer out the tokens.

**Result of the attack:** Blocking users from withdrawal until they make the unnecessary approval. In edge cases, when some party uses a contract as the MPC and they do not implement approval of the token, their funds would not be able to withdraw.

**Recommendation**

Remove the allowance check.

### References

1. SCSVS G5: Access control

## [ARC-8aa78524-M06] DoS via unbound loops and external calls

MEDIUM FIXED

**Retest (2024-03-18)**

The vulnerability has been removed.

- The contract now follows the pull-over-push pattern.
- The number of asset lenders is limited to 3 addresses and correctly tracked.
- The number of active investors is tracked and there are no more unbound loops.

### Affected files

- ArcadeMissionPools.sol#L485
- ArcadeMissionPools.sol#L515
- ArcadeMissionPools.sol#L762
- ArcadeMissionPools.sol#L777
- ArcadeMissionPools.sol#L714
- ArcadeMissionPools.sol#L1162

### Description

There are functions (see the Affected files section) that iterate over unbound loops and have multiple external calls. For example, the `closeMission` function. These unbound loops can lead to out-of-gas errors while external calls can revert. Both these cases lead to the revert of the main call. That would not allow for closing the mission.

Moreover, the fee distribution does not follow the pattern as well. This risk can materialize when tokens with callbacks are used and the fee receiver reverts.

**Result of the attack:** Denial of Service for selected functions.

**Recommendation**

- Limit the sizes of lists to protect from unbound loops.
- Follow the pull-over-push pattern that updates balances internally and requires all users (including lenders and fee receivers) to withdraw the funds with a separate call (like it is implemented for contributors).

### References

1. SCSVS G8: Denial of service

## [ARC-8aa78524-M07] Incorrect default mission status

MEDIUM ACKNOWLEDGED

**Retest (2024-03-18)**

The default status is still Funding status and the requirement of the signature is meant to protect from funding not started missions.

### Affected files

- ArcadeMissionPools.sol#L1229

### Description

Contributors can fund only those mission which pass the `isMissionFunding` modifier checks. However, the modifier checks whether the status of a mission is equal to zero, which is a default value.

```
1228    modifier isMissionFunding(uint256 _missionStatus) {
1229        require(_missionStatus == 0, "Mission is not in funding state");
1230        _;
1231    }
```

This means that all missions are in funding status by default, and any MPC can fund missions that were not created yet. The `fundMission` function does not require the amount of tokens to be greater than zero making it easier to block some functions, described in ARC-8aa78524-M06.

**Note:** This vulnerability requires to bypass the signature check, described in ARC-8aa78524-H02.

### Attack scenario

The attackers might take the following steps in turn to facilitate the DoS attack:

① Call `fundMission` many times with 0 amount for a mission that was not created yet to populate the `investors` mapping.
② Further call to the following functions would revert dues to looping over an unbound mapping:
- `withdrawLeftoverFunds`,
- `returnMissionPoolFunding`.

**Result of the attack:** Funding not yet created missions and facilitating an attack that would block calling some functions.

**Recommendation**

- Use non-default value for non-default statuses.
- Use ENUM types for statuses.
- Add default status, e.g. `NOT_STARTED`.

### References

1. SCSVS G5: Access control

## [ARC-8aa78524-M08] Incorrect handling of non-standard ERC20 tokens

MEDIUM ACKNOWLEDGED

**Retest (2024-03-18)**

No fee-on-transfer tokens are going to be accepted.

### Affected files

- ArcadeMissionPools.sol#L356
- ArcadeMissionPools.sol#L410
- ArcadeMissionPools.sol#L470

### Description

Using non-standard ERC20 tokens, such as **fee-on-transfer tokens**, can lead to inability to withdraw the funds for contributors who withdraw last.

### Vulnerable scenario

Faulty operation scenario with assumption that there are no Arcade fees for simplicity:

① The MPO creates a mission with an approved token that is fee-on-transfer token with fee 1%.
② Multiple contributors call `fundMission` function 100 times with 1000 tokens each, resulting in 99000 tokens on the contract.
③ The mission is finished without success and contributors can start withdrawing their funds.
④ 99 out of 100 contributors are able to withdraw their funds, 1000 tokens each.
⑤ The last contributor is not able to withdraw their 1000 tokens, because the mission pools contract is empty.

**Result of the attack:** Inability to withdraw tokens by MPC who withdraw last.

**Recommendation**

- If you plan to support non-standard ERC20 tokens, consider one of the following:
  - Check the balance **before** and **after** the transfer.
  - Another solution might be to validate whether the actual transferred amount is the same as the passed amount parameter.
- If you do not plan to support non-standard ERC20 tokens, create a policy to verify used tokens.

### References

1. SCSVS I2: Token
2. SCSVS G4: Business Logic

## [ARC-8aa78524-M09] Locking reward via multiple calls to closeMission

MEDIUM FIXED

**Retest (2024-03-18)**

The vulnerability has been removed as recommended.

