SCRL

Full Audit Report

DoubleUp Jackpot Security Assessment

DoubleUp Jackpot Security Assessment

FULL AUDIT REPORT

Security Assessment by SCRL on **Sunday, July 7, 2024**

SCRL is deliver a security solution for Web3 projects by expert security researchers.

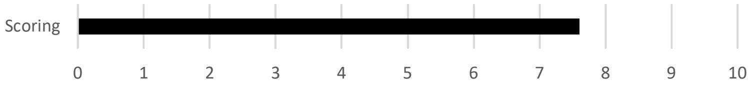
Executive Summary

For this security assessment, SCRL received a request on Sunday, May 16, 2024

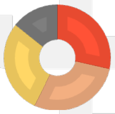
Client	Language	Audit Method	Confidential	Network Chain	Contract
DoubleUp Jackpot	**Solidity**	**Whitebox**	**Public**	**Polygon**	0x497EC8F6cc2445EE9C58e39f54E525F8F7D18392

Report Version	Twitter	Telegram	Website
1.4	https://twitter.com/doubleup_org	https://t.me/doubleup_org	https://doubleup.org/

Scoring:



Vulnerability Summary



13	2	10	1	2	0
Total Findings	Unresolved	Resolved	Mitigate	Acknowledge	Decline

0	Critical		Critical severity is assigned to security vulnerabilities that pose a severe threat to the smart contract and the entire blockchain ecosystem.
2	High	**2 Resolved**	High-severity issues should be addressed quickly to reduce the risk of exploitation and protect users' funds and data.
2	Medium	**1 Mitigate, 1 Resolved**	It's essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract.
2	Low	**2 Resolved**	While low-severity issues can be less urgent, it's still advisable to address them to improve the overall security posture of the smart contract.
0	Very Low		Very Low severity is used for minor security concerns that have minimal impact and are generally of low risk.
1	Informational	**1 Unresolved**	Used to categorize security findings that do not pose a direct security threat to the smart contract or its users. Instead, these findings provide additional information, recommendations
6	Gas-optimization	**1 Unresolved, 5 Resolved**	Suggestions for more efficient algorithms or improvements in gas usage, even if the current code is already secure.

Audit Scope:

File	SHA-1 Hash
src/Jackpot.sol	f1e461820a7345515755a2a15e305fbcbad6cbf9

Audit Version History:

Version	Date	Description
1.0	Thursday, May 23, 2024	Preliminary Report
1.1	Sunday, June 2, 2024	Update with re-assessment on github commit f6431bab29339c9f4d6e014418def81a6634249a
1.2	Wednesday, June 12, 2024	Update with re-assessment even doubleup team deployed contract at 0xF7894a68F236bf0a3Aba1CddC5e32284E48E1609
1.3	Wednesday, June 19, 2024	Update with re-assessment on deployed contract address 0x23C36d7a3363e1399c24280EB49fE0f6535Ab0cD
1.4	Sunday, July 7, 2024	Update with re-assessment on deployed contract address 0x497EC8F6cc2445EE9C58e39f54E525F8F7D18392

Audit information:

Request Date	Audit Date	Re-assessment Date
Thursday, May 16, 2024	Thursday, May 23, 2024	Sunday, July 7, 2024

Smart Contract Audit Summary

SCRL has assessed
the security of this smart contract.

The results of the security
assessment revealed

No Critical Vulnerabilities.

Full Audit Report by SCRL on July 7, 2024

Security Assessment Author

Auditor:	**Mark K.**	[Security Researcher | Redteam]
	Kevin N.	[Security Researcher | Web3 Dev]
	Yusheng T.	[Security Researcher | Incident Response]
Document Approval:	**Ronny C.**	CTO & Head of Security Researcher
	Chinnakit J.	CEO & Founder

Digital Sign

Digitally signed by Chinnakit J.
DN: cn=Chinnakit J., o=SCRL, ou=CEO & Founder, email=contact@scrl.io, c=TH
Date: 2024.07.07 18:21:59 +07'00'

