ExVul

SMART CONTRACT AUDIT REPORT

Aifortuna Smart Contract

SEPTEMBER 2025

Contents

1. EXECUTIVE SUMMARY

ExVul Web3 Security was engaged by **Aifortuna** to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- **Likelihood**: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- **Impact**: measures the technical loss and business damage of a successful attack.
- **Severity**: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly: Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood \ IMPACT	Informational	Low	Medium	High
High	INFO	MEDIUM	HIGH	CRITICAL
Medium	INFO	LOW	MEDIUM	HIGH
Low	INFO	LOW	LOW	MEDIUM

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- **Basic Coding Bugs**: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- **Code and business security testing**: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- **Additional Recommendations**: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	• Apply Verification Control • Authorization Access Control • Forged Transfer Vulnerability • Forged Transfer Notification • Numeric Overflow • Transaction Rollback Attack • Transaction Block Stuffing Attack • Soft Fail Attack • Hard Fail Attack • Abnormal Memo • Abnormal Resource Consumption • Secure Random Number

Advanced Source Code Scrutiny	• Asset Security • Cryptography Security • Business Logic Review • Source Code Functional Verification • Account Authorization Control • Sensitive Information Disclosure • Circuit Breaker • Blacklist Control • System API Call Analysis • Contract Deployment Consistency Check • Abnormal Resource Consumption
Additional Recommendations	• Semantic Consistency Checks • Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

2.1 Project Info And Contract Address

Project Name	Audit Time	Language
Aifortuna	20/09/2025 - 25/09/2025	Solidity

Repository
https://github.com/aifortuna/aifortuna-contract

Commit Hash
d5db4a3013699dce31b9660df6b7b4a2aedc81f1

2.2 Summary

Severity	Found
CRITICAL	1
HIGH	3
MEDIUM	7
LOW	12
INFO	8

2.3 Key Findings

Severity	Findings Title	Status
CRITICAL	buyToken User Pays But Receives No Output Tokens	Fixed
HIGH	Claim logic errors lead to fund loss	Fixed
HIGH	claimNodeCardRewards Variable Usage Error Leading to Incorrect Exchange	Fixed
HIGH	claimNodeCardRewards Lacks Actual Reward Distribution Mechanism	Acknowledge
MEDIUM	Buy transaction whitelist check bypassed by fee exempt users	Fixed
MEDIUM	Game contract fee allocation causes permanent token loss to zero address	Fixed
MEDIUM	Old team address retains privileges after updateTeam	Fixed
MEDIUM	PancakeSwapInfo and TokenInfo status are inconsistent	Fixed
MEDIUM	forceSetWithdrawCount Allows Setting Lower Values Leading to Data Overwrite	Fixed
MEDIUM	mintPairedToken Uses Hardcoded AGT Instead of Configured PairedToken	Acknowledge
MEDIUM	claimNodeCardRewards Lacks NodeCard Purchase Validation	Acknowledge
LOW	Constructor lacks fee wallet auto exemption logic	Fixed
LOW	setFeeWallet fails to revoke old fee wallet exemption status	Fixed
LOW	Comment and code mismatch for agt_pair_feeBps rate	Fixed
LOW	Event parameter mismatch in NodeCardFeeReceived	Fixed
LOW	AGT pair whitelist and fee bypass	Fixed

Severity	Findings Title	Status
LOW	Incorrect judgment causes the logic to never trigger	Fixed
LOW	Stale Balance Check Potential Transaction Failures	Fixed
LOW	Comment Implementation Mismatch in Alpha Calculation	Fixed
LOW	Operator Array Management Without pop Leading to Duplicates	Fixed
LOW	Inefficient WithdrawCount Initialization Wastes Storage Slot	Fixed
LOW	withdrawConfirm External Transfer Before State Update	Fixed
LOW	Missing Balance Validation in claimNodeCardRewards	Acknowledge
INFO	Setter methods lack duplicate value validation	Fixed
INFO	Constructor lacks FeeExemptUpdated event emission	Fixed
INFO	Unused AccessControl import increases deployment cost	Fixed
INFO	Unnecessary recoverETH function	Fixed
INFO	Misspelling of words	Fixed
INFO	Overwrite existing data	Fixed
INFO	Missing zero address check	Fixed
INFO	Missing minDeposits Parameter in FortunaInitialized Event	Fixed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 buyToken User Pays But Receives No Output Tokens

SEVERITY: CRITICAL **STATUS:** Fixed

PATH:

src/Fortuna.sol

DESCRIPTION:

buyToken allows users to pay USDT but fails to transfer the swapped output tokens to users, only triggering hedge logic without any user benefit.

```
function buyToken(
    address token,
    uint256 amount,
    uint256 deadline,
    string memory signContext,
    bytes memory signature
) external payable nonReentrant fundBNB {
    if (!pancakeSwapInfos[token].isSupported) revert TokenNotSupported();
    if (token != usdt) revert TokenNotSupported();
    if (amount == 0) revert AmountZero();
    if (pancakeSwapInfos[token].buyFeePercent == 0) revert BuyFeeNotSet();
    if (deadline < block.timestamp) revert SignatureExpired();
    if (usedSignContexts[signContext]) revert SignContextUsed();

    bytes32 message = keccak256(abi.encodePacked(block.chainid,
   msg.sender, token, amount, signContext, deadline));
    _verifySignature(message, signature);
    usedSignContexts[signContext] = true;

    IERC20(token).safeTransferFrom(msg.sender, address(this), amount);

    address pairedToken = pancakeSwapInfos[token].pairedToken;
    if (pairedToken == address(0)) revert PairedTokenNotSet();

    address[] memory path = new address[](2);
```

```
    path[0] = token;
    path[1] = pairedToken;

    // handle USDT 10% fee and transfer to fee wallet
    uint256 buy_fee = (amount * pancakeSwapInfos[token].buyFeePercent) /
    BPS;
    uint256 realAmount = amount - buy_fee;
    IERC20(token).safeTransfer(feeWallet, buy_fee);

    _approveExact(token, realAmount);
    uint256 amountOut = _swapSupportingFeeReturnOut(realAmount, path,
    deadline);

    emit TokenBought(msg.sender, token, amount, pairedToken, amountOut,
    signContext, signature);
    treasuryHedge.execute(amountOut, true);
}
```

IMPACT:

Users pay USDT but receive no output tokens, constituting a pay without receiving economic flaw.

RECOMMENDATIONS:

Transfer the swapped output tokens to the user.

```
_approveExact(token, realAmount);
uint256 amountOut = _swapSupportingFeeReturnOut(realAmount, path,
    deadline);

+ IERC20(pairedToken).safeTransfer(msg.sender, amountOut);

emit TokenBought(msg.sender, token, amount, pairedToken, amountOut,
    signContext, signature);
treasuryHedge.execute(amountOut, true);
```

3.2 Claim logic errors lead to fund loss

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`src/Fortuna.sol`

DESCRIPTION:

In the Fortuna.sol file, the `claimPairedTokenRewards` function incorrectly uses the required fee as the claim amount when calculating `amountPairedOut`.

```
function claimPairedTokenRewards(
    address token,
    uint256 amount,
    uint256 deadline,
    string memory signContext,
    bytes memory signature
) external payable nonReentrant fundBNB returns (uint256[] memory
   amounts) {
    ...
    uint256 sell_amount = (amount * claimFeePercent) / BPS;
    _approveExact(token, amount);
    uint256 amountPairedOut = _swapSupportingFeeReturnOut(sell_amount,
   path, deadline);

    emit PairedTokenRewardsClaimed(
        msg.sender,
        token,
        amount - sell_amount,
        address(agt),
        amountPairedOut,
        claimFeePercent,
        signContext,
        signature
    );

    ...
}
```

IMPACT:

Due to calculation errors, the actual claim reward is far less than the theoretical reward, resulting in economic losses.

