ExVul

SMART CONTRACT AUDIT REPORT

Crash Game Smart Contract

JANUARY 2025

Contents

1. EXECUTIVE SUMMARY

ExVul Web3 Security was engaged by **Crash Game** 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	Informational	Low	Medium	High
High	INFO	MEDIUM	HIGH	CRITICAL
Medium	INFO	LOW	MEDIUM	HIGH
Low	INFO	LOW	LOW	MEDIUM
		IMPACT		

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
Crash Game	January 15 2025 — January 20 2025	Solidity

Source code	Link
Crash Game	Private Repository (Under NDA)
Commit Hash	Reviewed Latest Version

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	1
MEDIUM	4
LOW	2
INFO	2

2.3 Key Findings

Severity	Findings Title	Status
HIGH	Round Zero Investment Lock Vulnerability	Acknowledge
MEDIUM	Missing Upper Limit for Fee Percentage	Acknowledge
MEDIUM	Accidental ETH and ERC-20 Token Lock-up	Acknowledge
MEDIUM	Old Game Contract Funds Not Migrated	Acknowledge
MEDIUM	Batch Number Zero DoS Vulnerability	Acknowledge
LOW	Immediacy Risk of gameContract Updates	Acknowledge
LOW	Missing Zero-Address Check in setOracles	Acknowledge
INFO	Inconsistent Event Parameter Naming	Acknowledge
INFO	Owner Can Set Equal Min/Max Parameters	Acknowledge

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Round Zero Investment Lock Vulnerability

SEVERITY: HIGH **STATUS:** Acknowledge

PATH:

`CrashoutGame.sol - _validateJoinRound(), joinRound()`

DESCRIPTION:

Users can place bets in uninitialized round 0, causing permanent fund lockup. After contract initialization, currentRoundId = 0 and rounds[0].status = Pending by default, allowing _validateJoinRound() to pass. When oracle starts the official game via startRound(), it increments the currentRoundId to 1, permanently abandoning round 0.

Round 0 cannot be closed because roundId != currentRoundId (0 != 1), and rewards cannot be distributed because keccak256(multiplier, salt) can never equal the default multiplierHash of 0x0. User funds remain locked in the pool with no recovery mechanism.

The vulnerable code section:

```
function _validateJoinRound(uint256 roundId, uint256 betAmount) private
    view {
     if (roundId != currentRoundId) revert InvalidRound();           // 0
        == 0 passes
     if (rounds[roundId].status != RoundStatus.Pending) revert
        RoundNotPending(); // Pending passes
     // MISSING: Check if rounds[roundId].multiplierHash == 0x0
}
```

RECOMMENDATIONS:

Add initialization check in _validateJoinRound():

```
function _validateJoinRound(uint256 roundId, uint256 betAmount) private
    view {
     if (roundId != currentRoundId) revert InvalidRound();
     if (rounds[roundId].status != RoundStatus.Pending) revert
         RoundNotPending();
     if (rounds[roundId].multiplierHash == 0x0) revert
         RoundNotInitialized(); // FIX
     // ... rest of validation
}

error RoundNotInitialized();
```

3.2 Missing Upper Limit for Fee Percentage

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

`CrashoutGame.sol - _updateFeePercentage()`

DESCRIPTION:

The _updateFeePercentage function in the CrashoutGame contract allows the contract owner to set the game's feePercentage. While the function includes a check _feePercentage <= FEE_BASE, the FEE_BASE constant is defined as 10_000. This implies that feePercentage can theoretically be set to 10_000.

According to the fee calculation formula fee = betAmount.mulDiv(feePercentage, FEE_BASE), if feePercentage is set to 10_000, the entire amount the user bets would be taken as a fee, leaving zero as the actual principal for participation in the round.

