SALUS SECURITY

MAR 2023

# CODE SECURITY ASSESSMENT

FUSIONIST

# Overview

## Project Summary

- Name: Fusionist
- Version: commit db4eefa
- Platform: Endurance Chain
- Language: Solidity
- Repository:  https://github.com/FusionistGame/contract_for_audit
- Audit Range: See Appendix - 1

# Project Dashboard

## Application Summary

| | |
|---|---|
| **Name** | Fusionist |
| **Version** | v1 |
| **Type** | Solidity |
| **Dates** | Mar 07 2023 |
| **Logs** | Mar 07 2023 |

## Vulnerability Summary

| | |
|---|---|
| **Total High-Severity issues** | 0 |
| **Total Medium-Severity issues** | 3 |
| **Total Low-Severity issues** | 6 |
| **Total informational issues** | 6 |
| **Total** | 15 |

## Contact

E-mail: support@salusec.io

# Risk Level Description

| | |
|---|---|
| **High Risk** | The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users. |
| **Medium Risk** | The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact. |
| **Low Risk** | The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances. |
| **Informational** | The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth. |

# Content

# Introduction

## 1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

## 1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

## 1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

# Findings

## 2.1 Summary of Findings

| ID | Title | Severity | Category | Status |
|---|---|---|---|---|
| 1 | The parameters defined in verifySig() are inconsistent with the ones used | Medium | Business Logic | Unresolved |
| 2 | A user making a new request when the last request has not been responded to will cause any response to the user's request to fail | Medium | Business Logic | Unresolved |
| 3 | Centralization risk | Medium | Centralization | Unresolved |
| 4 | Inaccurate boundary conditions in sanity checks | Low | Business Logic | Unresolved |
| 5 | Incorrect value for UPGRADER_ROLE role | Low | Configuration | Unresolved |
| 6 | Missing events for governor only functions that change critical parameters | Low | Auditing and Logging | Unresolved |
| 7 | Inconsistency between signature verification and documentation | Low | Configuration | Unresolved |
| 8 | Mismatch between comment and implementation | Low | Business Logic | Unresolved |
| 9 | Unnecessary grantRole operation in setSignerAddress() | Low | Business Logic | Unresolved |
| 10 | Could add an amount parameter to the EClaimSuc() events | Informational | Auditing and Logging | Unresolved |
| 11 | Spelling mistakes | Informational | Code Quality | Unresolved |
| 12 | Redundant code and comments | Informational | Redundancy | Unresolved |
| 13 | Inconsistent role design | Informational | Code Quality | Unresolved |
| 14 | Inconsistent revert pattern | Informational | Code Quality | Unresolved |
| 15 | Inconsistent import pattern | Informational | Code Quality | Unresolved |

## 2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

### 1. The parameters defined in verifySig() are inconsistent with the ones used

| | |
|---|---|
| Severity: Medium | Category: Business Logic |

Target:

- contracts/BOAT/BiMech2ACE.sol
- contracts/BOAT/QuartanPrimes2ACE.sol

### Description

contracts/BOAT/BiMech2ACE.sol:L111

```
verifySig(sender, serverTimeStamp, tokenID, signature) == false
```

contracts/BOAT/BiMech2ACE.sol:L286-L298

```
function verifySig(
    address sender,
    uint256 APTokenID,
    uint256 timeStamp,
    bytes calldata signature
) internal view returns (bool) {
    bytes32 messageHash = keccak256(
        abi.encode(sender, APTokenID, timeStamp, 77485632)
    );
    return
        _signerAddress ==
        messageHash.toEthSignedMessageHash().recover(signature);
}
```

contracts/BOAT/QuartanPrimes2ACE.sol:L109

```
verifySig(sender, serverTimeStamp, tokenID, signature) == false
```

contracts/BOAT/QuartanPrimes2ACE.sol:L279-L291

```
function verifySig(
    address sender,
    uint256 APTokenID,
    uint256 timeStamp,
    bytes calldata signature
) internal view returns (bool) {
    bytes32 messageHash = keccak256(
        abi.encode(sender, APTokenID, timeStamp, 53238545)
    );
    return
        _signerAddress ==
        messageHash.toEthSignedMessageHash().recover(signature);
}
```

Both **BiMech2ACE** and **QuartanPrimes2ACE** declare a signature verification function with the format `verifySig(address sender,uint256 APTokenID,uint256 timeStamp,bytes calldata signature)`. However, in their invocations, the parameter passing forms are `verifySig(sender, serverTimeStamp, tokenID, signature)`, where the formal parameters **APTokenID** and **timeStamp** do not match the passed arguments **serverTimeStamp** and **tokenID**.