RECOMMENDATIONS:

Modify the claim logic. Same problem in `claimNodeCardRewards`.

```
function claimPairedTokenRewards(
    address token,
    uint256 amount,
    uint256 deadline,
    string memory signContext,
    bytes memory signature
) external payable nonReentrant fundBNB returns (uint256[] memory
   amounts) {
    ...
-   uint256 sell_amount = (amount * claimFeePercent) / BPS;
+   uint256 claim_fee = (amount * claimFeePercent) / BPS;
+   uint256 sell_amount = amount - claim_fee;
    _approveExact(token, amount);
    uint256 amountPairedOut = _swapSupportingFeeReturnOut(sell_amount,
   path, deadline);

    emit PairedTokenRewardsClaimed(
        msg.sender,
        token,
-       amount - sell_amount,
+       sell_amount
        address(agt),
        amountPairedOut,
        claimFeePercent,
        signContext,
        signature
    );

    ...
}
```

3.3 claimNodeCardRewards Variable Usage Error Leading to Incorrect Exchange

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`src/Fortuna.sol`

DESCRIPTION:

`claimNodeCardRewards` when processing the tokenB branch, the code incorrectly uses `amount` (tokenA quantity) instead of `amountB` (tokenB quantity) to calculate `claim_fee` and `sell_amount`.

```
function claimNodeCardRewards(
    address tokenA,
    uint256 amount,
    address tokenB,
    uint256 amountB,
    uint256 deadline,
    string memory signContext,
    bytes memory signature
) external payable nonReentrant fundBNB {
    // ...

    if (amount > 0) {
        //A => B
        address[] memory path = new address[](2);
        path[0] = tokenA;
        path[1] = tokenB;

        _approveExact(tokenA, amount);

        uint256 claim_fee = (amount * claimFeePercent) / BPS;
        uint256 sell_amount = amount - claim_fee;
        uint256 amountPairedOut =
   _swapSupportingFeeReturnOut(sell_amount, path, deadline);

        emit PairedTokenARewardsClaimed(
```

```
            msg.sender, tokenA, sell_amount, tokenB, amountPairedOut,
    claimFeePercent, signContext, signature
        );
    }

    if (amountB > 0) {
        //B => A
        address[] memory path = new address[](2);
        path[0] = tokenB;
        path[1] = tokenA;
        _approveExact(tokenB, amountB);

        uint256 claim_fee = (amount * claimFeePercent) / BPS;
        uint256 sell_amount = amount - claim_fee;

        uint256 amountPairedOut =
    _swapSupportingFeeReturnOut(sell_amount, path, deadline);
        emit PairedTokenBRewardsClaimed(
            msg.sender, tokenB, sell_amount, tokenA, amountPairedOut,
    claimFeePercent, signContext, signature
        );
    }
}
```

```
if (amountB > 0) {
    //B => A
    address[] memory path = new address[](2);
    path[0] = tokenB;
    path[1] = tokenA;
    _approveExact(tokenB, amountB);

    uint256 claim_fee = (amount * claimFeePercent) / BPS;
    uint256 sell_amount = amount - claim_fee;

    uint256 amountPairedOut = _swapSupportingFeeReturnOut(sell_amount,
    path, deadline);
    emit PairedTokenBRewardsClaimed(
        msg.sender, tokenB, sell_amount, tokenA, amountPairedOut,
    claimFeePercent, signContext, signature
    );
}
```

IMPACT:

When `amount != amountB`, the tokenB branch will fail due to attempting to swap more tokens than approved.

RECOMMENDATIONS:

Fix the variable usage in the tokenB branch.

```
if (amountB > 0) {
    //B => A
    address[] memory path = new address[](2);
    path[0] = tokenB;
    path[1] = tokenA;
    _approveExact(tokenB, amountB);

-   uint256 claim_fee = (amount * claimFeePercent) / BPS;
-   uint256 sell_amount = amount - claim_fee;
+   uint256 claim_fee = (amountB * claimFeePercent) / BPS;
+   uint256 sell_amount = amountB - claim_fee;

    uint256 amountPairedOut = _swapSupportingFeeReturnOut(sell_amount,
   path, deadline);
    emit PairedTokenBRewardsClaimed(
        msg.sender, tokenB, sell_amount, tokenA, amountPairedOut,
   claimFeePercent, signContext, signature
    );
}
```

3.4 claimNodeCardRewards Lacks Actual Reward Distribution Mechanism

SEVERITY: HIGH **STATUS:** Acknowledge

PATH:

`src/Fortuna.sol`

DESCRIPTION:

`claimNodeCardRewards` despite its name suggesting it claims rewards, but only performs token inside swap without actually distributing any rewards to users.

```
function claimNodeCardRewards(
    address tokenA,
    uint256 amount,
    address tokenB,
    uint256 amountB,
    uint256 deadline,
    string memory signContext,
    bytes memory signature
) external payable nonReentrant fundBNB {
    require(deadline >= block.timestamp, "Signature expired");
    require(!usedSignContexts[signContext], "Sign context already used");

    bytes32 message = keccak256(
        abi.encodePacked(block.chainid, msg.sender, tokenA, amount,
   tokenB, amountB, signContext, deadline)
    );

    _verifySignature(message, signature);
    usedSignContexts[signContext] = true;

    if (amount > 0) {
        //A => B
        address[] memory path = new address[](2);
        path[0] = tokenA;
        path[1] = tokenB;

        _approveExact(tokenA, amount);
```

```
        uint256 claim_fee = (amount * claimFeePercent) / BPS;
        uint256 sell_amount = amount - claim_fee;
        uint256 amountPairedOut =
    _swapSupportingFeeReturnOut(sell_amount, path, deadline);

        emit PairedTokenARewardsClaimed(
            msg.sender, tokenA, sell_amount, tokenB, amountPairedOut,
    claimFeePercent, signContext, signature
        );
     }

     if (amountB > 0) {
        //B => A
        address[] memory path = new address[](2);
        path[0] = tokenB;
        path[1] = tokenA;
        _approveExact(tokenB, amountB);

        uint256 claim_fee = (amount * claimFeePercent) / BPS;
        uint256 sell_amount = amount - claim_fee;

        uint256 amountPairedOut =
    _swapSupportingFeeReturnOut(sell_amount, path, deadline);
        emit PairedTokenBRewardsClaimed(
            msg.sender, tokenB, sell_amount, tokenA, amountPairedOut,
    claimFeePercent, signContext, signature
        );
     }
}
```

IMPACT:

The function name is misleading. It suggests claiming rewards but only performs token swaps without distributing any actual rewards to users.

RECOMMENDATIONS:

Implement proper reward distribution mechanism, and also a similar problem with `claimPaired-TokenRewards`.

3.5 Buy transaction whitelist check bypassed by fee exempt users

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`src/AGT.sol`

DESCRIPTION:

`is_buy()` requires both `!feeExempt[from]` and `!feeExempt[to]` to return true, causing fee exempt users to bypass the whitelist check in buy transactions. When to address is fee exempt, `is_buy()` returns false, preventing the whitelist validation from executing. And the comment says charge fee but not.

```
} else if (is_buy(from, to)) {
    // Buying AGT from pair (FUSD -> AGT), check whitelist and charge fee
    require(whitelist[to], "not whitelisted");
}

function is_buy(address from, address to) public view returns (bool) {
    return swapPairs[from] && !feeExempt[from] && !feeExempt[to];
}

function addToWhitelist(address account, bool status) external onlyOwner {
    require(account != address(0), "Cannot whitelist zero address");
    whitelist[account] = status;
    emit WhitelistUpdated(account, status);
}
```

```
} else if (is_buy(from, to)) {
    // Buying AGT from pair (FUSD -> AGT), check whitelist and charge fee
    require(whitelist[to], "not whitelisted");
}

function is_buy(address from, address to) public view returns (bool) {
    return swapPairs[from] && !feeExempt[from] && !feeExempt[to];
}
```

IMPACT:

Fee exempt users can bypass whitelist restrictions during buy transactions.