Disclaimer

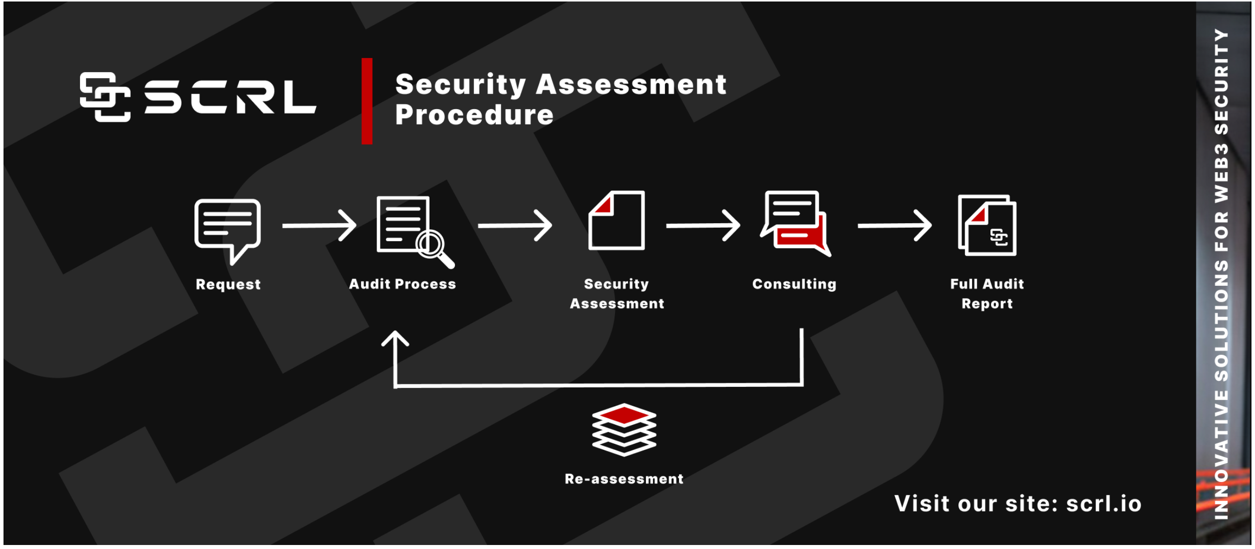
Regarding this security assessment, there are no guarantees about the security of the program instruction received from the client is hereinafter referred to as “**Source code**”.
And **SCRL** hereinafter referred to as “**Service Provider**”, the **Service Provider** will not be held liable for any legal liability arising from errors in the security assessment. The responsibility will be the responsibility of the **Client**, hereinafter referred to as “**Service User**” and the
Service User agrees not to be held liable to the **service provider** in any case. By contract
Service Provider to conduct security assessments with integrity with professional ethics, and transparency to deliver security assessments to users The **Service Provider** has the right to postpone the delivery of the security assessment. If the security assessment is delayed whether caused by any reason and is not responsible for any delayed security assessments.
If **the service provider** finds a vulnerability The **service provider** will notify the **service user** via the Preliminary Report, which will be kept confidential for security. The **service provider** disclaims responsibility in the event of any attacks occurring whether before conducting a security assessment. Or happened later All responsibility shall be sole with the **service user**.

Security Assessment Is Not Financial/Investment Advice Any loss arising from any investment in any project is the responsibility of the investor.

SCRL disclaims any liability incurred. Whether it's Rugpull, Abandonment, Soft Rugpull, Exploit, Exit Scam.

Security Assessment Procedure

1. **Request** The client must submit a formal request and follow the procedure. By submitting the source code and agreeing to the terms of service.
2. **Audit Process** Check for vulnerabilities and vulnerabilities from source code obtained by experts using formal verification methods, including using powerful tools such as Static Analysis, SWC Registry, Dynamic Security Analysis, Automated Security Tools, CWE, Syntax & Parameter Check with AI ,WAS (Warning Avoidance System a python script tools powered by SCRL) and Formal Verification
3. **Security Assessment** Deliver Preliminary Security Assessment to clients to acknowledge the risks and vulnerabilities.
4. **Consulting** Discuss on risks and vulnerabilities encountered by clients to apply to their source code to mitigate risks.
    a. **Re-assessment** Reassess the security when the client implements the source code improvements and if the client is satisfied with the results of the audit. We will proceed to the next step.
5. **Full Audit Report** SCRL provides clients with official security assessment reports informing them of risks and vulnerabilities. Officially and it is assumed that the client has been informed of all the information.



Risk Rating

Risk rating using this commonly defined: $Risk\ rating\ =\ impact\ *\ confidence$

Impact The severity and potential impact of an attacker attack

Confidence Ensuring that attackers expose and use this vulnerability

Confidence Impact [Likelihood]	Low	Medium	High
Low	**Very Low**	**Low**	**Medium**
Medium	**Low**	**Medium**	**High**
High	**Medium**	**High**	**Critical**

Severity is a risk assessment It is calculated from the Impact and Confidence values using the following calculation methods,

$Risk\ rating\ =\ impact\ *\ confidence$

It is categorized into

7 categories severity based

Gas-optimization	Informational	Very Low	Low	Medium	High	Critical

For **Informational** & **Non-class/Optimization/Best-practices will** <u>not be counted</u> as **severity**

Category

Centralization

Centralization Risk is The risk incurred by a sole proprietor, such as the Owner being able to change something without permission

Economics Risk

Economics Risk is Risks that may affect the economic mechanism system, such as the ability to increase Mint token

Logical Issue

Logical Issue is that can cause errors to core processing, such as any prior operations that cause background processes to crash.

Authorization

Authorization is Possible pitfalls from weak coding allows unrelated people to take any action to modify the values.

Mathematical

Mathematical Any erroneous arithmetic operations affect the operation of the system or lead to erroneous values.

Naming Conventions

Naming Conventions naming variables that may affect code understanding or naming inconsistencies

Security Risk

Security Risk of loss or damage if it's no mitigate

Coding Style

Coding Style is Tips coding for efficiency performance

Best Practices

Best Practices is suggestions for improvement

Optimization

Optimization is performance improvement

Gas Optimization

Gas Optimization is increase performance to avoid expensive gas

Dead Code

Dead Code having unused code This may result in wasted resources and gas fees.