### Affected files

- ArcadeMissionPools.sol#L442

### Description

The `closeMission` function can be called multiple times as there is no validation of the current status. If the MPO calls it for the first time with the reward greater than zero, the reward is pulled from them. Then, when the `closeMission` function is called again with no reward (e.g. called by mistake by the operator or the admin) the mission's reward is cleared in the mapping, but the tokens remain on the mission pool contract.

### Vulnerable scenario

The vulnerable scenario can occur in the following way:

① The operator closes the mission with reward equal to 100 tokens.
② The contract pulls 100 tokens from MPC and stores the reward amount in `missionDebriefs` mapping.
③ The operator or the admin calls again the `closeMission` function for the same mission identifier but with zero reward.
④ The contract updates the reward with zero amount.
⑤ The contributors cannot withdraw the initial reward (100 tokens).

**Result of the attack:** Reward tokens are locked for MPCs.

**Recommendation**

Validate whether the mission being closed has not been closed already.

### References

1. SCSVS G5: Access control

## [ARC-8aa78524-M10] Swap fee non-compliant with the documentation

MEDIUM FIXED

**Retest (2024-03-18)**

The vulnerability has been removed as recommended.

### Affected files

- ArcadeSwap.sol#L47

### Description

The white paper describes the situations where the platform fee for swaps between ARC and xARC are different depending on the direction. However, there is one fee rate variable which means that it cannot be different for both directions.

**Result of the attack:** Impossibility to specify different swap fees depending on the swap direction.

**Recommendation**

Define two variables to store swap fees for both swap directions.

### References

1. SCSVS G1: Architecture, design and threat modeling

## [ARC-8aa78524-M11] Excessively powerful owner

MEDIUM ACKNOWLEDGED

**Retest (2024-03-18)**

The team is planning to follow the recommendations.

### Affected files

- AirdropTokenClaim.sol#L43
- AirdropTokenClaim.sol#L101
- AirdropTokenClaim.sol#L110
- SolanaAirdropTokenClaim.sol#L39
- BaseTokenVault.sol#L30
- BaseTokenVault.sol#L40
- AdvisoryVault.sol#L19
- ArcadeTeamVault.sol#L19
- CommunityVault.sol#L25
- SolanaClaimVault.sol#L25
- TokenDistributionVault.sol#L25
- VestingTokenClaim.sol#L39
- ArcadeToken.sol#L117
- ArcadeToken.sol#L131
- ArcadeToken.sol#L149
- XArcadeToken.sol#L133

- XArcadeToken.sol#L147
- XArcadeToken.sol#L165
- XArcadeToken.sol#L174
- ArcadeSwap.sol#L251
- ArcadeSwap.sol#L265
- ArcadeSwap.sol#L281
- ArcadeSwap.sol#L297
- ArcadeMissionPools.sol#L956
- ArcadeMissionPools.sol#L985
- ArcadeMissionPools.sol#L1006

## Description

The current implementation of the system is highly centralized. Roles such as Owner and `DEFAULT_ADMIN_ROLE` have significant privileges that allow them to call functions that have a strong impact on operation and changes are made immediately.

In some cases, this allows for complete takeover of the user's tokens. Although the team plans to use multisig, which significantly reduces the threat, it is not an ideal solution that fully protects the user. Functions that are powerful and heavily impact business logic are listed in the Affected files section.

**Note:** The severity of the vulnerability has been downgraded to MEDIUM. This is predicated on the understanding that the protocol is intentionally centralized. However, it is crucial to acknowledge that incidents of rug pulls have eroded trust in centralized DeFi projects. Consequently, excessive power within these functionalities may act as a deterrent to prospective users.

**Result of the attack:** Immediate changes of critical parameters that in worst case could lead to theft of contributors' tokens or Denial of Service.

**Recommendation**

**Short-term:** Continue the planned use of multisig. Use timelocks to give users time to react to upcoming changes. Add limits on functions that support fees. Communicate the threats coming from centralization in the documentation.
**Long-term:** Propose a more decentralized approach or delegate ownership to DAO.

## References

1. SCSVS G1: Architecture, design and threat modeling

# [ARC-8aa78524-L01] Not following the Checks-Effects-Interactions pattern

LOW FIXED

**Retest (2024-03-25)**

The vulnerability has been removed as recommended.

## Affected files

- ArcadeMissionPools.sol#L490
- ArcadeMissionPools.sol#L515
- ArcadeMissionPools.sol#L534
- ArcadeMissionPools.sol#L575
- ArcadeMissionPools.sol#L610
- AirdropTokenClaim.sol#L55-56
- SolanaAirdropTokenClaim.sol#L49-50

## Description

Selected functions (see the Affected files section) do not follow the Checks-Effects-Interactions pattern. Its main idea is to first execute all checks to make sure the function call is legitimate, then all updates (e.g. decreasing balance), and at the end interactions (e.g. calls to external contracts).

There are `nonReentrant` modifiers added to functions in `ArcadeMissionPools` contract that protect from re-entrancy. Not using the CEI pattern leads to a situation where the contract's state might be invalid during external calls.

In the case of airdrop contracts, if the vault contract or the vault token had a callback to a user-controlled address, the attacker could drain the vault.