RECOMMENDATIONS:

Separate whitelist check from fee exemption logic and make sure charge fee or not.

```
} else if (swapPairs[from]) {
    // Buying AGT from pair (FUSD -> AGT), check whitelist and charge fee
-   require(whitelist[to], "not whitelisted");
+   require(whitelist[to], "not whitelisted");
+   if (!feeExempt[to]) {
+       // charge fee
+   }
}

function is_buy(address from, address to) public view returns (bool) {
-   return swapPairs[from] && !feeExempt[from] && !feeExempt[to];
+   return swapPairs[from];
}
```

3.6 Game contract fee allocation causes permanent token loss to zero address

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`src/AGT.sol`

DESCRIPTION:

Constructor initializes `feeWallet` but leaves `gameContract` uninitialized (defaults to address(0)). When sell transactions occur, `_calculateFees()` allocates 90% of fees to `gameContract` without checking if it's zero address, causing tokens to be permanently lost when transferred to address(0).

```
constructor(uint256 initialSupply, address _feeWallet) ERC20("AIFortuna
    Game Token", "AGT") Ownable(msg.sender) {
    uint256 mintAmount = initialSupply * 10 ** decimals();
    _mint(msg.sender, mintAmount);
    feeExempt[msg.sender] = true;
    feeWallet = _feeWallet;
    gameFeeBps = 9000;
}

function _calculateFees(uint256 totalFee) internal view returns (uint256
    gameFee, uint256 teamFee) {
    gameFee = (totalFee * gameFeeBps) / BPS;  // 90% to address(0)
    teamFee = totalFee - gameFee;
}

// Later in transferFrom
super._transfer(from, gameContract, gameFee);  // Sends to address(0)
```

```
function _calculateFees(uint256 totalFee) internal view returns (uint256
    gameFee, uint256 teamFee) {
    gameFee = (totalFee * gameFeeBps) / BPS;  // 90% to address(0)
    teamFee = totalFee - gameFee;
}

// Later in transferFrom
```

```
super._transfer(from, gameContract, gameFee);  // Sends to address(0)
```

IMPACT:

90% of transaction fees are permanently lost to zero address before `setGameContract()` is called, resulting in significant token destruction and revenue loss.

RECOMMENDATIONS:

Add zero address check in fee calculation to prevent token loss.

```
function _calculateFees(uint256 totalFee) internal view returns (uint256
    gameFee, uint256 teamFee) {
-    gameFee = (totalFee * gameFeeBps) / BPS;
-    teamFee = totalFee - gameFee;
+    if (gameContract == address(0)) {
+        gameFee = 0;
+        teamFee = totalFee;
+    } else {
+        gameFee = (totalFee * gameFeeBps) / BPS;
+        teamFee = totalFee - gameFee;
+    }
}
```

3.7 Old team address retains privileges after updateTeam

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`src/FUSD.sol`

DESCRIPTION:

When `updateTeam()` is called, the old team address retains its whitelist and fee exempt privileges, while the new team address is granted the same privileges.

```
function updateTeam(address _team) external onlyOwner {
    require(_team != address(0), "Team cannot be zero address");
    address oldTeam = team;
    team = _team;

    // Update whitelist and fee exempt status
    whitelist[_team] = true;
    feeExempt[_team] = true;

    emit TeamUpdated(oldTeam, _team);
    emit WhitelistUpdated(_team, true);
    emit FeeExemptUpdated(_team, true);
}
```

IMPACT:

Old team address retains whitelist and fee exempt privileges.

RECOMMENDATIONS:

Remove privileges from old team address when updating. The same problem exists with `updateGameContract`.

```
function updateTeam(address _team) external onlyOwner {
    require(_team != address(0), "Team cannot be zero address");
```

```
    address oldTeam = team;
    team = _team;

    // Remove old team privileges
+   whitelist[oldTeam] = false;
+   feeExempt[oldTeam] = false;
+   emit WhitelistUpdated(oldTeam, false);
+   emit FeeExemptUpdated(oldTeam, false);

    // Update whitelist and fee exempt status
    whitelist[_team] = true;
    feeExempt[_team] = true;
}
```

3.8 PancakeSwapInfo and TokenInfo status are inconsistent

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`src/Fortuna.sol`

DESCRIPTION:

In Fortuna.sol, operations such as `AddPancakeSwapInfos`, `addToken`, and `removeToken` may cause inconsistent token status.

Issue 1: In `addPancakeSwapInfos`, there is no check whether the incoming Token is supported in `supportedTokens`.

Issue 2: In the `removeToken` operation, only the status of `supportedTokens` is changed, and the status of `pancakeSwapInfos` is not changed, which may lead to inconsistent Token status.

IMPACT:

Inconsistent token status may lead to errors in contract logic judgment, affecting transactions, rewards or fee calculations.

RECOMMENDATIONS:

Recommend 1: Verify that the Token is supported before creating PancakeSwapInfo.

```
function addPancakeSwapInfos(address token, address pairedToken) external
    onlyOwner {
+    require(supportedTokens[token].isSupported, ”Token not supported”);
     pancakeSwapInfos[token] =
         PancakeSwapInfo({isSupported: true, pairedToken: pairedToken,
    buyFeePercent: 0, buySupported: false});

     emit PancakeSwapInfoAdded(msg.sender, token, pairedToken,
    block.timestamp);
}
```

Recommend 2: Modify the status of PancakeSwapInfo during the `removeToken` operation.

```
function removeToken(address token) external onlyOwner {
    require(supportedTokens[token].isSupported, "Token not supported");

    supportedTokens[token].isSupported = false;
+   pancakeSwapInfos[token].isSupported = false;

    // Remove from tokenList
    for (uint256 i = 0; i < tokenList.length; i++) {
        if (tokenList[i] == token) {
            tokenList[i] = tokenList[tokenList.length - 1];
            tokenList.pop();
            break;
        }
    }

    emit TokenRemoved(msg.sender, token, block.timestamp);
}
```

3.9 forceSetWithdrawCount Allows Setting Lower Values Leading to Data Overwrite

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`src/Fortuna.sol`

DESCRIPTION:

`forceSetWithdrawCount` allows setting `_withdrawCount` to any value without validation, can lead to data overwritten in the `withdraws` mapping when the new value is lower than the current count.

```
function forceSetWithdrawCount(uint256 _withdrawCount) external onlyOwner
    {
     uint256 oldCount = withdrawCount;
     withdrawCount = _withdrawCount;
     emit WithdrawCountUpdated(msg.sender, oldCount, _withdrawCount,
    block.timestamp);
}

function withdrawRequest(address token, uint256 amount) external
    nonReentrant {
     // ...
     withdrawCount++;
     withdraws[withdrawCount] = Withdraw({
         user: msg.sender,
         token: token,
         amount: amount,
         timestamp: block.timestamp,
         isConfirmed: false,
         isCanceled: false
     });
     playerWithdrawRequest[msg.sender] = withdrawCount;
     // ...
}
```

IMPACT:

Setting `_withdrawCount` to a lower value will cause new withdraw requests to overwrite existing records in the `withdraws` mapping.