Table Of Content

Source Units in Scope

Source Units Analyzed: **1**

Source Units in Scope: **1** (**100%**)

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	src/Jackpot.sol	1		553	538	415	23	254	💰🌀
📝	**Totals**	**1**		**553**	**538**	**415**	**23**	**254**	💰🌀

Legend: [➖]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Visibility, Mutability, Modifier function testing

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
1	0	0	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
15	4

External	Internal	Private	Pure	View
14	11	0	0	5

StateVariables

Total	🌐Public
21	13

Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
^0.8.20		yes		

📤 Transfers ETH	⚡ Low-Level Calls	👥 Delegate Call	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
					yes → NewContract:PriceFeed

♻️ TryCatch	Σ Unchecked

Dependencies / External Imports

Dependency / Import Path	Count
@chainlink/contracts/src/v0.8/vrf/dev/VRFConsumerBaseV2Plus.sol	1
@chainlink/contracts/src/v0.8/vrf/dev/interfaces/IVRFCoordinatorV2Plus.sol	1
@chainlink/contracts/src/v0.8/vrf/dev/libraries/VRFV2PlusClient.sol	1
@openzeppelin/contracts/security/ReentrancyGuard.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	1

Vulnerability Findings

ID	Vulnerability Detail	Severity	Category	Status
REG-01	Potential Reentrancy Attack	High	Logical Issue	**Resolved**
VRF-01	fulfillRandomWords Function must not revert	High	Logical Issue	**Resolved**
CEN-01	Centralization Risk	Medium	Centralization	**Mitigate**
OFL-01	Potential Overflow Issues	Medium	Logical Issue	**Resolved**
SEC-01	Missing Zero Address Validation (missing-zero-check)	Low	Best Practices	**Resolved**
OPN-01	Unsafe ERC20 operation(s)	Low	Best Practices	**Resolved**
SEC-02	Conformity to Solidity naming conventions (naming-convention)	Informational	Naming Conventions	**Acknowledge**
GAS-01	Cache array length outside of loop	Gas-optimization	Gas Optimization	**Resolved**
GAS-02	Use Custom Errors	Gas-optimization	Gas Optimization	**Resolved**
GAS-03	Long revert strings	Gas-optimization	Gas Optimization	**Resolved**
GAS-04	Functions guaranteed to revert when called by normal users can be marked `payable`	Gas-optimization	Gas Optimization	**Resolved**
GAS-05	`++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)	Gas-optimization	Gas Optimization	**Resolved**
GAS-06	Use != 0 instead of > 0 for unsigned integer comparison	Gas-optimization	Gas Optimization	**Acknowledge**

REG-01: Potential Reentrancy Attack

Vulnerability Detail	Severity	Location	Category	Status
Potential Reentrancy Attack	High	Check on finding	Logical Issue	**Resolved**

Finding:

```
Function joinGame() (Jackpot.sol:154-193)
Function cancelGame() (Jackpot.sol:287-329)
Function handleWinner(uint256 _randomWord) (Jackpot.sol:391-470)
```

Description:
The following functions are vulnerable to reentrancy attacks as they involve sending ETH (MATIC) and can be exploited if reentrancy is not properly guarded:

- **Function joinGame() (Jackpot.sol:154-193):**

This function allows a player to join the game by sending ETH (MATIC). Without proper reentrancy protection, an attacker could reenter the function and manipulate the contract state.

- **Function cancelGame() (Jackpot.sol:287-329):**

This function enables the owner to cancel a game and refund the player's bet. If not properly guarded, an attacker could reenter the function and potentially withdraw more than their original bet.

- **Function handleWinner(uint256 _randomWord) (Jackpot.sol:391-470):**

This function handles the distribution of the prize to the winner. Without reentrancy protection, an attacker could manipulate the distribution logic by reentering the function.

Recommendation:
To prevent reentrancy attacks, it is recommended to use the Checks-Effects-Interactions pattern and consider adding a reentrancy guard (nonReentrant) from OpenZeppelin's ReentrancyGuard.

References: SWC-107: Reentrancy: https://swcregistry.io/docs/SWC-107
OpenZeppelin ReentrancyGuard:
https://docs.openzeppelin.com/contracts/4.x/api/security#ReentrancyGuard
Chainlink Document Prevent Revert
https://docs.chain.link/vrf/v2/security/#fulfillrandomwords-must-not-revert

Alliviation:

The doubleup team has already resolved this issue.

VRF-01: fulfillRandomWords Function must not revert

Vulnerability Detail	Severity	Location	Category	Status
fulfillRandomWords Function must not revert	High	Check on finding	Logical Issue	**Resolved**

Finding:

```
Function fulfillRandomWords(uint256 _requestId, uint256[] memory _randomWords)
(Jackpot.sol:376–389)
```

Description:

The function fulfillRandomWords must not revert to ensure the reliability and resilience of the contract. This function is a callback from Chainlink VRF, which is crucial for generating randomness. If this callback fails or reverts due to gas limits or other issues, it can prevent the determination of the game winner and potentially lock the game funds indefinitely.