## Attack scenario

The attackers might take the following steps in turn:

① Become a legitimate airdrop receiver.
② The airdrop contract is deployed and the attacker's address is added to the airdrop.
③ Call the `claim` function to claim their tokens.
- When the airdrop contract calls the transfer function, the token calls back the receiver's address.
- The attacker calls the `claim` function again (re-entrancy call) that passes the `airdropClaims`

`[msg.sender].claimed` check and sends tokens again.

**Result of the attack:** Non-compliance with the pattern can lead to such bugs like stealing tokens via re-entrancy. There is a scenario that could allow attackers to drain airdrop vaults with ARC or xARC tokens.

**Recommendation**

Make sure that interactions are implemented after checks and effects.

### References

1. SCSVS G6: Communications

## [ARC-8aa78524-L02] Incorrect assignment of beneficiaries to the vault

LOW ACKNOWLEDGED

**Retest (2024-03-18)**

No recommendation has been implemented.
Team response:
*Costly to re-architect and deploy the vaults from a platform interaction perspective and minting new vaults contracts as our vaults have been created and minted to.*

### Affected files

- VestingTokenClaim.sol#L50-L65

### Description

Each vault is deployed separately to manage funds for a specific beneficiary type. Thise vaults are managed by separately deployed `VestingTokenClaim` instances in the following schema:

```
ArcadeTeamVault -> VestingTokenClaim 1, beneficiary type Team
AdvisoryVault -> VestingTokenClaim 2, beneficiary type Advisor
TokenDistributionVault -> VestingTokenClaim 3, beneficiary types Seed, Private,
    Public, Strategic
```

Only one `Vault` can be assigned to each `VestingTokenClaim`, but `setVestingSchedules` allows to assign any `beneficiaryTypes`. That introduces risk of mismatched assignment of

beneficiaries and claiming funds from incorrect vault i.e. advisory could claim tokens from `ArcadeTeamVault` instead of `AdvisoryVault`.

**Recommendation**

- Validate off-chain the appropriate beneficiary type when configuring vesting schedules for specific `VestingTokenClaim` contract.
- Remove the `_beneficiaryType` parameter from the `setVestingSchedules` function because if you implemented recommendation from issue ARC-8aa78524-L04 you would not need it anymore.

### References

1. SCSVS G1: Architecture, design and threat modeling

## [ARC-8aa78524-L03] The FundingEnded event is not emitted

LOW FIXED

**Retest (2024-03-18)**

The `FundingEnded` event has been removed globally.

### Affected files

- ArcadeMissionPools.sol#L421

### Description

The `FundingEnded` event is emitted at the end of `endMissionTopOffFunding` function to communicate that the funding process of a mission has ended.

However, the funding process of a mission can end when the last contributor funds the remaining amount of tokens to reach `poolTarget`. This is achieved with `fundMission` function, but it does not emit the `FundingEnded` event.

**Result of the attack:** No emission of the `FundingEnded` event.

**Recommendation**

Check whether the contributor funds the remaining amount and if so, emit the event.

### References

1. SCSVS G4: Business Logic

## [ARC-8aa78524-L04] Full control over rewards by MPO

LOW ACKNOWLEDGED

**Retest (2024-03-18)**

There has been added a limit of lenders that can be assigned to a mission.
Additionally, the `addAssetLender` function is protected with the backend signature verification.
However, the recommendation regarding the minimal reward has not been implemented.

### Affected files

- ArcadeMissionPools.sol#L442
- ArcadeMissionPools.sol#L806

### Description

The MPO is able to add multiple lenders when the mission is in funding status. This allows the operator to spread the lenders reward on an unverified list of alleged lenders.

Moreover, MPO controls the value of the reward when closing the mission. This is a big disincentive for contributors to fund the mission.

**Result of the attack:** Lack of rewards for contributors or lenders.

**Recommendation**

- Allow adding an asset lender to a mission only before the financing of the mission begins.
- Consider requiring to specify the final or at least minimal reward value when the mission is created by the operator.

### References

1. SCSVS G4: Business Logic

# [ARC-8aa78524-L05] Costly and unnecessarily complicated vault architecture

LOW ACKNOWLEDGED

**Retest (2024-03-18)**

No recommendation has been implemented.

**Team response:**
*Costly to re-architect and deploy the vaults from a platform interaction perspective and minting new vaults contracts as our vaults have been created and minted to.*

## Affected files

- VestingTokenClaim.sol
- VestingTokenClaimCore.sol
- AdvisoryVault.sol
- ArcadeTeamVault.sol
- CommunityVault.sol
- GeneralReserveVault.sol
- SolanaClaimVault.sol
- TokenDistributionVault.sol

## Description

The current vault implementation is unnecessarily complicated and expensive to use. Separating funds between different vaults only makes sense if they have different business logic, different owner or different parameters. Otherwise, separation only unnecessarily increases usage costs and enlarges the codebase.