## Recommendation

Consider adjusting the order of the parameters passed to the **verifySig**() function to match the formal parameters.

## 2. A user making a new request when the last request has not been responded to will cause any response to the user’s request to fail

| | |
|---|---|
| Severity: Medium | Category: Business Logic |

Target:

- contracts/BOAT/APPoint2ACE.sol

## Description

contracts/BOAT/APPoint2ACE.sol:L72-L87

```
function request(uint256 nonce, bytes calldata signature)
    external
    payable
    whenNotPaused
{
    address sender = msg.sender;
    if (verifySig(sender, nonce, signature) == false) {
        revert("ShouldCorrectSignature");
    }
    uint256 oldNonce = userRequestNonceHistory[sender];
    if (nonce != (oldNonce + 1)) {
        revert("Invalid nonce");
    }
    userRequestNonceHistory[sender] = nonce;
    emit ERequestSuc(sender, nonce);
}
```

contracts/BOAT/APPoint2ACE.sol:L99-L134

```
function response(
    address payable user,
    uint256 nonce,
    uint256 aceAmount
) external onlyRole(CTO_ROLE) {
    uint256 userRequestNonce = userRequestNonceHistory[user];
    if (userRequestNonce != nonce) {
        revert("ShouldMatchNonce");
    }
    uint256 oldResponseNonce = serverResponseNonceHistory[user];
    if ((oldResponseNonce + 1) != nonce) {
        revert("ShouldMatchNonce2");
    }
    serverResponseNonceHistory[user] = nonce;
    uint256 currentTimeStamp = block.timestamp;
    uint256 oldTimeStamp = _timeStampForLastSuccessClaim_Wallet[user];
    if (currentTimeStamp <= oldTimeStamp) {
        revert ShouldFutureTimeStamp();
    }
    uint256 elapsedTime = currentTimeStamp - oldTimeStamp;
    if (elapsedTime <= claimCD) {
        revert ShouldInCD();
    }
    _timeStampForLastSuccessClaim_Wallet[user] = currentTimeStamp;
    aceAmount = (aceAmount * calcBOATBUFF(user)) / 10;
    if (aceAmount >= address(this).balance) {
        revert ShouldSufficientBalance();
    }
    (bool result, ) = user.call{value: aceAmount}("");
    if (result == false) {
        revert("ShouldbeAbletoReceive");
    }
    emit EResponSuc(user, nonce, aceAmount);
}
```

The APPoint2ACE contract implements a process where a user sends a request by using **request**() and the CTO role returns a response by using **response()**. The **response()** function performs sanity checks on the **userRequestNonceHistory** and **serverResponseNonceHistory** state variables to maintain the order of requests and responses.

However, there is a risky scenario where if the CTO fails to respond to a user’s request, and does not notify the user in a timely manner, the user may make a new request by calling the **request**() function again. In this case, it will be impossible to respond to the user's request using the **response()** function because the check on the **userRequestNonceHistory** and **serverResponseNonceHistory** state variables cannot be satisfied simultaneously.

Take the following scenario as an example:

1. The user sends a request() with nonce 1 and the correct signature. Then, userRequestNonceHistory[user] == 1;
2. The response() for nonce 1 fails and the user is not aware of this. Thus, serverResponseNonceHistory[user] == 0;
3. The user sends another request() with nonce 2 and the correct signature. Thus, userRequestNonceHistory[user] == 2;
4. In this case, any response() to the user’s request will fail:
    a. response() to nonce 1 will fail because the userRequestNonceHistory[user] is 2 now, and the userRequestNonce == nonce check will fail.
    b. response() to nonce 2 will fail because the serverResponseNonceHistory[user] is 0 now, and the (serverResponseNonceHistory[user] + 1) == nonce check will fail.