Recommendation:

We recommend splitting the 'handleWinner' function from 'fulfillRandomWords' to enhance the contract's resilience and reliability. The 'fulfillRandomWords' function, being a callback from Chainlink VRF, is critical for generating randomness. If this callback fails or reverts due to gas limits or other issues, it can prevent the determination of the game winner and potentially lock the game funds indefinitely

References: Chainlink VRF: https://docs.chain.link/vrf/v2/introduction
Chainlink Document Prevent Revert
https://docs.chain.link/vrf/v2/security/#fulfillrandomwords-must-not-revert

Alleviation:

The doubleup team has already resolved this issue.

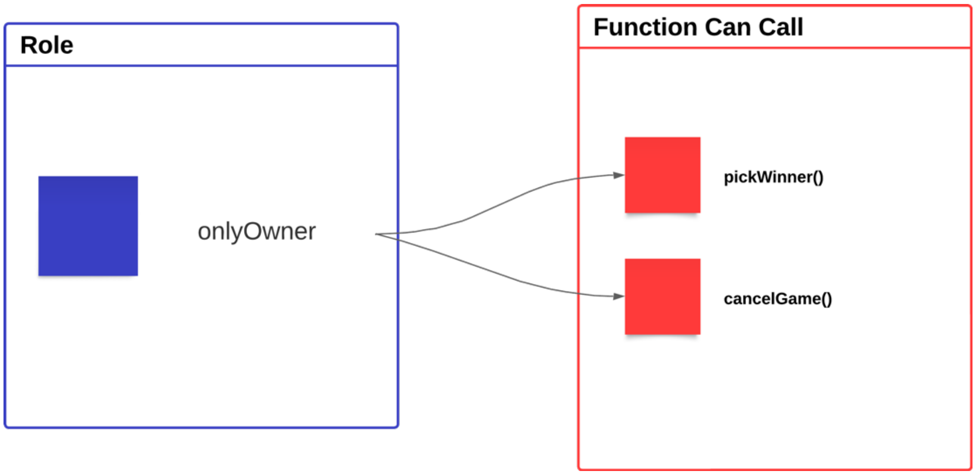
CEN-01: Centralization Risk

Vulnerability Detail	Severity	Location	Category	Status
Centralization Risk	Medium	Check on finding	Centralization	**Mitigate**

Finding:

```
File: Jackpot.sol

267:     function pickWinner() external onlyOwner {

287:     function cancelGame() external onlyOwner {

```
```

Explain Function Capability:

The contract provides several functions:

1. **pickWinner()**
   - This function is responsible for selecting a winner for the game.
   - It checks that the game exists, has not ended, has at least 2 players, and that the pick deadline has passed.
   - Upon meeting these conditions, it sets isPickWinner to true and requests a random number from Chainlink VRF to determine the winner.

   Impact:
   - The function is marked with onlyOwner, meaning only the contract owner can call it.
   - This centralizes the control over when and how a winner is picked, potentially allowing for manipulation or favoritism.
2. **cancelGame()**
   - This function allows the owner to cancel a game under certain conditions.
   - It checks that the game exists, has not been canceled before, no winner has been picked, only one player has joined, and the cancel deadline has passed.
   - If these conditions are met, it cancels the game, refunds the player's bet, and emits a CancelGame event.

   Impact:
   - The function is marked with onlyOwner, meaning only the contract owner can call it.
   - This centralizes the control over game cancellations, allowing the owner to decide unilaterally when a game should be canceled.
```

Centralization Risk



Recommendation:

In terms of timeframes, there are three categories: short-term, long-term, and permanent.

For short-term solutions, a combination of timelock and multi-signature (2/3 or 3/5) can be used to mitigate risk by delaying sensitive operations and avoiding a single point of failure in key management. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; assigning privileged roles to multi-signature wallets to prevent private key compromise; and sharing the timelock contract and multi-signer addresses with the public via a medium/blog link.

For long-term solutions, a combination of timelock and DAO can be used to apply decentralization and transparency to the system. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; introducing a DAO/governance/voting module to increase transparency and user involvement; and sharing the timelock contract, multi-signer addresses, and DAO information with the public via a medium/blog link.

Finally, permanent solutions should be implemented to ensure the ongoing security and protection of the system.

Alleviation:

The doubleup team will using multi-signature it’s will mitigated this centralization risk, but still remember **doubleup team still can call this centralized function.**

OFL-01: Potential Overflow Issues

Vulnerability Detail	Severity	Location	Category	Status
Potential Overflow Issues	Medium	Check on finding	Logical Issue	**Resolved**

Finding:

```
Function handleWinner(uint256 _randomWord) (Jackpot.sol:391–470)
```

Description:
The function handleWinner involves arithmetic operations that should be checked for potential overflows. Although Solidity 0.8.x has built-in overflow checks, it is essential to ensure these calculations are logically sound and that developers are aware of the built-in protections.

Recommendation:
Even though Solidity 0.8.x includes built-in overflow checks, it is good practice to explicitly state that these protections are relied upon and to ensure that all calculations are reviewed for logical correctness.