RECOMMENDATIONS:

Add validation to ensure `_withdrawCount` can only be set to a value greater than or equal to the current count.

```
function forceSetWithdrawCount(uint256 _withdrawCount) external onlyOwner
    {
+    require(_withdrawCount >= withdrawCount, "Cannot decrease withdraw
    count");
     uint256 oldCount = withdrawCount;
     withdrawCount = _withdrawCount;
     emit WithdrawCountUpdated(msg.sender, oldCount, _withdrawCount,
    block.timestamp);
}
```

3.10 mintPairedToken Uses Hardcoded AGT Instead of Configured PairedToken

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

src/Fortuna.sol

DESCRIPTION:

mintPairedToken hardcodes the output token to AGT instead of using the configured pairedToken from pancakeSwapInfos.

```
function mintPairedToken(
    address token,
    uint256 amount,
    uint256 deadline,
    string memory signContext,
    bytes memory signature
) external payable nonReentrant fundBNB returns (uint256) {
    if (!pancakeSwapInfos[token].isSupported) revert TokenNotSupported();

    if (amount == 0) revert AmountZero();
    require(amount <= getBalance(token), "Insufficient balance");
    if (deadline < block.timestamp) revert SignatureExpired();

    if (usedSignContexts[signContext]) revert SignContextUsed();

    bytes32 message = keccak256(abi.encodePacked(block.chainid,
   msg.sender, token, amount, signContext, deadline));
    _verifySignature(message, signature);
    usedSignContexts[signContext] = true;
    address[] memory path = new address[](2);
    path[0] = token;
    path[1] = address(agt);

    _approveExact(token, amount);
    uint256 amountOut = _swapSupportingFeeReturnOut(amount, path,
   deadline);
    emit PairedTokenMinted(msg.sender, token, amount, address(agt),
```

```
    amountOut, signContext, signature);
     return amountOut;
}
```

IMPACT:

The function name suggests minting paired tokens but hardcodes AGT instead of using the configured paired token.

RECOMMENDATIONS:

Use the configured `pairedToken` from `pancakeSwapInfos` instead of hardcoded AGT to match the function name.

```
function mintPairedToken(
    address token,
    uint256 amount,
    uint256 deadline,
    string memory signContext,
    bytes memory signature
) external payable nonReentrant fundBNB returns (uint256) {
    if (!pancakeSwapInfos[token].isSupported) revert TokenNotSupported();

    if (amount == 0) revert AmountZero();
    require(amount <= getBalance(token), "Insufficient balance");
    if (deadline < block.timestamp) revert SignatureExpired();

    if (usedSignContexts[signContext]) revert SignContextUsed();

    bytes32 message = keccak256(abi.encodePacked(block.chainid,
   msg.sender, token, amount, signContext, deadline));
    _verifySignature(message, signature);
    usedSignContexts[signContext] = true;

+   address pairedToken = pancakeSwapInfos[token].pairedToken;
+   if (pairedToken == address(0)) revert PairedTokenNotSet();

    address[] memory path = new address[](2);
    path[0] = token;
-   path[1] = address(agt);
+   path[1] = pairedToken;
```

```
    _approveExact(token, amount);
    uint256 amountOut = _swapSupportingFeeReturnOut(amount, path,
    deadline);
-   emit PairedTokenMinted(msg.sender, token, amount, address(agt),
    amountOut, signContext, signature);
+   emit PairedTokenMinted(msg.sender, token, amount, pairedToken,
    amountOut, signContext, signature);
    return amountOut;
}
```

3.11 claimNodeCardRewards Lacks NodeCard Purchase Validation

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

`src/Fortuna.sol`

DESCRIPTION:

`claimNodeCardRewards` only validates oracle signature but lacks verification that the tokens were actually purchased through NodeCard.

```
function claimNodeCardRewards(
    address tokenA,
    uint256 amount,
    address tokenB,
    uint256 amountB,
    uint256 deadline,
    string memory signContext,
    bytes memory signature
) external payable nonReentrant fundBNB {
    require(deadline >= block.timestamp, "Signature expired");
    require(!usedSignContexts[signContext], "Sign context already used");

    bytes32 message = keccak256(
        abi.encodePacked(block.chainid, msg.sender, tokenA, amount,
   tokenB, amountB, signContext, deadline)
    );

    _verifySignature(message, signature);
    usedSignContexts[signContext] = true;

    // ... rest of function
}
```

IMPACT:

Users could potentially claim rewards without actually purchasing through NodeCard if oracle signature

is compromised or incorrectly issued.

RECOMMENDATIONS:

Add NodeCard purchase validation by requiring purchase context verification.

```
function claimNodeCardRewards(
    address tokenA,
    uint256 amount,
    address tokenB,
    uint256 amountB,
    uint256 deadline,
    string memory signContext,
+   string memory purchaseContext,
    bytes memory signature
) external payable nonReentrant fundBNB {
    require(deadline >= block.timestamp, "Signature expired");
    require(!usedSignContexts[signContext], "Sign context already used");
+   require(!nodeCard.usedPurchaseContexts(purchaseContext), "Purchase
    context already used");

    bytes32 message = keccak256(
-       abi.encodePacked(block.chainid, msg.sender, tokenA, amount,
    tokenB, amountB, signContext, deadline)
+       abi.encodePacked(block.chainid, msg.sender, tokenA, amount,
    tokenB, amountB, signContext, purchaseContext, deadline)
    );

    _verifySignature(message, signature);
    usedSignContexts[signContext] = true;
+   nodeCard.markPurchaseContextUsed(purchaseContext);

    // ...
}
```

3.12 Constructor lacks fee wallet auto exemption logic

SEVERITY: LOW **STATUS:** Fixed

PATH:

`src/AGT.sol`

DESCRIPTION:

Constructor sets `feeWallet = _feeWallet` without automatically setting `feeExempt[_feeWallet] = true`, while the `setFeeWallet()` function correctly implements this auto-exemption logic.

```
constructor(uint256 initialSupply, address _feeWallet) ERC20("AIFortuna
    Game Token", "AGT") Ownable(msg.sender) {
     uint256 mintAmount = initialSupply * 10 ** decimals();
     _mint(msg.sender, mintAmount);
     feeExempt[msg.sender] = true;
     feeWallet = _feeWallet;

     gameFeeBps = 9000;

     emit AGTInitialized(msg.sender, mintAmount, block.timestamp);
     emit FeeWalletUpdated(address(0), _feeWallet);
}

function setFeeWallet(address wallet) external onlyOwner {
     require(wallet != address(0), "op zero");
     emit FeeWalletUpdated(feeWallet, wallet);
     feeWallet = wallet;
     feeExempt[wallet] = true; // auto exempt fee wallet
     emit FeeExemptUpdated(wallet, true);
}
```

```
constructor(uint256 initialSupply, address _feeWallet) ERC20("AIFortuna
    Game Token", "AGT") Ownable(msg.sender) {
     uint256 mintAmount = initialSupply * 10 ** decimals();
     _mint(msg.sender, mintAmount);
     feeExempt[msg.sender] = true;
     feeWallet = _feeWallet;
```

```
    gameFeeBps = 9000;

    emit AGTInitialized(msg.sender, mintAmount, block.timestamp);
    emit FeeWalletUpdated(address(0), _feeWallet);
}
```

IMPACT:

Initial fee wallet may be subject to transfer fees, creating inconsistent behavior and potential unexpected fee deductions when the fee wallet performs token transfers.

RECOMMENDATIONS:

Add auto exemption logic in constructor to maintain consistency.

```
constructor(uint256 initialSupply, address _feeWallet) ERC20("AIFortuna
    Game Token", "AGT") Ownable(msg.sender) {
    uint256 mintAmount = initialSupply * 10 ** decimals();
    _mint(msg.sender, mintAmount);
    feeExempt[msg.sender] = true;
    feeWallet = _feeWallet;
+   feeExempt[_feeWallet] = true;

    gameFeeBps = 9000;

    emit AGTInitialized(msg.sender, mintAmount, block.timestamp);
    emit FeeWalletUpdated(address(0), _feeWallet);
+   emit FeeExemptUpdated(_feeWallet, true);
}
```

3.13 setFeeWallet fails to revoke old fee wallet exemption status

SEVERITY: LOW **STATUS:** Fixed

PATH:

`src/AGT.sol`

DESCRIPTION:

`setFeeWallet()` updates the fee wallet address but fails to revoke the fee exemption status of the previous fee wallet. This allows the old fee wallet to retain `feeExempt = true` status even after being replaced.

```
function setFeeWallet(address wallet) external onlyOwner {
    require(wallet != address(0), "op zero");
    emit FeeWalletUpdated(feeWallet, wallet);
    feeWallet = wallet;
    feeExempt[wallet] = true; // auto exempt fee wallet
    emit FeeExemptUpdated(wallet, true);
}
```

IMPACT:

Previous fee wallet addresses retain fee exemption privileges indefinitely, allowing unauthorized fee-free transfers.