In the case of the current implementation, if a vulnerability or problem occurs, it will affect all vaults. Vaults currently differ mainly in naming. Additionally, the `GeneralReserveVault` does not have any `claim` function. Tokens can be withdrawn using only the `rescue` function which serves another purpose.

**Result of the attack:** Complicated vault management, unnecessarily large codebase requiring maintenance.

**Recommendation**

- Remove `VestingTokenClaimCore`.
- Inside the `VestingTokenClaim` constructor create selected vaults, passing them appropriate `CLIFF` and `DURATION` values. This way, each vault will be started with different values. Consider how this will affect the rest of the business logic and verify the correct operation of contracts after introducing changes using tests.
- Add claim functions to the `GeneralReserveVault` if you plan to keep this contract.
- Since vaults have very similar implementations and essentially do exactly the same thing, it is proposed to unify their implementations. To differentiate, you can add an additional parameter with a name or simply base it on the address instead of creating separate files.

### References

1. SCSVS G1: Architecture, design and threat modeling

## [ARC-8aa78524-L06] Lack of 2 step ownership transfer

LOW ACKNOWLEDGED

**Retest (2024-03-18)**

No recommendation has been implemented.

### Affected files

- AirdropTokenClaim.sol
- SolanaAirdropTokenClaim.sol
- BaseTokenVault.sol
- VestingTokenClaim.sol
- XArcadeToken.sol#L35
- ArcadeToken.sol#L9

### Description

Some of the vault contracts inherit from `Ownable` contract which allows to instantly transfer ownership to any address except `address(0)`. In case of invalid address passed as the new owner (e.g. a contract that cannot make calls to protected functions and the `transferOwnership` function) there is no way to get the ownership back.

The `ArcadeToken` and `XArcadeToken` contracts have similar, but a bit different issue. The `ArcadeToken` inherits from `AccessControl` contract but uses only one role. The `XArcadeToken`

contract has a custom variable `contractAdmin` that can be updated with `updateContractAdmin` function. In both cases we recommend to use `Ownable2Step` contract.

**Result of the attack:** Loss of the ownership for the contract.

**Recommendation**

- Use `Ownable2Step` contract instead of `Ownable` or custom implementation.
- Consider overriding the `renounceOwnership` function to make it always revert if you do not plan to do it
- Additionally, consider using `Ownable2Step` contract instead of `AccessControl` for `ArcadeToken` or create a new role for `ArcadeSwapContract`.

### References

1. Ownable2Step.sol
2. SCSVS G5: Access control

## [ARC-8aa78524-L07] Multiple entries with the same investor

LOW ACKNOWLEDGED

**Retest (2024-03-18)**

The recommendation has not been implemented.

**Team response:**
*Noted for long term mission pool iteration fixes.*

### Affected files

- ArcadeMissionPools.sol#L359
- ArcadeMissionPools.sol#L413

### Description

When a contributor funds a mission multiple times, their address is added to the `investors` struct multiple times as well.

**Result of the attack:** Increase of the gas consumption that in edge cases can lead to out-of-gas errors (described in ARC-8aa78524-M07).

**Recommendation**

Use an `EnumerableSet` instead of a mapping to store investors.

### References

1. EnumerableSet.sol
2. SCSVS G4: Business Logic

## [ARC-8aa78524-L08] No event emitted in important state-changing functions

LOW FIXED

**Retest (2024-03-18)**

The vulnerability has been removed as recommended.

### Affected files

- ArcadeMissionPools.sol#L974
- ArcadeMissionPools.sol#L988

### Description

The following functions in mission pools contract do not emit any event when the important storage variables are updated:

- `setPlatformFeeVault`,
- `setSignatory`.

**Result of the attack:** Impossibility to track events for important functions.

**Recommendation**

Emit events in all important state-changing functions.

### References

1. SCSVS G1: Architecture, design and threat modeling

# [ARC-8aa78524-L09] The vestingSchedules function lacks appropriate validation

LOW ACKNOWLEDGED

**Retest (2024-03-18)**

The team decided to follow recommendation of off-chain validation for input parameters.

## Affected files

- VestingTokenClaim.sol#L50

## Description

Current implementation of the `setVestingSchedules` do not perform sufficient validation of input parameters.

```
50    function setVestingSchedules(
51        address[] memory _beneficiaries,
52        uint256[] memory _amounts,
53        BeneficiaryType _beneficiaryType
54    ) external onlyOwner {
55        require(
56            _beneficiaries.length == _amounts.length,
57            "Beneficiaries and amounts length mismatch"
58        );
59
60        VestingScheduleParams memory vestingScheduleParams =
              _getVestingScheduleParams(
61            _beneficiaryType
62        );
63
64        _setVestingSchedules(_beneficiaries, _amounts, vestingScheduleParams,
              _beneficiaryType);
65    }
```

That might introduce unnecessary issues like inability to vest by one of the beneficiaries or redundant gas consumption.

## Vulnerable scenario

The vulnerable scenario can occur in the following way:

① The total amounts allocated to beneficiaries exceed the value stored in the vault.
② Beneficiaries, despite the amount assigned to them, will not be able to claim the funds if 100% of the content is claimed earlier.