The vulnerable code section:

```
function _updateFeePercentage(uint256 _feePercentage) internal {
    require(_feePercentage <= FEE_BASE, InvalidConfig());
    feePercentage = _feePercentage;
}
```

While contract owners are generally expected to act in the best interest of the project, the lack of an explicit, lower upper bound for the fee percentage could lead to several issues:

- **Operational Risk**: An unintentional operational error, such as a typo when setting the fee (e.g., adding an extra zero), could result in the fee being set to an unexpectedly high level, inadvertently eroding user funds.
- **Trust Risk**: This absence of a code-enforced cap means users must fully trust the project team not to raise the fee percentage to an unreasonable level (e.g., 100%). In decentralized applications, users typically expect the code itself to provide strong assurances.
- **Economic Model Stability**: Although not leading to an immediate drain of the fund pool, excessively high fees would quickly stifle user participation, thereby impacting the game's long-term economic health and sustainability.

RECOMMENDATIONS:

We recommend implementing a more stringent and explicit upper limit for the feePercentage within the _updateFeePercentage function. This cap should represent the maximum acceptable percentage based on the game's economic model, for example, 5% or 10%, which would correspond to feePercentage values of 500 or 1000 (given FEE_BASE of 10,000).

```
uint256 public constant MAX_FEE_PERCENTAGE = 1000; // 10% maximum

function _updateFeePercentage(uint256 _feePercentage) internal {
    require(_feePercentage <= MAX_FEE_PERCENTAGE, ”Fee percentage too
       high”);
    feePercentage = _feePercentage;
}
```

3.3 Accidental ETH and ERC-20 Token Lock-up

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

`CrashoutGame.sol - receive() function`

DESCRIPTION:

The CrashoutGame contract includes a receive() external payable {} function, allowing it to accept and hold Ether. In the normal operational flow of the game, user bet funds (ETH) are intended to be sent directly to the CrashoutPool contract, not the CrashoutGame contract itself.

However, if the following scenarios occur, ETH or other ERC-20 tokens could be accidentally sent to and become permanently locked within the CrashoutGame contract:

1. **User Error or External Misdirection**: Any user or external contract unfamiliar with the intended design might mistakenly send ETH or ERC-20 tokens directly to the CrashoutGame contract address. These funds would be received by the contract's receive() function and subsequently become inaccessible.
2. **Future Logical Errors**: While not apparent in the current codebase, future modifications to the contract might inadvertently route funds to the CrashoutGame contract.

The current contract lacks any functions callable by the contract owner or other authorized roles to withdraw these accidentally received ETH or other ERC-20 tokens. This means that once funds are mistakenly sent, they cannot be recovered, leading to their permanent loss.

RECOMMENDATIONS:

It is strongly recommended to add owner-controlled withdrawal functions to the CrashoutGame contract. These functions would enable the recovery of ETH and ERC-20 tokens that are accidentally sent to or become stuck within the contract.

```
function rescueETH(address to, uint256 amount) external onlyOwner {
    require(to != address(0), "Invalid address");
    require(amount <= address(this).balance, "Insufficient balance");
    (bool success, ) = to.call{value: amount}("");
    require(success, "Transfer failed");
}

function rescueToken(address token, address to, uint256 amount) external
    onlyOwner {
     require(to != address(0), ”Invalid address”);
     IERC20(token).transfer(to, amount);
}
```

3.4 Batch Number Zero DoS Vulnerability

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

`CrashoutGame.sol - updateDistributeRewardsNumber(), distributeRewards()`

DESCRIPTION:

The updateDistributeRewardsNumber() function allows owner to set distributeRewardsBatchNumber to 0 without validation. When set to 0, the distributeRewards() function becomes permanently unusable, preventing all reward distributions and effectively locking user funds.

The vulnerable code sections:

```
function updateDistributeRewardsNumber(uint256 _number) external
    onlyOwner {
     distributeRewardsBatchNumber = _number; // No validation, allows 0
     emit DistributedRewardsNumberUpdated(_number);
}

function distributeRewards(..., UserInfo[] memory infos) external
    onlyOracle {
     // ...
     uint256 length = infos.length;
     if (length == 0) return; // No rewards distributed
     if (length > distributeRewardsBatchNumber) revert
         ExceedDistributeRewardsNumber();
     // If distributeRewardsBatchNumber = 0, any length > 0 will revert
}
```

If distributeRewardsBatchNumber = 0, any attempt to distribute rewards to users (length > 0) will revert, while empty distributions (length = 0) succeed but distribute nothing.