## Recommendation

Consider adding a sanity check for **serverResponseNonceHistory** in the **request()** function to ensure that the **request**() function fails when the previous request has not been responded to.

| 3. Centralization risk | |
|---|---|
| Severity: Medium | Category: Centralization |
| Target:<br>- all | |

## Description

The upgradeable proxy pattern is used in Fusionist. The proxy admin controls the upgrade mechanism to upgradeable proxies, they can change the respective implementations. Should the admin’s private key be compromised, an attacker could upgrade the logic contract to execute their own malicious logic on the proxy state.

In addition, there are multiple role based access control models in the project, but all the addresses corresponding to the privileged roles are the deployer of the contract. When the private key is leaked and obtained by a malicious user, they can call critical functions such as **setClaimCD** and **setDailyACE** to disrupt the normal operation of the contract.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the users.

## Recommendation

Consider transferring the privileged roles to multi-sig accounts.

## 4. Inaccurate boundary conditions in sanity checks

| | |
|---|---|
| Severity: Low | Category: Business Logic |

Target:

- contracts/BOAT/APPoint2ACE.sol
- contracts/BOAT/WaterPump.sol
- contracts/BOAT/BOATClaimHub.sol
- contracts/BOAT/BiMech2ACE.sol
- contracts/BOAT/APDailyACE.sol
- contracts/BOAT/APHistory2ACE.sol
- contracts/BOAT/QuartanPrimes2ACE.sol

## Description

contracts/BOAT/APPoint2ACE.sol:L126-L128

```
if (aceAmount >= address(this).balance) {
    revert ShouldSufficientBalance();
}
```

In the if statement, the comparison between **aceAmount** and **address(this).balance** is used to ensure the function works properly. If **aceAmount** is greater than or equal to **address(this).balance**, revert will be triggered. However, in real-world scenarios, if **aceAmount** is equal to **address(this).balance**, revert should not occur and the function should execute normally. The **>=** check should be **>**.

The same issue also applies to the following codes:

contracts/BOAT/WaterPump.sol:L128-L130

```
if (randomWater >= address(this).balance) {
    revert ShouldSufficientBalance();
}
```

contracts/BOAT/BOATClaimHub.sol:L147-L149

```
if (sendAmount >= address(this).balance) {
    revert ShouldSufficientBalance();
}
```

contracts/BOAT/BiMech2ACE.sol:L137-L139

```
if (sendAmount >= address(this).balance) {
    revert ShouldSufficientBalance();
}
```

contracts/BOAT/APDailyACE.sol:L74-L76

```
if (sendAmount >= address(this).balance) {
    revert ShouldSufficientBalance();
}
```

contracts/BOAT/APHistory2ACE.sol:L49-L51

```
if (aceAmount >= address(this).balance) {
    revert ShouldSufficientBalance();
}
```

contracts/BOAT/QuartanPrimes2ACE.sol:L133-L135

```
if (sendAmount >= address(this).balance) {
    revert ShouldSufficientBalance();
}
```

## Recommendation

Consider changing **>=** to **>** in the mentioned comparison statement.

## 5. Incorrect value for UPGRADER_ROLE role

| | |
|---|---|
| Severity: Low | Category: Configuration |
| Target:<br>- contracts/BOAT/WaterPump.sol | |

## Description

contracts/BOAT/WaterPump.sol:L41-L42

```
bytes32 public constant COO_ROLE = keccak256("COO_ROLE");
bytes32 public constant UPGRADER_ROLE = keccak256("COO_ROLE");
```

The **UPGRADER_ROLE** above should use “UPGRADER_ROLE” as input for keccak256. However, it uses “COO_ROLE” as input, and this results in the same value of the keccak256 hash in both COO_ROLE and UPGRADE_ROLE. As a result, the UPGRADER_ROLE can call functions modified by onlyRole(COO_ROLE) while the COO_ROLE can call functions modified by onlyRole(UPGRADER_ROLE).

## Recommendation

Consider setting the UPGRADER_ROLE variable to an appropriate value.