**Result of the attack:** Redundant gas consumption and inability to vest by some of the beneficiaries.

**Recommendation**

- Introduce off-chain validation for input parameters. Validate off-chain:
  - Each beneficiary's address should be uniquely represented within the array.
  - The address(0) should be excluded as a beneficiary.
  - The amounts with 0 value should be excluded.
  - The cumulative amounts designated cannot surpass 100%.
  - The length of beneficiaries and amounts arrays should be the equal.

### References

1. SCSVS G1: Architecture, design and threat modeling

## [ARC-8aa78524-L10] Invalid use of transferFrom

LOW FIXED

**Retest (2024-03-25)**

The vulnerability has been removed as recommended.

### Affected files

- ArcadeSwap.sol#L179
- ArcadeSwap.sol#L224
- ArcadeToken.sol#L122
- XArcadeToken.sol#L139

### Description

The `ArcadeSwap` contract uses `transferFrom` function to transfer out tokens. This requires the self-approval which is senseless.

**Result of the attack:** Use of functions contrary to their intended purpose, which requires additional operations.

**Recommendation**

- Use the `safeTransfer` function (from `SafeERC20` library) to transfer tokens from the contract that calls this function.
- Remove calls to `_approve` and `increaseAllowance` functions that approves the `ArcadeSwap` contract to transfer its tokens (ArcadeToken.sol#L122, XArcadeToken.sol#L139).

### References

1. SafeERC20.sol
2. SCSVS I2: Token

## [ARC-8aa78524-L11] Lock of Ether via payable functions

LOW FIXED

**Retest (2024-03-18)**

The vulnerability has been removed as recommended.

### Affected files

- ArcadeSwap.sol#L153
- ArcadeSwap.sol#L198

### Description

The `ArcadeSwap` contract contains functions that are `payable` and can accept mistakenly sent Ether. On the other hand there is no way to withdraw it.

**Result of the attack:** Lock of Ether sent mistakenly via the `payable` functions.

**Recommendation**

Remove `payable` modifier from functions.

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-L12] Temporal block of OTF channel

LOW ACKNOWLEDGED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

## Affected files

- XArcadeToken.sol#L9

## Description

The `XArcadeToken` contract inherits from `OFTV2` contract that is an implementation of Omni Fungible Token by LayerZero, build on top of `NonblockingLzApp` contract. The default implementation of `_blockingLzReceive` function in `NonblockingLzApp` forwards all gas left to `nonblockingLzReceive`. Next, the `nonblockingLzReceive`, in case of `OFTV2`, calls `onOFTReceived` on destination contract and sets gas equal to the adapter parameter (controlled by the user).

The attacker can easily use that flow to drain gas and revert within `onOFTReceived` function, leaving too less gas to store failed message (OFTCoreV2.sol#L221 and NonblockingLzApp.sol#L36). It ends up in storing payload on the Endpoint level and blocking the channel temporarily. The Arcade team would have to make a call on `XArcadeToken` contract to unblock the channel.

Here are the conditions that have to be met to execute that malicious flow on the base of OFT:

- OFT must allow `PT_SEND_AND_CALL` message type (set `minDstGasLookup` greater than 0),
- OFT must allow to specify any address as the receiver (met in the default OFT implementation),
- OFT must allow to specify adapter parameters (met in the default OFT implementation).

**Result of the attack:** Temporal block of cross-transferring of `xARC` between specific chains.

**Recommendation**

- If you do not plan to support cross-transferring the token with a call to `onOFTReceived`, do not set `minDstGasLookup` greater than zero for `PT_SEND_AND_CALL` message type.
- Otherwise, you can implement one of the following:
  - Reserve some gas in `_sendAndCallAck` function (in `OFTCoreV2` contract) for the possible need of storing failed message. That would require to change the source code of LZ's example contracts.
  - Monitor channel blocks and automatically unblock it using dedicated function `retryMessage`. This is a mitigation recommended by LZ team.

### References

1. SCSVS G4: Business Logic

## [ARC-8aa78524-L13] Invalid check for max token supply when swapping

LOW FIXED

**Retest (2024-03-18)**

The vulnerability has been removed as recommended.

### Affected files

- ArcadeSwap.sol#L156
- ArcadeSwap.sol#L201

### Description

Functions `swapArcadeToXArcade` and `swapXArcadeToArcade` check whether the supply of both tokens does not exceed the `MAX_SUPPLY` after the swap. However, the swap changes the supplies of tokens by the same value which means that their sum does not change.

In short, if the sum of tokens was correct before the swap, it must be correct after the swap.

**Result of the attack:** Ineffective validation that always pass.

**Recommendation**

Remove the check for max token supply.

### References

1. SCSVS G4: Business Logic

## [ARC-8aa78524-L14] Lack of safeTransfer

LOW FIXED

**Retest (2024-03-26)**

The vulnerability has been removed as recommended.