RECOMMENDATIONS:

Add minimum value validation in updateDistributeRewardsNumber():

```
function updateDistributeRewardsNumber(uint256 _number) external
    onlyOwner {
     if (_number == 0) revert InvalidDistributeNumber(); // FIX
     distributeRewardsBatchNumber = _number;
     emit DistributedRewardsNumberUpdated(_number);
}

error InvalidDistributeNumber();
```

3.5 Old Game Contract Funds Not Migrated During setGameContract Update

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

`CrashoutPool.sol - setGameContract()`

DESCRIPTION:

The setGameContract function allows the owner to update the associated Game contract address. However, this function lacks a mechanism to clear or migrate any ETH or ERC-20 tokens that might be stuck in the old Game contract.

While the Game contract shouldn't normally hold significant funds (as user bets go directly to the Pool), unexpected scenarios like direct user transfers or future feature quirks could cause the old Game contract to retain assets. If the project team doesn't manually extract these funds before updating the gameContract address, they will be permanently locked in the old contract, leading to a loss for the project.

The vulnerable code section:

```
function setGameContract(address _gameContract) external onlyOwner {
    if (_gameContract == address(0)) revert InvalidGameContract();
    if (_gameContract == gameContract) revert SameAddress();
    address oldGame = gameContract;
    gameContract = _gameContract;
    emit GameContractUpdate(oldGame, gameContract);
}
```

RECOMMENDATIONS:

It's highly recommended to enforce the clearing of funds from the old Game contract as part of the gameContract update process.

Operational Procedure: Before calling setGameContract, ensure you manually check the old Game contract's ETH and ERC-20 token balances and use its rescueETH() and rescueToken() functions to withdraw all stuck funds to a safe address.

3.6 Immediacy Risk and User Experience Impact of gameContract Updates

SEVERITY: LOW **STATUS:** Acknowledge

PATH:

`CrashoutPool.sol - setGameContract()`

DESCRIPTION:

The setGameContract function allows the contract's owner to immediately and unilaterally change the gameContract address. This instantaneous change introduces an operational risk: if the owner inadvertently sets the gameContract to an incorrect address (e.g., a contract deployed to the wrong network by mistake, or an unverified contract), funds could become inaccessible or locked.

This immediate effect can also have a minor impact on user experience and trust. Users generally expect core contract addresses to be stable. A sudden, unannounced change to the gameContract might cause user confusion or unease. While it doesn't directly lead to fund loss (assuming no malicious intent from the owner), the lack of transparency and preparation time could subtly affect user perception of the protocol's long-term stability.

```
function setGameContract(address _gameContract) external onlyOwner {
    if (_gameContract == address(0)) revert InvalidGameContract();
    if (_gameContract == gameContract) revert SameAddress();
    address oldGame = gameContract;
    gameContract = _gameContract;
    emit GameContractUpdate(oldGame, gameContract);
}
```

RECOMMENDATIONS:

To mitigate this operational risk and enhance user experience, a Timelock mechanism is recommended:

1. **Introduce a Timelock**: Modify the setGameContract function so that the change is not effective immediately. When the owner proposes a new gameContract address, this change would enter a queue and become active only after a predefined delay period (e.g., 24-72 hours).

2. **Event Notification**: Regardless of whether a timelock is implemented, ensure that the Game-ContractUpdate event clearly records both the oldGame and newGame addresses. If a timelock mechanism is adopted, it is also advisable to emit an event when a change is proposed, providing a more detailed on-chain audit trail.