### 6. Missing events for governor only functions that change critical parameters

| | |
|---|---|
| Severity: Low | Category: Auditing and Logging |
| Target:<br>- contracts/BOAT/APDailyACE.sol<br>- contracts/BOAT/APHistory2ACE.sol<br>- contracts/BOAT/APPoint2ACE.sol<br>- contracts/BOAT/BiMech2ACE.sol<br>- contracts/BOAT/BOATClaimHub.sol<br>- contracts/BOAT/BOATCollectionCommonClaimConfigV1.sol<br>- contracts/BOAT/QuartanPrimes2ACE.sol<br>- contracts/BOAT/WaterPump.sol | |

## Description

The governor only functions that change critical parameters should emit events. Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes with timelocks that allow users to evaluate them and consider if they would like to engage/exit based on how they perceive the changes as affecting the trustworthiness of the protocol. The alternative of directly querying on-chain contract state for such changes is not considered practical for most users/usages.

Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

Throughout the Fusionist codebase, events are lacking in the privileged setter functions (e.g. setSignerAddress(), setClaimCD()) and withdrawal functions (e.g. withdraw()).

## Recommendation

Consider adding events to all privileged functions that change critical parameters.

## 7. Inconsistency between signature verification and documentation

| | |
|---|---|
| Severity: Low | Category: Configuration |

Target:
- contracts/BOAT/APDailyACE.sol
- contracts/BOAT/APPoint2ACE.sol
- contracts/BOAT/BiMech2ACE.sol
- contracts/BOAT/BOATClaimHub.sol
- contracts/BOAT/QuartanPrimes2ACE.sol
- contracts/BOAT/WaterPump.sol

## Description

contracts/BOAT/APDailyACE.sol:L224-L232

```
function verifySig(
    address sender,
    uint256 APTokenID,
    uint256 timeStamp,
    bytes calldata signature
) internal view returns (bool) {
    bytes32 messageHash = keccak256(
        abi.encode(sender, APTokenID, timeStamp, 66468061)
    );
```

contracts/BOAT/APPoint2ACE.sol:L172-L177

```
function verifySig(
    address sender,
    uint256 nonce,
    bytes calldata signature
) internal view returns (bool) {
    bytes32 messageHash = keccak256(abi.encode(sender, nonce, 96911437));));
```

contracts/BOAT/BiMech2ACE.sol:L286-294

```
function verifySig(
    address sender,
    uint256 APTokenID,
    uint256 timeStamp,
    bytes calldata signature
) internal view returns (bool) {
    bytes32 messageHash = keccak256(
        abi.encode(sender, APTokenID, timeStamp, 77485632)
    );
```

contracts/BOAT/BOATClaimHub.sol:L254-L259

```
function verifySig1(
    address sender,
    uint256 tokenID,
    bytes calldata signature
) internal view returns (bool) {
    bytes32 messageHash = keccak256(abi.encode(sender, tokenID, 44584911));
```

contracts/BOAT/QuartanPrimes2ACE.sol:L279-287

```
function verifySig(
    address sender,
    uint256 APTokenID,
    uint256 timeStamp,
    bytes calldata signature
) internal view returns (bool) {
    bytes32 messageHash = keccak256(
        abi.encode(sender, APTokenID, timeStamp, 53238545)
    );
```

contracts/BOAT/WaterPump.sol:L239-L246

```
function verifySig(
    address sender,
    uint256 timeStamp,
    bytes calldata signature
) internal view returns (bool) {
    bytes32 messageHash = keccak256(
        abi.encode(sender, timeStamp, 36168085)
    );
```

The highlighted parameters in the above verifySig() functions differ from those described in the documentation that was provided by the project. Moreover, the highlighted parameters are magic numbers, which both obscure the purpose of the parameters and unnecessarily lead to potential error if the numbers are changed during development.

## Recommendation

Consider using constant variables with descriptive names to replace the highlighted magic numbers and documenting the purpose of these parameters in the comments.

## 8. Mismatch between comment and implementation

| Severity: Low | Category: Business Logic |
|---|---|
| Target:<br>- contracts/BOAT/WaterPump.sol | |

## Description

contracts/BOAT/WaterPump.sol:L114

```
//finalACE = ((0.01*N+0.008) + snowflakeLevelBonus)*(1/1.2);
```

The highlighted comment shows the corresponding calculation is multiple by 1 or 1.2.

contracts/BOAT/WaterPump.sol:L126-L127

```
uint256 multiple10 = calcBOATBUFF(sender, signature);
randomWater = (randomWater * multiple10) / 10;
```

However, the variable **multiple10** can only be 10 or 15. So the highlighted part in the comment should be (1 or 1.5) instead of (1/1.2).