### Affected files

- BaseTokenVault.sol#L31
- BaseTokenVault.sol#L41
- AdvisoryVault.sol#L51
- ArcadeTeamVault.sol#L51
- CommunityVault.sol#L56
- TokenDistributionVault.sol#L57
- SolanaClaimVault.sol#L57
- ArcadeMissionPools.sol#L605
- ArcadeMissionPools.sol#L610
- ArcadeMissionPools.sol#L724
- ArcadeMissionPools.sol#L1095
- ArcadeMissionPools.sol#L1143
- ArcadeMissionPools.sol#L1170

### Description

Although the main token contract to be used is `xArcadeToken` or `ArcadeToken`, the contracts can accept other tokens that could not revert on invalid transfers, but return `false`.

The mission pool and vault contracts would not detect that and assumed that the transfer was correct.

**Result of the attack:** Contracts cannot detect failed transfers.

**Recommendation**

Use the `safeTransfer` function (from `SafeERC20` library) to transfer tokens from the contract that calls this function.

### References

1. SafeERC20.sol
2. SCSVS I2: Token

## [ARC-8aa78524-L15] Use of dependencies with known vulnerabilities

LOW ACKNOWLEDGED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

## Affected files

- package.json
- package.json

## Description

The Arcade project uses smart contract dependencies that have known vulnerabilities:

- @openzeppelin/contracts "4.9.2"
- @openzeppelin/contracts-upgradeable "4.9.2"
- @openzeppelin/contracts "3.4.2" (required by @layerzerolabs/lz-evm-sdk-v1-0.7)

**Note:** The severity of the vulnerability has been decreased because the known vulnerabilities do not impact the Arcade project.

**Result of the attack:** Potential introduction of known vulnerabilities in the project.

**Recommendation**

- Update versions of dependencies to the latest versions. In case the major or minor version cannot be updated, use the latest patch version.
- Use `npm audit` to check whether the dependencies have known vulnerabilities. It covers all dependencies, not only smart contract packages.

## References

1. SCSVS G1: Architecture, design and threat modeling

# 6. Recommendations

## [ARC-8aa78524-R01] Consider using the latest solidity version

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

### Description

In accordance with the best security practices, it is recommended to use the latest stable versions of major Solidity releases. Very often, older versions contain bugs that have been discovered and fixed in newer versions.

Moreover, it is worth remembering that the version should be clearly specified so that all tests and compilations are performed with the same version.

**Recommendation**

If it is planned to deploy on multiple chains, stay aware that some of them don't support `PUSH0` opcode. If `solc >=0.8.20` is used, the `PUSH0` opcode will be present in the bytecode. In this situation, it is recommended to choose `0.8.19`.

```
pragma solidity 0.8.19;
```

### References

1. SCSVS V1: Architecture, design and threat modeling
2. Floating pragma SWC-103

## [ARC-8aa78524-R02] Unnecessary expensive claim function

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

### Description

The beneficiary can claim their vested tokens through the `claim` function. Based on their `vestingSchedules`, the `claimableAmount` is calculated and then released to the `msg.sender`.

However, the implementation unnecessarily does external calls in every `for` loop iteration making it costly to use.

```
70 function claim() external {
71     VestingSchedule[] storage beneficiaryVestingSchedules = vestingSchedules[msg.
       sender];
72     require(beneficiaryVestingSchedules.length > 0, "No vesting schedules found")
       ;
73
74     for(uint256 i = 0; i < beneficiaryVestingSchedules.length; i++) {
75         VestingSchedule storage vestingSchedule = beneficiaryVestingSchedules[i];
76         require(vestingSchedule.claimedAmount < vestingSchedule.totalAmount, "
           Nothing to claim");
77         uint256 claimableAmount = _getClaimableAmount(vestingSchedule);
78         require(claimableAmount > 0, "Nothing to claim");
79         vestingSchedule.claimedAmount += claimableAmount;
80         vault.release(msg.sender, claimableAmount);
81         emit Claimed(msg.sender, claimableAmount);
82     }
83 }
```

**Recommendation**

- Calculate the `totalClaimableAmount` inside `for` loop and do external call to the vault after the `for` loop.
  - VestingTokenClaim.sol#L84

### References

1. SCSVS V1: Architecture, design and threat modeling

## [ARC-8aa78524-R03] Consider new logic for setVaultAddress function

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

### Description

The current implementation requires the owner to call not only the `setVaultAddress` function in vesting and airdrop contracts, but also the corresponding function on the other contract (`setAirdropTokenClaimAddress` or `setVestingTokenClaimAddress`). Otherwise, the new vault does not work and release operations are reverted.

**Recommendation**

Consider implementation where calling one function will set all necessary parameters and do not break functionality. This can be achieved through abstracting all the management functions to the separate contract.

### References

1. SCSVS V1: Architecture, design and threat modeling

## [ARC-8aa78524-R04] Follow the layout order from Solidity Style Guide

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

### Description

Ordering helps to keep code clarity and allow readers to navigate easily. Currently the layout order do not follow the Solidity Style Guide.

**Recommendation**

Functions should be grouped according to their visibility and ordered in the following format:

```
constructor
receive function (if exists)
fallback function (if exists)
```

```
external
public
internal
private
```

Within a grouping, place the `view` and `pure` functions last.