3.7 Missing Zero-Address Check in setOracles

SEVERITY: LOW **STATUS:** Acknowledge

PATH:

`Common.sol - _setOracles()`

DESCRIPTION:

The _setOracles function allows the owner to add or remove oracle addresses. However, when iterating through the _oracles array, there's no check for address(0) for each _oracles[i]. This means the owner can potentially set address(0) as an oracle.

While setting the zero address as an oracle might not immediately cause functional issues (as it's unlikely to be a valid oracle account), allowing the zero address in data storage is generally considered bad practice. It could lead to unexpected behavior or confusion in certain edge cases.

The vulnerable code section:

```
function _setOracles(address[] calldata _oracles, bool allow) internal {
    uint length = _oracles.length;
    for (uint i = 0; i < length; i++) {
        oracles[_oracles[i]] = allow;
    }
}
```

RECOMMENDATIONS:

To improve data integrity, it's recommended to add a non-zero address check for each oracle address within the loop of the _setOracles function:

```
function _setOracles(address[] calldata _oracles, bool allow) internal {
    uint length = _oracles.length;
    for (uint i = 0; i < length; i++) {
        require(_oracles[i] != address(0), ”Invalid oracle address”);
        oracles[_oracles[i]] = allow;
    }
}
```

3.8 Inconsistent Event Parameter Naming

SEVERITY: INFO **STATUS:** Acknowledge

PATH:

`CrashoutPool.sol - GameContractUpdate event`

DESCRIPTION:

The second parameter of the GameContractUpdate event is named newGamem, which appears to be a typographical error. The correct naming should be newGame to accurately reflect that it represents the new game contract address. This inconsistent naming reduces code readability and maintainability, potentially causing confusion for other developers when reading or using this event.

The problematic event definition:

```
event GameContractUpdate(address oldGame, address newGamem);
```

RECOMMENDATIONS:

Change the definition of the GameContractUpdate event to use consistent and accurate parameter naming:

```
event GameContractUpdate(address oldGame, address newGame);
```

Ensure that all event parameter names are clear, accurate, and follow consistent naming conventions throughout the codebase.

3.9 Owner Can Set Equal Min/Max Parameters

SEVERITY: INFO **STATUS:** Acknowledge

PATH:

`CrashoutGame.sol - updateLimits()`

DESCRIPTION:

The updateLimits function allows setting minBetAmount == maxBetAmount and minMultiplier == maxMultiplier. While this doesn't cause immediate security issues, it may not align with the intended game design where ranges are typically expected to provide flexibility for users.

The current validation code:

```
function updateLimits(
    uint256 _minBetAmount,
    uint256 _maxBetAmount,
    uint256 _minMultiplier,
    uint256 _maxMultiplier
) external onlyOwner {
    if (_minBetAmount > _maxBetAmount || _minBetAmount == 0) revert
        InvalidBetLimits();
    if (_minMultiplier > _maxMultiplier || _minMultiplier <
        MULTIPLIER_BASIS) revert InvalidMultiplierLimits();
}
```

RECOMMENDATIONS:

Add validation to prevent equal min/max values if the game design requires meaningful ranges:

```
function updateLimits(
    uint256 _minBetAmount,
    uint256 _maxBetAmount,
    uint256 _minMultiplier,
    uint256 _maxMultiplier
) external onlyOwner {
    if (_minBetAmount >= _maxBetAmount || _minBetAmount == 0) revert
        InvalidBetLimits();
    if (_minMultiplier >= _maxMultiplier || _minMultiplier <
       MULTIPLIER_BASIS) revert InvalidMultiplierLimits();
}
```

4. CONCLUSION

In this audit, we thoroughly analyzed **Crash Game** 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 identified 1 high-severity issue, 4 medium-severity issues, 2 low-severity issues, and 2 informational findings. We recommend addressing all identified issues, particularly the high and medium severity vulnerabilities, before deployment to ensure the security and reliability of the game platform.

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

Contact

Website
www.exvul.com

Email
contact@exvul.com

Twitter
@EXVULSEC

Github
github.com/EXVUL-Sec

ExVul