Another mismatch occurs in the following code:
contracts/BOAT/WaterPump.sol:L205-L219

```
///@dev inculding BUFF. If you have token4, the return value will +4 hours.
function timeSinceLastSuccClaim2(address user)
    public
    view
    returns (uint256)
{
    uint256 currentTimeStamp = block.timestamp;
    if (
        hasToken(user, 5) == true &&
        currentTimeStamp < getBuffEndTimeStamp(5, user)
    ) {
        currentTimeStamp = currentTimeStamp + 4 hours;
    }
    return currentTimeStamp - _timeStampForLastSuccessClaim[user];
}
```

The highlighted comment indicates token4 is used in the timeSinceLastSuccClaim2() function, while tokenId 5 instead of token4 is actually checked and used in the function.

## Recommendation

Consider fixing the mismatch between comments and implementations.

## 9. Unnecessary grantRole operation in setSignerAddress()

| Severity: Low | Category: Business Logic |
|---|---|

Target:
- contracts/BOAT/APPoint2ACE.sol

## Description

contracts/BOAT/APPoint2ACE.sol:L144-L147

```
function setSignerAddress(address newValue) public onlyRole(CTO_ROLE) {
    signerAddress = newValue;
    grantRole(CTO_ROLE, signerAddress);
}
```

As the function name implies, the **setSignerAddress()** function should only be used to set the signer address. However, it also calls the **grantRole()** function, which results in bad functionality coupling. Instead of embedding grantRole() in the setSignerAddress() function, it would be preferable to call setSignerAddress() and grantRole() separately.
In addition, the *grantRole(CTO_ROLE, signerAddress)*; line requires the caller to have the **getRoleAdmin(CTO_ROLE)** role, while the setSignerAddress() function only explicitly requires the caller to have the **CTO_ROLE** role.

## Recommendation

Consider decoupling the grantRole() logic from the setSignerAddress() function.

# 2.3 Informational Findings

### 10. Could add an amount parameter to the EClaimSuc() events

| | |
|---|---|
| Severity: Informational | Category: Auditing and Logging |
| Target:<br>- contracts/BOAT/APDailyACE.sol<br>- contracts/BOAT/APHistory2ACE.sol<br>- contracts/BOAT/BiMech2ACE.sol<br>- contracts/BOAT/QuartanPrimes2ACE.sol | |

## Description

contracts/BOAT/APDailyACE.sol:L128

```
emit EClaimSuc(sender);
```

contracts/BOAT/APHistory2ACE.sol:L58

```
emit EClaimSuc(sender);
```

contracts/BOAT/BiMech2ACE.sol:L146

```
emit EClaimSuc(sender, zeroTo99, buffFactor);
```

contracts/BOAT/QuartanPrimes2ACE.sol:L142

```
emit EClaimSuc(sender, buffFactor);
```

The EClaimSuc() event is emitted when a user successfully claims native tokens from the contract. However, the amount claimed is not logged in the EClaimSuc() event.

## Recommendation

Consider adding the **amount** parameter to the **EClaimSuc()** event for easier off-chain monitoring.

## 11. Spelling mistakes

| | |
|---|---|
| Severity: Informational | Category: Code Quality |
| Target:<br>- contracts/BOAT/BOATClaimHub.sol | |

## Description

contracts/BOAT/BOATClaimHub.sol:L25

```
error ShouldCorrentSig();
```

contracts/BOAT/BOATClaimHub.sol:L135

```
revert ShouldCorrentSig();
```

contracts/BOAT/BOATClaimHub.sol:L26

```
error ShouldOnlyOnwOne();
```

contracts/BOAT/BOATClaimHub.sol:L117

```
revert ShouldOnlyOnwOne();
```

These are spelling mistakes in Should**Corrent**Sig and ShouldOnly**Onw**One.

## Recommendation

Consider changing Should**Corrent**Sig to Should**Correct**Sig and ShouldOnly**Onw**One to ShouldOnly**Own**One.