### References

1. SCSVS G11: Code clarity
2. Solidity Style Guide

## [ARC-8aa78524-R05] Remove reduntant checks

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation has not been implemented.

### Description

There are unnecessary requirements inside the contract that are checked elsewhere or are not achievable. Removing them will make the codebase smaller and reduce the cost of calling the function.

**Recommendation**

- Remove the following code snippets
  - VestingTokenClaim.sol#L136-L138: checked in the next `if` clause.
  - ArcadeToken.sol#L103: reverts if `_amount` is greater than `totalSupply()`.
  - XArcadeToken.sol#L119: reverts if `_amount` is greater than `totalSupply()`.

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-R06] Reorder storage variables

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

## Description

The details of mission are kept in multiple storage variables. That complicates the source code and is error prone. It is recommended to keep all the details in `Mission` struct and in one missions mapping.

Notes:

- Starting from solidity 0.7.0, the structs containing (nested) mappings must have storage as data location.
- It is not a security bug, but rather a recommendation to make the code clearer.

**Recommendation**

Consider adding the following variables to the mission struct and have only one mapping for missions in the contract:

- `missionDistributions`
- `missionDebriefs`
- `investors`
- `invested`
- `investments`
- `investorCount`
- `assetLenderAddresses`
- `assetLenders`
- `assetLenderCount`
- `assetLenderWithdrawn`
- `mpoWithdrawn`
- `adminWithdrawn`

Consider using `EnumerableSet` instead of nested mapping for the following variables:

- `investors`
- `invested`
- `assetLenderAddresses`
- `assetLenders`
- `assetLenderWithdrawn`
- `mpoWithdrawn`
- `adminWithdrawn`
- `approvedTokens`

Consider removing the following variables as their value could be taken as the size of `EnumerableSet` or from other variables:

- `invested`
- `investorCount`
- `assetLenderCount`

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-R07] Do not duplicate events

INFO IMPLEMENTED

**Retest (2024-03-18)**

Recommendation has been implemented.

### Description

There are `RoleGranted` and `RoleRevoked` events in `MissionEvents` library. Those events are also emitted when `AccessControl` contract emits the same contracts.

**Recommendation**

Use existing events from `AccessControl` contract.

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-R08] Remove reduntant code

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation has been implemented partially.
The event is emitted and `missionDebriefs` mapping is updated outside the `if` statement.
The `_reward` parameter is not checked in case of failed mission.

### Description

The emission of `MissionClosed` event and update of `missionDebriefs` mapping in `closeMission` function is copied in two `if` branches and could be moved out of the `if` clause.

**Recommendation**

- Emit the event outside the `if` clause.
- Reconsider whether the `_reward` parameter can be greater than zero when the mission is failed and no tokens are pulled from the operator. We recommend to clear it in case of failed mission.
- Update the `missionDebriefs` mapping outside the `if` clause.

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-R09] Remove unused code

INFO IMPLEMENTED

**Retest (2024-03-18)**

Recommendation has been implemented. The only difference was that now `XArcadeFeesForward` event is emitted.

### Description

There exist fragments of code that are never used and should be removed.

**Recommendation**

- The `isMissionSuccessful` modifier is never used.
- The `CONTRACT_ADMIN_ROLE` is never used.
- The `XArcadeFeesForward` event is never emitted.

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-R10] Save values in temporal memory variables

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

### Description

The result of a call to the `current` function from the `Counters` library should be stored in a temporal memory variable to save gas and increase the code clarity.

**Recommendation**

Store the following occurrences in temporal variables:

- ArcadeMissionPools.sol#L481
- ArcadeMissionPools.sol#L485

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-R11] Use ENUMs instead of hardcoded numeric values

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

### Description

The variables such as statuses should have ENUM values instead of numeric ones. The ENUM types are self-explanatory.

**Recommendation**

Use ENUM type for the `status` variable.

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-R12] Use custom errors

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

### Description

Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

**Recommendation**

Use Custom Error in the following places:

- ArcadeMissionPools.sol#L211
- ArcadeMissionPools.sol#L232
- ArcadeMissionPools.sol#L436
- ArcadeMissionPools.sol#L1038
- ArcadeMissionPools.sol#L1129
- ArcadeMissionPools.sol#L1139
- ArcadeMissionPools.sol#L1149
- ArcadeMissionPools.sol#L1159
- ArcadeSwap.sol#L160
- ArcadeSwap.sol#L205
- ArcadeToken.sol#L38
- ArcadeToken.sol#L82
- ArcadeToken.sol#L135
- ArcadeToken.sol#L139
- ArcadeToken.sol#L140
- XArcadeToken.sol#L55
- XArcadeToken.sol#L99
- XArcadeToken.sol#L152
- XArcadeToken.sol#L156
- XArcadeToken.sol#L157
- XArcadeToken.sol#L185

### References

1. Custom Errors in Solidity
2. SCSVS G11: Code clarity

## [ARC-8aa78524-R13] Use hash of role names

INFO IMPLEMENTED

**Retest (2024-03-18)**

Recommendation has been implemented.