RECOMMENDATIONS:

Revoke old fee wallet exemption status when updating.

```
function setFeeWallet(address wallet) external onlyOwner {
    require(wallet != address(0), "op zero");
+   address oldFeeWallet = feeWallet;
    emit FeeWalletUpdated(feeWallet, wallet);
    feeWallet = wallet;
+
+   // Revoke old fee wallet exemption
```

```
+    if (oldFeeWallet != address(0) && oldFeeWallet != wallet) {
+        feeExempt[oldFeeWallet] = false;
+        emit FeeExemptUpdated(oldFeeWallet, false);
+    }
+
     feeExempt[wallet] = true; // auto exempt fee wallet
     emit FeeExemptUpdated(wallet, true);
}
```

3.14 Comment and code mismatch for agt_pair_feeBps rate

SEVERITY: LOW **STATUS:** Fixed

PATH:

`src/FUSD.sol`

DESCRIPTION:

`agt_pair_feeBps` variable has a comment stating “3% default” but the actual value is 1000 basis points, which equals 10%.

```
uint256 public agt_pair_feeBps = 1000; // 3% default (basis points)
```

IMPACT:

Comment and code mismatch may cause confusion during code review and maintenance.

RECOMMENDATIONS:

Fix the comment to match the actual value or adjust the value to match the comment.

```
-uint256 public agt_pair_feeBps = 1000; // 3% default (basis points)
+uint256 public agt_pair_feeBps = 300; // 3% default (basis points)
```

3.15 Event parameter mismatch in NodeCardFeeReceived

SEVERITY: LOW **STATUS:** Fixed

PATH:

`src/FUSD.sol`

DESCRIPTION:

The `NodeCardFeeReceived` event is emitted with incorrect parameter order. The first parameter should be `from` (the address paying the fee) but it's currently using `to` (the agtSwapPair address).

```
if (feeAmount > 0) {
    //Sell FUSD for age, so trasnfer FUSD to agtSwapPair
    if (to == agtSwapPair) {
        (uint256 gameFee, uint256 teamFee) = _calculateFees(feeAmount);

        _transfer(from, team, teamFee);
        emit FeeReceived(from, team, teamFee);

        _transfer(from, gameContract, gameFee);
        emit NodeCardFeeReceived(to, gameContract, gameFee);
    } else {
        _transfer(from, team, feeAmount);
        emit FeeReceived(from, team, feeAmount);
    }
}
```

IMPACT:

Incorrect event parameter logging affects off-chain monitoring and analytics systems.

RECOMMENDATIONS:

Fix the event emission to use the correct parameter.

```
-     emit NodeCardFeeReceived(to, gameContract, gameFee);
```

```
+     emit NodeCardFeeReceived(from, gameContract, gameFee);
```

3.16 AGT pair whitelist and fee bypass

SEVERITY: LOW **STATUS:** Fixed

PATH:

`src/FUSD.sol`

DESCRIPTION:

When `from == agtSwapPair`, the code branch is empty, neither charging fees nor performing whitelist checks, allowing "buy FUSD" transactions from AGT-FUSD pair to bypass whitelist restrictions and fees.

```
if (isTrade && !feeExempt[from] && !feeExempt[to]) {
    if (to == agtSwapPair) {
        //  Buy AGT
        feeAmount = (amount * agt_pair_feeBps) / FEE_DENOMINATOR;
        transferAmount = amount - feeAmount;
    } else if (from == agtSwapPair) {
        // Sell AGT
    } else {
        // IF sell FUSD to pair, we charge fee
        if (swapPairs[to]) {
            feeAmount = (amount * feeBps) / FEE_DENOMINATOR;
            transferAmount = amount - feeAmount;
        } else {
            // If buy FUSD from pair, only allow whitelisted address
            require(whitelist[to], "Not whitelisted");
        }
    }
}
```

IMPACT:

AGT pair transactions can bypass both whitelist checks and fee charges, creating inconsistent fee logic.

RECOMMENDATIONS:

Make sure the logic here should be Add whitelist check and fee logic to the empty branch or not.

```
if (isTrade && !feeExempt[from] && !feeExempt[to]) {
    if (to == agtSwapPair) {
        //  Buy AGT
        feeAmount = (amount * agt_pair_feeBps) / FEE_DENOMINATOR;
        transferAmount = amount - feeAmount;
    } else if (from == agtSwapPair) {
        // Sell AGT
+       // charge fee logic or not
    } else {
        // IF sell FUSD to pair, we charge fee
        if (swapPairs[to]) {
            feeAmount = (amount * feeBps) / FEE_DENOMINATOR;
            transferAmount = amount - feeAmount;
        } else {
            // If buy FUSD from pair, only allow whitelisted address
            require(whitelist[to], "Not whitelisted");
        }
    }
}
```

3.17 Overwrite existing data

SEVERITY: LOW **STATUS:** Fixed

PATH:

src/Fortuna.sol

DESCRIPTION:

When the Owner executes the addPancakeSwapInfos operation, it does not check whether the Token has been created. If it is added repeatedly, there will be a risk of data overwriting.

```
function addPancakeSwapInfos(address token, address pairedToken) external
    onlyOwner {
     pancakeSwapInfos[token] =
         PancakeSwapInfo({isSupported: true, pairedToken: pairedToken,
    buyFeePercent: 0, buySupported: false});

     emit PancakeSwapInfoAdded(msg.sender, token, pairedToken,
    block.timestamp);
}
```

IMPACT:

Adding an existing token will cause the data to be overwritten, which may cause the contract to not function properly.

RECOMMENDATIONS:

Verify that it exists before executing.

```
function addPancakeSwapInfos(address token, address pairedToken) external
    onlyOwner {
+    require(!pancakeSwapInfos[token].isSupported, "Token already exists");
     pancakeSwapInfos[token] =
         PancakeSwapInfo({isSupported: true, pairedToken: pairedToken,
    buyFeePercent: 0, buySupported: false});
```

```
    emit PancakeSwapInfoAdded(msg.sender, token, pairedToken,
    block.timestamp);
}
```

3.18 Incorrect judgment causes the logic to never trigger

SEVERITY: LOW **STATUS:** Fixed

PATH:

`src/Fortuna.sol`

DESCRIPTION:

In the `swapTokensForTokens` operation, only `fusd -> usdt` and `agt -> fusd` are allowed. However, in the subsequent logic, `usdt` will be checked. Such an incorrect judgment will cause the related operations to never be executed.

```
function swapTokensForTokens(address tokenIn, address tokenOut, uint256
    amountIn, address to, uint256 deadline)
     external
     payable
     nonReentrant
     fundBNB
{
     // ...
     // Restrict buying, only allow selling
     if (
         !(
             (path[0] == address(fusd) && path[1] == address(usdt))
                 || (path[0] == address(agt) && path[1] == address(fusd))
         )
     ) revert OnlySellPathsAllowed();

     uint256 amountOut = _swapSupportingFeeReturnOut(amountIn, path,
    deadline);
     IERC20(tokenOut).safeTransfer(to, amountOut);

     if (path[0] == address(usdt) || path[1] == address(usdt)) {
         // handle USDT <-> FUSD
         if (path[0] == address(usdt)) {
             treasuryHedge.execute(amountOut, true);
         } else {
             treasuryHedge.execute(amountOut, false);
```

```
        }
    }

    //emit Event
    emit Swap(msg.sender, path[0], path[1], amountIn, amountOut);
}
```

IMPACT:

Since only selling is allowed in the function, `path[0] == address(usdt)` will never be true, and subsequent logic will not be executed, which may affect the normal operation of the contract.