## 12. Redundant code and comments

| Severity: Informational | Category: Redundancy |
|---|---|

Target:

- contracts/BOAT/APPoint2ACE.sol
- contracts/BOAT/APDailyACE.sol
- contracts/BOAT/APHistory2ACE.sol
- contracts/BOAT/BoilerplateERC1155_BOAT.sol
- contracts/BOAT/BOATClaimHub.sol
- contracts/BOAT/BOATCollectionCommonClaimConfigV1.sol
- contracts/BOAT/BoilerplateERC1155_BOAT.sol
- contracts/BOAT/BiMech2ACE.sol
- contracts/BOAT/QuartanPrimes2ACE.sol

## Description

**1. Unnecessary payable modifier**

The follow functions have unnecessary **payable** modifiers:

- claimBatch(), withdraw() in the APDailyACE contract
- claim(), withdraw() in the APHistory2ACE contract
- request(), withdraw() in the APPoint2ACE contract
- withdraw() in the BOATClaimHub contract
- claimBatch(), withdraw() in the BiMech2ACE contract
- claimBatch(), withdraw() in the QuartanPrimes2ACE contract
- claim(), withdraw() in the WaterPump contract

**2. Unused import**

contracts/BOAT/BoilerplateERC1155_BOAT.sol:L8

```
import {Strings} from "@openzeppelin/contracts/utils/Strings.sol";
```

contracts/BOAT/QuartanPrimes2ACE.sol:L10

```
import {IBOATCollectionCommonClaimConfigV1, BOATCollectionCommonClaimConfigV1} from
"./BOATCollectionCommonClaimConfigV1.sol";
```

contracts/BOAT/BiMech2ACE.sol:L10

```
import {IBOATCollectionCommonClaimConfigV1, BOATCollectionCommonClaimConfigV1} from
"./BOATCollectionCommonClaimConfigV1.sol";
```

contracts/BOAT/APPoint2ACE.sol:L10

```
import {IBOATCollectionCommonClaimConfigV1, BOATCollectionCommonClaimConfigV1} from
"./BOATCollectionCommonClaimConfigV1.sol";
```

contracts/BOAT/WaterPump.sol:L10

```
import {IBOATCollectionCommonClaimConfigV1, BOATCollectionCommonClaimConfigV1} from
"./BOATCollectionCommonClaimConfigV1.sol";
```

The **Strings** and **IBOATCollectionCommonClaimConfigV1** are imported but not used in the above contracts.

**3. Comment-out code**

contracts/BOAT/APDailyACE.sol:L131-L170

```
// function claim(
//     uint256 APTokenID,
...
//     emit EClaimSuc(sender);
// }
```

contracts/BOAT/BiMech2ACE.sol:L149-L196

```
// function claim(
//     uint256 serverTimeStamp,
...
//     emit EClaimSuc(sender, zeroTo99, buffFactor);
// }
```

contracts/BOAT/QuartanPrimes2ACE.sol:L145-L189

```
// function claim(
//     uint256 APTokenID,
...
//     emit EClaimSuc(sender);
// }
```

The above comment-out code should be removed before the contracts are deployed to the mainnet.

## Recommendation

Consider removing unnecessary code.

## 13. Inconsistent role design

| | |
|---|---|
| Severity: Informational | Category: Code Quality |
| Target:<br>- all | |

## Description

The Fusionist project uses a role based access control system. However the responsibilities for the roles are not consistent throughout the codebase.

contracts/BOAT/APHistory2ACE.sol:L70-L72

```
function setSignerAddress(address newValue) public onlyRole(PAUSER_ROLE) {
    signerAddress = newValue;
}
```

contracts/BOAT/APDailyACE.sol:L189-L1942

```
function setSignerAddress(address newValue)
    public
    onlyRole(DEFAULT_ADMIN_ROLE)
{
    _signerAddress = newValue;
}
```

Take setSignerAddress() as an example. The setSignerAddress() function in the APHistory2ACE contract requires the PAUSE_ROLE to operate, while the one in the APDailyACE contract requires the DEFAULT_ADMIN_ROLE to operate.

## Recommendation

Consider using a consistent design for the roles throughout the project.