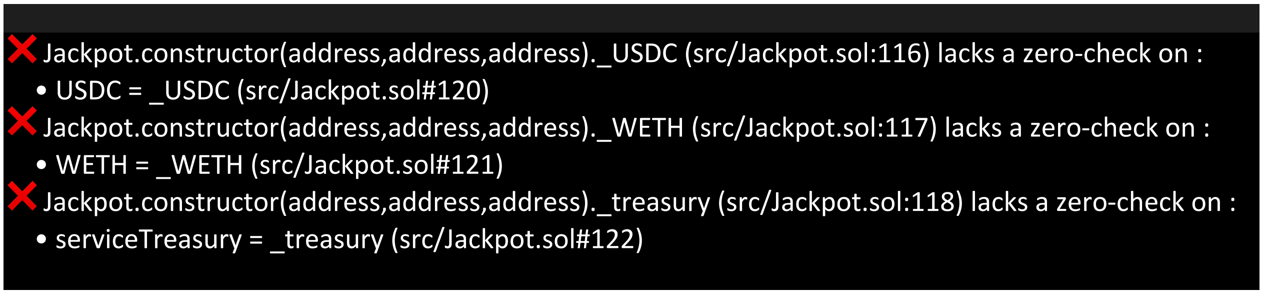
References: Solidity 0.8.0 Release Notes
https://docs.soliditylang.org/en/v0.8.0/080-breaking-changes.html
Solidity Documentation - Arithmetic Operations
https://docs.soliditylang.org/en/v0.8.0/control-structures.html#checked-or-unchecked-arithmetic

Alleviation:
The doubleup team has already resolved this issue.

SEC-01: Missing Zero Address Validation (missing-zero-check)

Vulnerability Detail	Severity	Location	Category	Status
Missing Zero Address Validation (missing-zero-check)	Low	Check on finding	Best Practices	**Resolved**

Finding:



```
❌ Jackpot.constructor(address,address,address)._USDC (src/Jackpot.sol:116) lacks a zero-check on :
  • USDC = _USDC (src/Jackpot.sol#120)
❌ Jackpot.constructor(address,address,address)._WETH (src/Jackpot.sol:117) lacks a zero-check on :
  • WETH = _WETH (src/Jackpot.sol#121)
❌ Jackpot.constructor(address,address,address)._treasury (src/Jackpot.sol:118) lacks a zero-check on :
  • serviceTreasury = _treasury (src/Jackpot.sol#122)
```

Recommendation:

Check that the address is not zero.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Alleviation:

The doubleup team has already resolved this issue.

OPN-01: Unsafe ERC20 operation(s)

Vulnerability Detail	Severity	Location	Category	Status
Unsafe ERC20 operation(s)	Low	Check on finding	Best Practices	**Resolved**

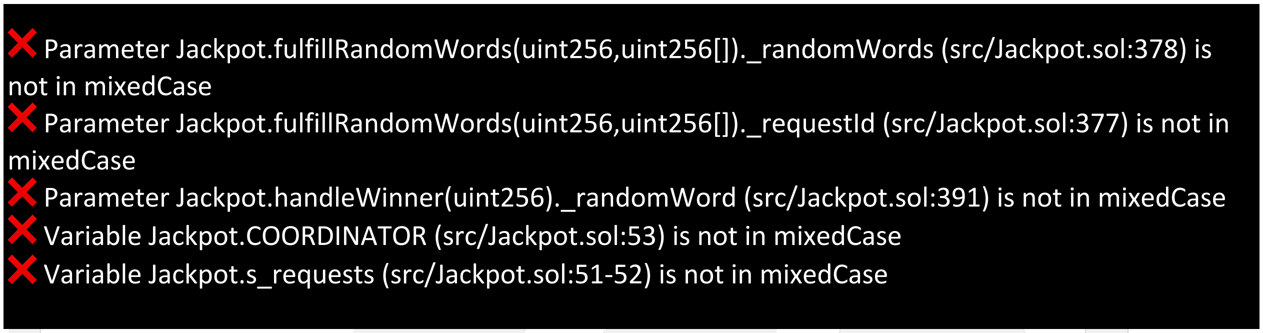
Finding:

```
File: Jackpot.sol

216:            IERC20(gameToken).transferFrom(

246:            IERC20(gameToken).transferFrom(

314:            IERC20(WETH).transfer(

322:            IERC20(USDC).transfer(

435:            IERC20(WETH).transfer(

441:            IERC20(WETH).transfer(

450:            IERC20(USDC).transfer(

457:            IERC20(USDC).transfer(

```
```

Recommendation:

To mitigate this issue, it is recommended to use OpenZeppelin's SafeERC20 library, which wraps these operations and automatically handles the return value, reverting the transaction if the transfer fails. This approach aligns with the best practices for safe ERC20 interactions as outlined in the OpenZeppelin documentation.

References: CWE-252: Unchecked Return Value: https://cwe.mitre.org/data/definitions/252.html
SWC-104: Unchecked Return Value from Low-Level Calls: https://swcregistry.io/docs/SWC-104
OpenZeppelin SafeERC20 Library:
https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#SafeERC20

Alleviation:

The doubleup team has already resolved this issue.
```