### Description

Do not use numeric values for roles. Use `keccak256` of the role name to avoid potential collisions.

**Recommendation**

Use `keccak256` to generate value for `ROLE_MISSION_POOL_OPERATOR`:

```
29 bytes32 public immutable ROLE_MISSION_POOL_OPERATOR = keccak256("
       ROLE_MISSION_POOL_OPERATOR");
```

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-R14] Use reward to determine the result of a mission

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

### Description

The `_reward` parameter in `closeMission` function could be used to determine the result of a mission: if the reward is greater than zero, the mission was successful.

Additionally, currently the MPO can specify `_result` equal to `false` and `_reward` greater than zero. That leads to a situation when the reward is not pulled from the operator and not distributed but the `missionReward` variable is greater than zero.

**Recommendation**

- Remove `_result` parameter and assume that mission was successful when the `_reward` is greater than 0.
- Reconsider whether the `_reward` parameter can be greater than zero when the mission is failed and no tokens are pulled from the operator. We recommend to clear it in case of failed mission.

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-R15] Keep validation checks consistent

INFO IMPLEMENTED

**Retest (2024-03-18)**

Recommendation has been implemented.

### Description

The mission pools contract checks during its initialization whether the `SIGNATORY` address is not zero. However, it does not check it in `setSignatory` function.

**Recommendation**

Check on every update of the `SIGNATORY` variable whether the address parameter is not zero.

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-R16] Use packages for external contracts

INFO NOT IMPLEMENTED

**Retest (2024-03-18)**

Recommendation acknowledged for future iterations.

### Description

Do not copy-paste LayerZero files from external "solidity-examples" repository. Instead, add the package to dependencies and import them from the package.

**Recommendation**

Add the "@layerzerolabs/solidity-examples" to your project.
**Note:** if you plan to change the source code of `OFTCoreV2` contract (as mentioned in ARC-8aa78524-L12), you will have to copy-paste this one file.

### References

1. SCSVS G11: Code clarity

## [ARC-8aa78524-R17] Insecure upgradeability

INFO IMPLEMENTED

**Retest (2024-03-18)**

Recommendation has been implemented. The team decided to remove upgradeability.

### Description

The `ArcadeMissionPools` and `ArcadeSwap` contracts are built as upgradeable contracts. They inherit from `Initializable` and implement `initialize` function. However, they also inherit from contracts that are not compatible with upgradeability (`AccessControl`, `ReentrancyGuard`). Additionally, the deployment script does not use proxies.

Some non-upgradeable contracts can break proxies because of functions that use storage variables defined in the constructor (not called by the proxies).

**Recommendation**

- If you plan to support upgradeability, inherit from upgradeability compatible contracts (`AccessControlUpgradeable`, `ReentrancyGuardUpgradeable`).
- If you do not plan to support upgradeability, remove `Initializable` inheritance and `initialize` function (move its code to constructor).

## References

1. SCSVS G1: Architecture, design and threat modeling

# 7. Impact on risk classification

Risk classification is based on the one developed by OWASP [1], however it has been adapted to the immutable and transparent code nature of smart contracts. The Web3 ecosystem forgives much less mistakes than in the case of traditional applications, the servers of which can be covered by many layers of security.

Therefore, the classification is more strict and indicates higher priorities for paying attention to security.

| OVERALL RISK SEVERITY | | | | |
|---|---|---|---|---|
| Impact on risk | HIGH | CRITICAL | HIGH | MEDIUM |
| | MEDIUM | MEDIUM | MEDIUM | LOW |
| | LOW | LOW | LOW | INFO |
| | | HIGH | MEDIUM | LOW |
| | | Likelihood | | |

[1] OWASP Risk Rating methodology

# 8. Long-term best practices

## 8.1. Use automated tools to scan your code regularly

It's a good idea to incorporate automated tools (e.g. slither) into the code writing process. This will allow basic security issues to be detected and addressed at a very early stage.

## 8.2. Perform threat modeling

Before implementing or introducing changes to smart contracts, perform threat modeling and think with your team about what can go wrong. Set potential targets of the attacker and possible ways to achieve them, keep it in mind during implementation to prevent bad design decisions.

## 8.3. Use Smart Contract Security Verification Standard

Use proven standards to maintain a high level of security for your contracts. Treat individual categories as checklists to verify the security of individual components. Expand your unit tests with selected checks from the list to be sure when introducing changes that they did not affect the security of the project.

## 8.4. Discuss audit reports and learn from them

The best guarantee of security is the constant development of team knowledge. To use the audit as effectively as possible, make sure that everyone in the team understands the mistakes made. Consider whether the detected vulnerabilities may exist in other places, audits always have a limited time and the developers know the code best.

## 8.5. Monitor your and similar contracts

Use the tools available on the market to monitor key contracts (e.g. the ones where user's tokens are kept). If you have used code from another project, monitor their contracts as well and introduce procedures to capture information about detected vulnerabilities in their code.


## Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com



## Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com