## 14. Inconsistent revert pattern

| Severity: Informational | Category: Code Quality |
|---|---|

Target:
- contracts/BOAT/APPoint2ACE.sol
- contracts/BOAT/BOATCollectionCommonClaimConfigV1.sol

## Description

Custom errors are used throughout the Fusionist codebase, however, the following revert statements throw error messages instead of custom errors.

contracts/BOAT/APPoint2ACE.sol:L78-L84

```
if (verifySig(sender, nonce, signature) == false) {
   revert("ShouldCorrectSignature");
}
uint256 oldNonce = userRequestNonceHistory[sender];
if (nonce != (oldNonce + 1)) {
   revert("Invalid nonce");
}
```

contracts/BOAT/APPoint2ACE.sol:L105-L112

```
if (userRequestNonce != nonce) {
   revert("ShouldMatchNonce");
}

uint256 oldResponseNonce = serverResponseNonceHistory[user];
if ((oldResponseNonce + 1) != nonce) {
   revert("ShouldMatchNonce2");
}
```

contracts/BOAT/APPoint2ACE.sol:L130-L132

```
if (result == false) {
   revert("ShouldReceiveACE");
}
```

contracts/BOAT/APPoint2ACE.sol:L166-L168

```
if (result == false) {
   revert("ShouldReceiveACE");
}
```

contracts/BOAT/BOATCollectionCommonClaimConfigV1.sol:L82-L84

```
if (address(boatClaimHub) == address(0)) {
   revert("Address 0");
}
```

## Recommendation

Consider using custom errors instead of error messages to save gas.

## 15. Inconsistent import pattern

| Severity: Informational | Category: Code Quality |
|---|---|
| Target:<br>- contracts/BOAT/BOATClaimHub.sol<br>- contracts/BOAT/BOATCollectionCommonClaimConfigV1.sol | |

## Description

Explicit imports (i.e. import {...} from “...”) are used throughout the Fusionist codebase, however the following imports are non-explicit imports (i.e. import “...”).

contracts/BOAT/BOATClaimHub.sol:L4-L9

```
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
import {BOATCollectionCommonClaimConfigV1, IBOATCollectionCommonClaimConfigV1} from
"./BOATCollectionCommonClaimConfigV1.sol";
import {BOATCollection} from "./BOATCollection.sol";
import {ECDSAUpgradeable} from
"@openzeppelin/contracts-upgradeable/utils/cryptography/ECDSAUpgradeable.sol";
```

contracts/BOAT/BOATCollectionCommonClaimConfigV1.sol:L4-L7

```
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
import {BOATClaimHub} from "./BOATClaimHub.sol";
```

## Recommendation

Consider using explicit imports instead of non-explicit imports.

# Appendix

## Appendix 1 - Files in Scope

This audit covered the following file in commit db4eefa:

| File | SHA-1 hash |
|---|---|
| contracts/BOAT/APDailyACE.sol | 5e2f3983820796c02744c38da015e131393163ac |
| contracts/BOAT/APHistory2ACE.sol | e6f28109a86572593ad992847f443935764bb417 |
| contracts/BOAT/APPoint2ACE.sol | 7083abdf5a774728babc01042dc851b8987968e4 |
| contracts/BOAT/BiMech2ACE.sol | 67b386fc84f5d35b6d49457c01db584c12883cc0 |
| contracts/BOAT/BOATClaimHub.sol | c73a0aba276adfb3ef72b06423a8004822b8aba9 |
| contracts/BOAT/BOATCollection.sol | 5da76e60c1d7aa4606c09e25d0d9b73cdf8afe23 |
| contracts/BOAT/BOATCollectionCommonClaimConfigV1.sol | 2d74fcb23aa58d44c332e6199c7f7e9adee8f17d |
| contracts/BOAT/QuartanPrimes2ACE.sol | c2c5ea15cb2db77618003280bf3912df6406cffa |
| contracts/BOAT/WaterPump.sol | 6be4e44d22fa294a5868c0c91104dae766b8d70b |
| contracts/common/OperatorFilterer.sol | 9298fc8ce30765295f96086d0f8e66a37ae78c81 |
| contracts/BoilerplateERC1155_BOAT.sol | 387e18883c33b0f9c20b6d3140ae146e98b5964e |