SEC-02: Conformity to Solidity naming conventions (naming-convention)

Vulnerability Detail	Severity	Location	Category	Status
Conformity to Solidity naming conventions (naming-convention)	Informational	Check on finding	Naming Conventions	**Acknowledge**

Finding:



```
❌ Parameter Jackpot.fulfillRandomWords(uint256,uint256[])._randomWords (src/Jackpot.sol:378) is not in mixedCase
❌ Parameter Jackpot.fulfillRandomWords(uint256,uint256[])._requestId (src/Jackpot.sol:377) is not in mixedCase
❌ Parameter Jackpot.handleWinner(uint256)._randomWord (src/Jackpot.sol:391) is not in mixedCase
❌ Variable Jackpot.COORDINATOR (src/Jackpot.sol:53) is not in mixedCase
❌ Variable Jackpot.s_requests (src/Jackpot.sol:51-52) is not in mixedCase
```

Recommendation:

Follow the Solidity [naming convention](https://solidity.readthedocs.io/en/v0.4.25/style-guide.html#naming-conventions).

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Alleviation:

-

GAS-01: Cache array length outside of loop

Vulnerability Detail	Severity	Location	Category	Status
Cache array length outside of loop	Gas-optimization	Check on finding	Gas Optimization	**Resolved**

Finding:

```
File: Jackpot.sol

148:          for (uint8 i = 0; i < gameData.players.length; i++) {

398:          for (uint8 i = 0; i < gameData.players.length; i++) {

408:          for (uint8 i = 0; i < gameData.players.length; i++) {

```
```

Recommendation:

When iterating over an array in Solidity, reading the array length in each iteration can lead to unnecessary gas costs. Caching the array length outside the loop can optimize gas usage. This applies to both storage arrays (which incur sload operations) and memory arrays (which incur mload operations).

Alleviation:

The doubleup team has already resolved this issue.
```

GAS-02: Use Custom Errors

Vulnerability Detail	Severity	Location	Category	Status
Use Custom Errors	Gas-optimization	Check on finding	Gas Optimization	**Resolved**

Finding:

```
File: Jackpot.sol

133:                require(gameData.endTime > 0, "You can't create new game now");

155:            require(gameCounter > 0, "No room");

156:            require(msg.value > 0, "Join amount is 0.");

159:            require(!gameData.isPickWinner, "This game has already been finished.");

201:            require(gameCounter > 0, "No room");

202:            require(amount > 0, "Join amount is 0.");

205:            require(!gameData.isPickWinner, "This game has already been finished.");

268:            require(gameCounter > 0, "No room");

288:            require(gameCounter > 0, "No room");

291:            require(!gameData.cancelFlag, "This game has already been cancelled.");

296:            require(gameData.players.length == 1, "You can't cancel game.");

309:                require(send, "Transfer failed.");

380:            require(s_requests[_requestId].exists, "request not found");

426:                require(send1, "Transfer failed.");

430:                require(send2, "Transfer failed.");

```
```
```

Recommendation:

[Source](https://blog.soliditylang.org/2021/04/21/custom-errors/)
Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

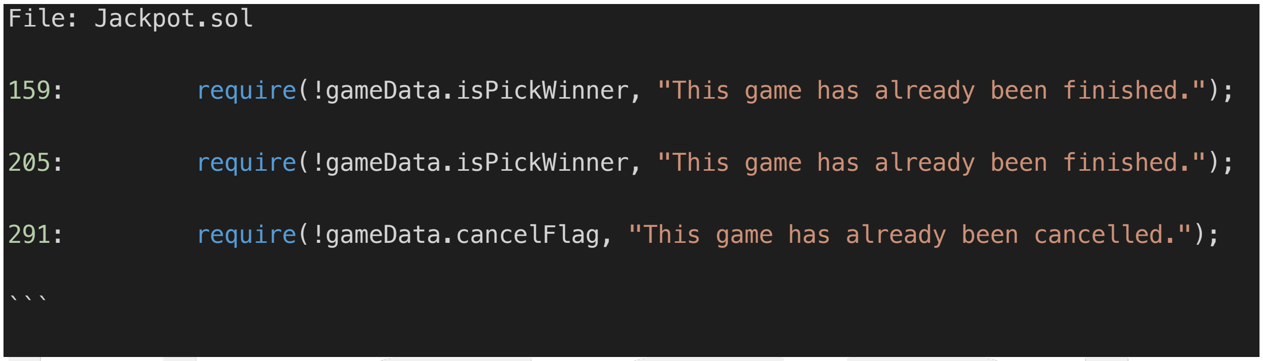
Alleviation:

The doubleup team has already resolved this issue.