RECOMMENDATIONS:

Delete useless judgment logic, or modify it according to project requirements.

3.19 Stale Balance Check Potential Transaction Failures

SEVERITY: LOW **STATUS:** Fixed

PATH:

src/Treasury.sol

DESCRIPTION:

_executeUSDTAcquisition and _executeUSDTReduction use stale balance values for the second operation check, fail when attempting to add liquidity with insufficient balance.

```
function _executeUSDTAcquisition(uint256 hedgeAmount, bool isUpZone)
    internal {
    uint256 treasuryFUSD = fusd.balanceOf(address(this));
    uint256 lpAmount = (hedgeAmount * feeToLPBps) / BPS;
    uint256 swapAmount = hedgeAmount - lpAmount;

    if (treasuryFUSD >= hedgeAmount) {
        _swap(true, swapAmount, isUpZone); // FUSD -> USDT
    }

    if (lpAmount > 0 && treasuryFUSD >= lpAmount) {
        // Add portion to LP (10% default)
        _addLP(address(fusd), address(usdt), lpAmount);
    }
}

function _executeUSDTReduction(uint256 hedgeAmount, bool isUpZone)
    internal {
    uint256 treasuryUSDT = usdt.balanceOf(address(this));
    uint256 lpAmount = (hedgeAmount * feeToLPBps) / BPS;
    uint256 swapAmount = hedgeAmount - lpAmount;

    if (treasuryUSDT >= swapAmount && swapAmount > 0) {
        // Use USDT to acquire FUSD (reduce USDT exposure)
        _swap(false, swapAmount, isUpZone); // USDT -> FUSD
    }
```

```
    if (lpAmount > 0 && treasuryUSDT >= lpAmount) {
        // Add portion to LP (10% default)
        _addLP(address(usdt), address(fusd), lpAmount);
    }
}
```

IMPACT:

Using stale balance values after swap operations can cause `addLiquidity` to fail due to insufficient balance.

RECOMMENDATIONS:

Refresh the balance before the second operation check to ensure sufficient funds are available.

```
function _executeUSDTAcquisition(uint256 hedgeAmount, bool isUpZone)
   internal {
    uint256 treasuryFUSD = fusd.balanceOf(address(this));
    uint256 lpAmount = (hedgeAmount * feeToLPBps) / BPS;
    uint256 swapAmount = hedgeAmount - lpAmount;

    if (treasuryFUSD >= hedgeAmount) {
        _swap(true, swapAmount, isUpZone); // FUSD -> USDT
+       treasuryFUSD = fusd.balanceOf(address(this));
    }

    if (lpAmount > 0 && treasuryFUSD >= lpAmount) {
        // Add portion to LP (10% default)
        _addLP(address(fusd), address(usdt), lpAmount);
    }
}
```

3.20 Comment Implementation Mismatch in Alpha Calculation

SEVERITY: LOW **STATUS:** Fixed

PATH:

`src/Treasury.sol`

DESCRIPTION:

`computeAlpha` has inconsistent comments and implementation for the alpha calculation curves.

```
function computeAlpha(HedgeParams memory p) public view returns (uint256
    alphaBps) {
     //...

     uint256 gamma = t.smootherGamma(gamma); // 1e18
     // Four curves mapping:
     // Up zone: userBuy -> 50%->100%; userSell -> 45%->90%
     // Down zone: userBuy -> 50%->10%; userSell -> 50%->100%
     // Interpolate: alpha = 0.5 +/- 0.4 * s
     // Return in basis points (x 100%)
     if (p.isUpZone && p.isBuy) {
         // 0.5 -> 1
         alphaBps = 5000 + (5000 * s) / 1e18;
     } else if (p.isUpZone && !p.isBuy) {
         // up zone sell: 0.45 -> 0.95
         alphaBps = 4500 + (5000 * s) / 1e18;
     } else if (!p.isUpZone && p.isBuy) {
         // down zone buy: 0.5 -> 0.9
         alphaBps = 5000 - (5000 * s) / 1e18;
     } else {
         // down zone sell: 0.5 -> 1
         alphaBps = 5000 + (5000 * s) / 1e18;
     }
}
```

IMPACT:

Comment and implementation inconsistency may cause confusion during code review and maintenance.

RECOMMENDATIONS:

Align comments with implementation or vice versa to ensure consistency.

3.21 Operator Array Management Without pop Leading to Duplicates

SEVERITY: LOW **STATUS:** Fixed

PATH:

`src/Treasury.sol`

DESCRIPTION:

`setOperator` has flawed array management that duplicate operators in the array when re-enabling previously disabled operators.

```
function setOperator(address operator, bool enabled) external onlyOwner {
    if (enabled) {
        if (!operatorsMap[operator]) {
            operators.push(operator);
            operatorsMap[operator] = true;
            emit OperatorAdded(operator, msg.sender, block.timestamp);
        }
    } else {
        if (operatorsMap[operator]) {
            operatorsMap[operator] = false;
            emit OperatorRemoved(operator, msg.sender, block.timestamp);
        }
    }
}
```

IMPACT:

When an operator is disabled and then re-enabled, they will appear multiple times in the `operators` array while only having one entry in `operatorsMap`.

RECOMMENDATIONS:

Remove operators from the array when disabling them to maintain consistency.

```
function setOperator(address operator, bool enabled) external onlyOwner {
```

```
    if (enabled) {
        if (!operatorsMap[operator]) {
            operators.push(operator);
            operatorsMap[operator] = true;
            emit OperatorAdded(operator, msg.sender, block.timestamp);
        }
    } else {
        if (operatorsMap[operator]) {
            operatorsMap[operator] = false;
+           // Remove from array
+           for (uint256 i = 0; i < operators.length; i++) {
+               if (operators[i] == operator) {
+                   operators[i] = operators[operators.length - 1];
+                   operators.pop();
+                   break;
+               }
+           }
            emit OperatorRemoved(operator, msg.sender, block.timestamp);
        }
    }
}
```

3.22 Inefficient WithdrawCount Initialization Wastes Storage Slot

SEVERITY: LOW **STATUS:** Fixed

PATH:

src/Fortuna.sol

DESCRIPTION:

withdrawCount is initialized to 1, but the first withdraw request actually uses ID 2, wasting the withdraws[1] storage slot.

```
function initialize(...) external initializer {
    // ...
    withdrawCount = 1;
    // ...
}

function withdrawRequest(address token, uint256 amount) external
   nonReentrant {
    // ...
    withdrawCount++;
    withdraws[withdrawCount] = Withdraw({
        user: msg.sender,
        token: token,
        amount: amount,
        timestamp: block.timestamp,
        isConfirmed: false,
        isCanceled: false
    });
    playerWithdrawRequest[msg.sender] = withdrawCount;
    // ...
}
```

IMPACT:

withdraws[withdrawCount] starting count is 2 but not 1.

RECOMMENDATIONS:

Initialize `withdrawCount` to 0, so the first withdraw request uses ID 1.

```
function initialize(...) external initializer {
    // ...
-   withdrawCount = 1;
+   withdrawCount = 0;
    // ...
}
```

3.23 Missing Balance Validation in claimNodeCardRewards

SEVERITY: LOW **STATUS:** Acknowledge

PATH:

`src/Fortuna.sol`

DESCRIPTION:

`claimNodeCardRewards` lacks balance validation checks before attempting to swap tokens.

```
function claimNodeCardRewards(
    address tokenA,
    uint256 amount,
    address tokenB,
    uint256 amountB,
    uint256 deadline,
    string memory signContext,
    bytes memory signature
) external payable nonReentrant fundBNB {
    require(deadline >= block.timestamp, "Signature expired");
    require(!usedSignContexts[signContext], "Sign context already used");

    bytes32 message = keccak256(
        abi.encodePacked(block.chainid, msg.sender, tokenA, amount,
   tokenB, amountB, signContext, deadline)
    );

    _verifySignature(message, signature);
    usedSignContexts[signContext] = true;

    if (amount > 0) {
        //A => B
        address[] memory path = new address[](2);
        path[0] = tokenA;
        path[1] = tokenB;

        _approveExact(tokenA, amount);
```

```
        uint256 claim_fee = (amount * claimFeePercent) / BPS;
        uint256 sell_amount = amount - claim_fee;
        uint256 amountPairedOut =
    _swapSupportingFeeReturnOut(sell_amount, path, deadline);

        emit PairedTokenARewardsClaimed(
            msg.sender, tokenA, sell_amount, tokenB, amountPairedOut,
    claimFeePercent, signContext, signature
        );
     }

     if (amountB > 0) {
        //B => A
        address[] memory path = new address[](2);
        path[0] = tokenB;
        path[1] = tokenA;
        _approveExact(tokenB, amountB);

        uint256 claim_fee = (amount * claimFeePercent) / BPS;
        uint256 sell_amount = amount - claim_fee;

        uint256 amountPairedOut =
    _swapSupportingFeeReturnOut(sell_amount, path, deadline);
        emit PairedTokenBRewardsClaimed(
            msg.sender, tokenB, sell_amount, tokenA, amountPairedOut,
    claimFeePercent, signContext, signature
        );
     }
}
```

IMPACT:

Function may fail due to insufficient balance without proper validation checks.

RECOMMENDATIONS:

Add balance validation checks before attempting to swap tokens, also the same problem in `dailyBurnToken`.

```
if (amount > 0) {
    //A => B
```

```
+   require(amount <= getBalance(tokenA), "Insufficient tokenA balance");
    address[] memory path = new address[](2);
    path[0] = tokenA;
    path[1] = tokenB;

    _approveExact(tokenA, amount);
    // ...
}

if (amountB > 0) {
    //B => A
+   require(amountB <= getBalance(tokenB), "Insufficient tokenB balance");
    address[] memory path = new address[](2);
    path[0] = tokenB;
    path[1] = tokenA;
    _approveExact(tokenB, amountB);
    // ...
}
```

3.24 Setter methods lack duplicate value validation

SEVERITY: INFO **STATUS:** Fixed

PATH:

`src/AGT.sol`

DESCRIPTION:

Multiple setter methods in AGT.sol execute storage writes and emit events even when the new value is identical to the current value. Include `setFeeExempt()`, `setSwapPair()`, `setFeeWallet()`, `setFeeBps()`, `setGameFeeBps()`, `setGameContract()`, and `addToWhitelist()`.

```
function setFeeExempt(address a, bool s) external onlyOwner {
    feeExempt[a] = s;
    emit FeeExemptUpdated(a, s);
}
```

IMPACT:

Unnecessary gas consumption for redundant operations, event log.

RECOMMENDATIONS:

Add duplicate value validation before state modifications.

```
function setFeeExempt(address a, bool s) external onlyOwner {
+   require(feeExempt[a] != s, "Value unchanged");
    feeExempt[a] = s;
    emit FeeExemptUpdated(a, s);
}

function setFeeWallet(address wallet) external onlyOwner {
    require(wallet != address(0), "op zero");
+   require(feeWallet != wallet, "Value unchanged");
    emit FeeWalletUpdated(feeWallet, wallet);
    feeWallet = wallet;
```

```
    feeExempt[wallet] = true;
    emit FeeExemptUpdated(wallet, true);
}
```

3.25 Constructor lacks FeeExemptUpdated event emission

SEVERITY: INFO **STATUS:** Fixed

PATH:

`src/AGT.sol`

DESCRIPTION:

The constructor sets `feeExempt[msg.sender] = true` without emitting the corresponding `FeeExemptUpdated` event, while other functions that modify `feeExempt` mapping consistently emit this event.

IMPACT:

Incomplete event tracking for fee exemption status changes, affecting off-chain monitoring systems.

RECOMMENDATIONS:

Add missing event emission in constructor.

```
constructor(uint256 initialSupply, address _feeWallet) ERC20("AIFortuna
    Game Token", "AGT") Ownable(msg.sender) {
    uint256 mintAmount = initialSupply * 10 ** decimals();
    _mint(msg.sender, mintAmount);
    feeExempt[msg.sender] = true;
    feeWallet = _feeWallet;

    gameFeeBps = 9000;

    emit AGTInitialized(msg.sender, mintAmount, block.timestamp);
    emit FeeWalletUpdated(address(0), _feeWallet);
+   emit FeeExemptUpdated(msg.sender, true);
}
```

3.26 Unused AccessControl import increases deployment cost

SEVERITY: INFO **STATUS:** Fixed

PATH:

`src/AGT.sol`

DESCRIPTION:

Contract imports `@openzeppelin/contracts/access/AccessControl.sol` but never inherits from it or uses any of its functionality. All access control is implemented through the Ownable contract.

```
import "@openzeppelin/contracts/access/AccessControl.sol";

contract AGT is ERC20, Ownable, ReentrancyGuard {
```

IMPACT:

Unused imports increase deployment cost and contract size unnecessarily.

RECOMMENDATIONS:

Remove unused import.

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
-import "@openzeppelin/contracts/access/AccessControl.sol";
```

3.27 Unnecessary recoverETH function

SEVERITY: INFO **STATUS:** Fixed

PATH:

`src/FUSD.sol`

DESCRIPTION:

`recoverETH()` is unnecessary because the FUSD contract cannot receive ETH. The contract has no `receive()` or `fallback()` functions, and no functions are marked as `payable`.

```
function recoverETH() external onlyOwner {
    address recipient = owner();
    uint256 amount = address(this).balance;
    payable(recipient).transfer(amount);
    emit ETHRecovered(recipient, amount, block.timestamp);
}
```

IMPACT:

Dead code that serves no purpose as the contract cannot receive ETH.

RECOMMENDATIONS:

Remove the `recoverETH()` function since the contract cannot receive ETH.

3.28 Misspelling of words

SEVERITY: INFO **STATUS:** Fixed

PATH:

`src/FUSD.sol`

DESCRIPTION:

Note word spelling errors

IMPACT:

Code comment spelling errors affect code readability and professionalism.

RECOMMENDATIONS:

Fix spelling errors in comments.

```
function _transferWithFee(address from, address to, uint256 amount)
    internal returns (bool) {

-       // Sell FUSD for age, so trasnfer FUSD to agtSwapPair
+       // Sell FUSD for agt, so trasnfer FUSD to agtSwapPair
}
```

3.29 Missing zero address check

SEVERITY: INFO **STATUS:** Fixed

PATH:

`src/Fortuna.sol`

DESCRIPTION:

In the src/Fortuna.sol file, some configuration functions related to setting addresses, such as `setPancakeRouter`, `addPancakeSwapInfos`, and `addToken`, do not verify the existence of zero addresses in advance, which may pose a security risk.

```
function setPancakeRouter(address _pancakeRouter) external onlyOwner {
    pancakeRouter = _pancakeRouter;
    emit PancakeRouterUpdated(msg.sender, _pancakeRouter,
    block.timestamp);
}

function addPancakeSwapInfos(address token, address pairedToken) external
    onlyOwner {
    pancakeSwapInfos[token] =
        PancakeSwapInfo({isSupported: true, pairedToken: pairedToken,
    buyFeePercent: 0, buySupported: false});

    emit PancakeSwapInfoAdded(msg.sender, token, pairedToken,
    block.timestamp);
}

function addToken(address token, uint256 minDeposit) external onlyOwner {
    supportedTokens[token] = TokenInfo({isSupported: true, minDeposit:
    minDeposit});
    tokenList.push(token);

    emit TokenAdded(msg.sender, token, minDeposit, block.timestamp);
}
```

IMPACT:

If a zero address is passed in, the contract may run abnormally, affecting the user experience.

RECOMMENDATIONS:

Add `require(token != address(0), "Token address cannot be zero");` related checks. Including some other functions that set or modify addresses need to be verified.

3.30 Missing minDeposits Parameter in FortunaInitialized Event

SEVERITY: INFO **STATUS:** Fixed

PATH:

`src/Fortuna.sol`

DESCRIPTION:

`FortunaInitialized` event does not emit the `_minDeposits` parameter, but other inputs are all recorded.

```
function initialize(
    address[] memory _gameTokens,
    uint256[] memory _minDeposits,
    address _oracle,
    address _feeWallet,
    uint256 _feePercent,
    address _distributeAddress,
    address _pancakeRouter,
    address _usdt,
    address _treasuryHedge,
    address _fusd,
    address _agt,
    uint256 _claimFeePercent
) external initializer {
    //...

    emit FortunaInitialized(
        msg.sender,
        _gameTokens,
        _oracle,
        _feeWallet,
        _feePercent,
        _distributeAddress,
        _pancakeRouter,
        _usdt,
        _treasuryHedge,
        _fusd,
```

```
        _agt,
        _claimFeePercent,
        block.timestamp
    );
}
```

IMPACT:

Incomplete event logging affects off-chain monitoring and data analysis systems.

RECOMMENDATIONS:

Add the `_minDeposits` parameter to both the event definition and emit statement.

```
event FortunaInitialized(
    address indexed owner,
    address[] tokens,
+   uint256[] minDeposits,
    address oracle,
    address feeWallet,
    uint256 feePercent,
    address distributeAddress,
    address pancakeRouter,
    address usdt,
    address treasuryHedge,
    address fusd,
    address agt,
    uint256 claimFeePercent,
    uint256 timestamp
);

emit FortunaInitialized(
    msg.sender,
    _gameTokens,
+   _minDeposits,
    _oracle,
    _feeWallet,
    _feePercent,
    _distributeAddress,
    _pancakeRouter,
    _usdt,
```

```
    _treasuryHedge,
    _fusd,
    _agt,
    _claimFeePercent,
    block.timestamp
);
```

3.31 withdrawConfirm External Transfer Before State Update

SEVERITY: INFO **STATUS:** Fixed

PATH:

`src/Fortuna.sol`

DESCRIPTION:

`withdrawConfirm` transfer before updating state variables, violating the CEI pattern.

```
function withdrawConfirm(uint256 withdrawId, address user, address token,
    uint256 amount, bytes memory signature)
     external
     nonReentrant
     onlyRole(OPERATOR_ROLE)
{
     require(!withdraws[withdrawId].isConfirmed, "Withdraw request is
    confirmed");
     require(!withdraws[withdrawId].isCanceled, "Withdraw request is
    canceled");
     require(withdraws[withdrawId].user == user, "You are not the user of
    this withdraw request");
     require(withdraws[withdrawId].token == token, "Token mismatch");
     require(withdraws[withdrawId].amount == amount, "Amount mismatch");

     oracleNonce++;
     bytes32 message = keccak256(abi.encodePacked(block.chainid, user,
    token, amount, withdrawId, oracleNonce));
     _verifySignature(message, signature);

     IERC20(token).safeTransfer(user, amount);

     playerWithdraw[user][token] += amount;
     totalWithdraw[token] += amount;
     withdraws[withdrawId].isConfirmed = true;

     emit WithdrawConfirm(msg.sender, user, token, amount,
    block.timestamp, withdrawId);
}
```

IMPACT:

Violates CEI pattern best practices but poses low risk due to `nonReentrant` protection.

RECOMMENDATIONS:

```
function withdrawConfirm(uint256 withdrawId, address user, address token,
    uint256 amount, bytes memory signature)
     external
     nonReentrant
     onlyRole(OPERATOR_ROLE)
{
     // ...

-    IERC20(token).safeTransfer(user, amount);

     playerWithdraw[user][token] += amount;
     totalWithdraw[token] += amount;
     withdraws[withdrawId].isConfirmed = true;

+    IERC20(token).safeTransfer(user, amount);

     emit WithdrawConfirm(msg.sender, user, token, amount,
    block.timestamp, withdrawId);
}
```

4. CONCLUSION

In this audit, we thoroughly analyzed **Aifortuna** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be **PASSED**.

To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

Description	The security of apply verification
Result	Not found
Severity	CRITICAL

5.1.2 Authorization Access Control

Description	Permission checks for external integral functions
Result	Not found
Severity	CRITICAL

5.1.3 Forged Transfer Vulnerability

Description	Assess whether there is a forged transfer notification vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.4 Transaction Rollback Attack

Description	Assess whether there is transaction rollback attack vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.5 Transaction Block Stuffing Attack

Description	Assess whether there is transaction blocking attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.6 Soft Fail Attack Assessment

Description	Assess whether there is soft fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.7 Hard Fail Attack Assessment

Description	Examine for hard fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.8 Abnormal Memo Assessment

Description	Assess whether there is abnormal memo vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.9 Abnormal Resource Consumption

Description	Examine whether abnormal resource consumption in contract processing
Result	Not found
Severity	CRITICAL

5.1.10 Random Number Security

Description	Examine whether the code uses insecure random number
Result	Not found
Severity	CRITICAL

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

Description	Examine for weakness in cryptograph implementation
Result	Not found
Severity	HIGH

5.2.2 Account Permission Control

Description	Examine permission control issue in the contract
Result	Not found
Severity	MEDIUM

5.2.3 Malicious Code Behavior

Description	Examine whether sensitive behavior present in the code
Result	Not found
Severity	MEDIUM

5.2.4 Sensitive Information Disclosure

Description	Examine whether sensitive information disclosure issue present in the code
Result	Not found
Severity	MEDIUM

5.2.5 System API

Description	Examine whether system API application issue present in the code
Result	Not found
Severity	LOW

6. DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without ExVul's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts ExVul to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

7. REFERENCES

[1] MITRE. CWE-191: Integer Underflow (Wrap or Wraparound). https://cwe.mitre.org/data/definitions/191.html.

[2] MITRE. CWE-197: Numeric Truncation Error. https://cwe.mitre.org/data/definitions/197.html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption. https://cwe.mitre.org/data/definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation. https://cwe.mitre.org/data/definitions/440.html.

[5] MITRE. CWE-684: Protection Mechanism Failure. https://cwe.mitre.org/data/definitions/693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems. https://cwe.mitre.org/data/definitions/438.html.

[8] MITRE. CWE CATEGORY: Numeric Errors. https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors. https://cwe.mitre.org/data/definitions/399.html.

[10] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

8. About Exvul Security

Premier Security for the Web3 Ecosystem

ExVul is a premier Web3 security firm committed to forging a secure and trustworthy decentralized ecosystem. Our elite team consists of security veterans from world-leading technology and blockchain security firms, including Huawei, YBB Captical, Qihoo 360, Amber, ByteDance, MoveBit, and PeckShield. Team member Nolan is ranked as a top-40 whitehat on Immunefi and is the platform's sole All-Star in the APAC region.

Our expertise covers the full spectrum of Web3 security. We conduct **meticulous smart contract audits**, having fortified thousands of projects on chains like Evm, Solana, Aptos, Sui etc. Our **Blockchain Protocol Audits** secure the core infrastructure of L1/L2 by uncovering deep-seated vulnerabilities. We also offer **comprehensive wallet audits** to protect user assets and provide **proactive web3 pentest**, enabling partners to neutralize threats before they strike.

Trusted by industry leaders, ExVul is the security partner for **OKX, Bitget, Cobo, Infini, Stacks, Aptos, Sui, CoreDAO, Sei** etc.

Contact

Website
www.exvul.com

Email
contact@exvul.com

Twitter
@EXVULSEC

Github
github.com/EXVUL-Sec