GAS-03: Long revert strings

Vulnerability Detail	Severity	Location	Category	Status
Long revert strings	Gas-optimization	Check on finding	Gas Optimization	**Resolved**

Finding:



```
File: Jackpot.sol

159:          require(!gameData.isPickWinner, "This game has already been finished.");

205:          require(!gameData.isPickWinner, "This game has already been finished.");

291:          require(!gameData.cancelFlag, "This game has already been cancelled.");

```
```

Recommendation:

Long revert strings in the require statements consume more gas. It's more gas-efficient to use short revert strings or error codes. In addition, using custom errors can further optimize gas usage while providing clear and meaningful error messages.

Alleviation:

The doubleup team has already resolved this issue.
```

GAS-04: Functions guaranteed to revert when called by normal users can be marked `payable`

Vulnerability Detail	Severity	Location	Category	Status
Functions guaranteed to revert when called by normal users can be marked `payable`	Gas-optimization	Check on finding	Gas Optimization	**Resolved**

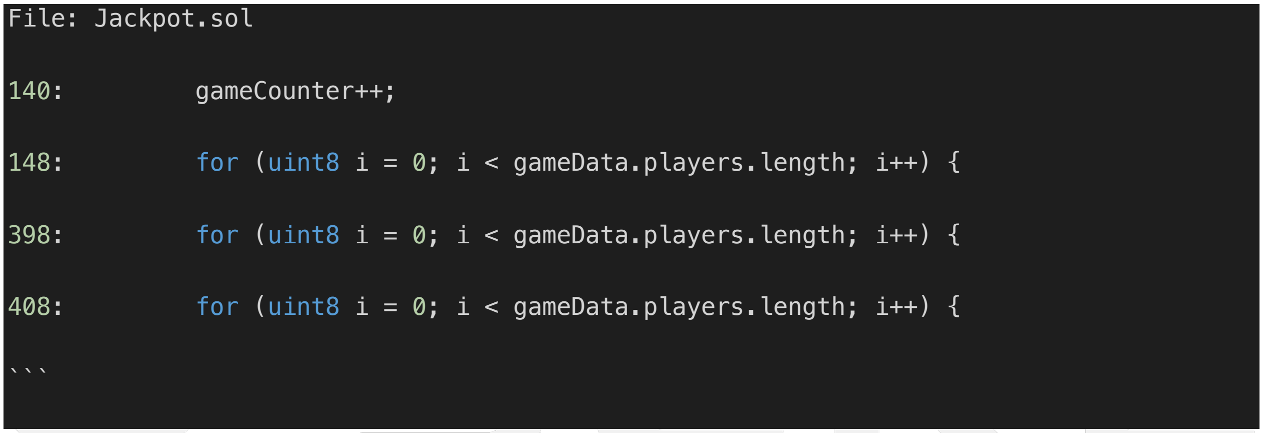
Finding:

```
File: Jackpot.sol

267:      function pickWinner() external onlyOwner {

287:      function cancelGame() external onlyOwner {

```
```

Recommendation:

Functions that are restricted to certain roles (e.g., onlyOwner) and will revert if called by normal users can be marked as payable. This reduces gas costs for legitimate callers by eliminating the need for the compiler to include checks for whether a payment was provided.

Mark the pickWinner and cancelGame functions as payable. This optimization will reduce the gas cost for the owner when these functions are called.

Alleviation:

The doubleup team has already resolved this issue.
```

GAS-05: `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)

Vulnerability Detail	Severity	Location	Category	Status
`++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)	Gas-optimization	Check on finding	Gas Optimization	**Resolved**

Finding:

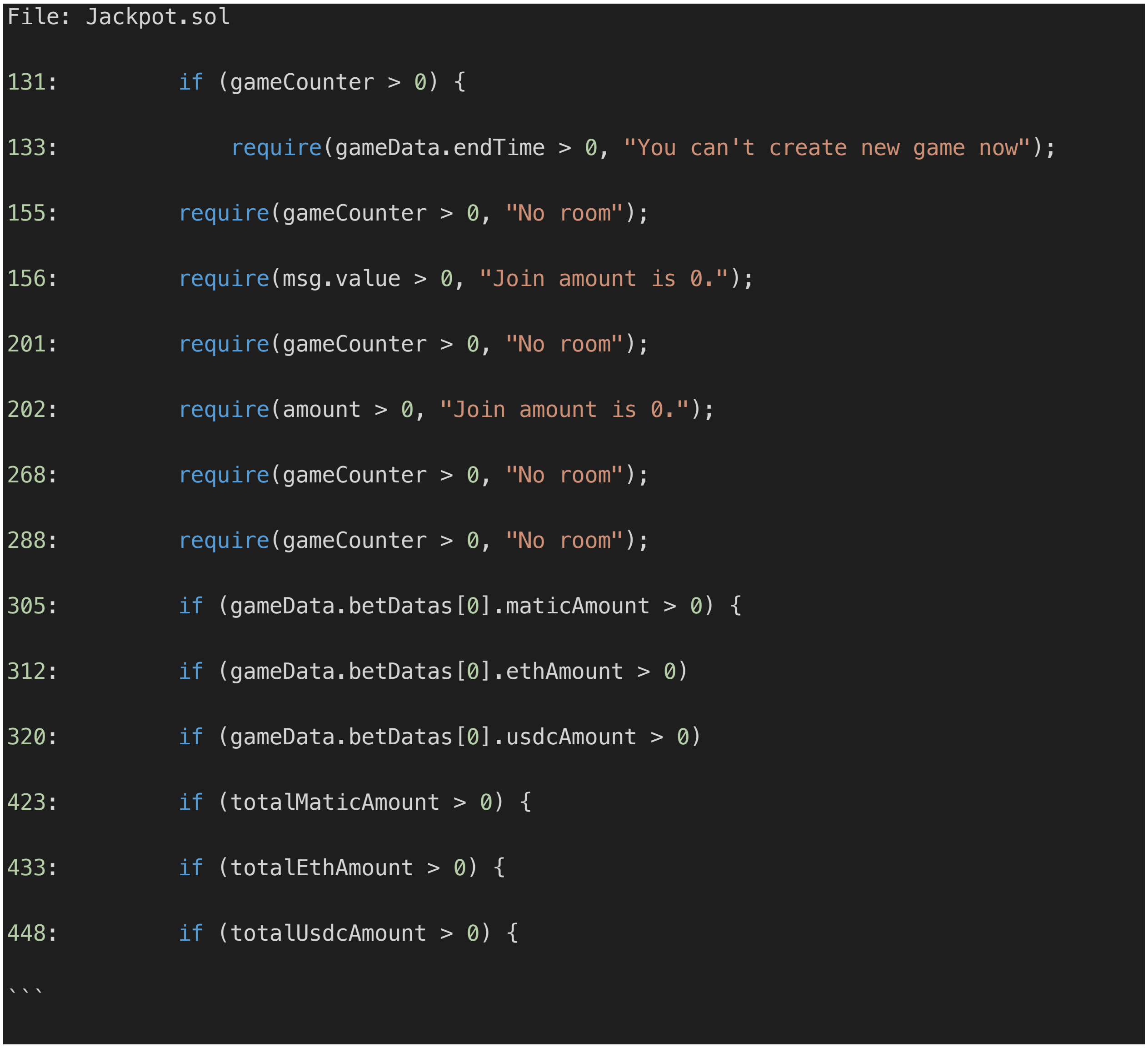


```
File: Jackpot.sol

140:         gameCounter++;

148:         for (uint8 i = 0; i < gameData.players.length; i++) {

398:         for (uint8 i = 0; i < gameData.players.length; i++) {

408:         for (uint8 i = 0; i < gameData.players.length; i++) {

```
```

Recommendation:

Using ++i (pre-increment) instead of i++ (post-increment) can save gas, especially in for loops. The same principle applies to decrement operations (--i vs i--).

Change post-increment i++ to pre-increment ++i to optimize gas usage.

Alleviation:

The doubleup team has already resolved this issue.
```

GAS-06: Use != 0 instead of > 0 for unsigned integer comparison

Vulnerability Detail	Severity	Location	Category	Status
Use != 0 instead of > 0 for unsigned integer comparison	Gas-optimization	Check on finding	Gas Optimization	**Acknowledge**

Finding:



```
File: Jackpot.sol

131:          if (gameCounter > 0) {

133:              require(gameData.endTime > 0, "You can't create new game now");

155:          require(gameCounter > 0, "No room");

156:          require(msg.value > 0, "Join amount is 0.");

201:          require(gameCounter > 0, "No room");

202:          require(amount > 0, "Join amount is 0.");

268:          require(gameCounter > 0, "No room");

288:          require(gameCounter > 0, "No room");

305:          if (gameData.betDatas[0].maticAmount > 0) {

312:          if (gameData.betDatas[0].ethAmount > 0)

320:          if (gameData.betDatas[0].usdcAmount > 0)

423:          if (totalMaticAmount > 0) {

433:          if (totalEthAmount > 0) {

448:          if (totalUsdcAmount > 0) {

```
```
```

Recommendation:

Using != 0 for checking if an unsigned integer is greater than zero can save gas compared to using > 0.

Alleviation:

-

SWC Findings

ID	Title	Scanning	Result
SWC-100	Function Default Visibility	Complete	No risk
SWC-101	Integer Overflow and Underflow	Complete	No risk
SWC-102	Outdated Compiler Version	Complete	No risk
SWC-103	Floating Pragma	Complete	No risk
SWC-104	Unchecked Call Return Value	Complete	No risk
SWC-105	Unprotected Ether Withdrawal	Complete	No risk
SWC-106	Unprotected SELFDESTRUCT Instruction	Complete	No risk
SWC-107	Reentrancy	Complete	No risk
SWC-108	State Variable Default Visibility	Complete	No risk
SWC-109	Uninitialized Storage Pointer	Complete	No risk
SWC-110	Assert Violation	Complete	No risk
SWC-111	Use of Deprecated Solidity Functions	Complete	No risk
SWC-112	Delegatecall to Untrusted Callee	Complete	No risk
SWC-113	DoS with Failed Call	Complete	No risk
SWC-114	Transaction Order Dependence	Complete	No risk
SWC-115	Authorization through tx.origin	Complete	No risk

SWC-116	Block values as a proxy for time	Complete	No risk
SWC-117	Signature Malleability	Complete	No risk
SWC-118	Incorrect Constructor Name	Complete	No risk
SWC-119	Shadowing State Variables	Complete	No risk
SWC-120	Weak Sources of Randomness from Chain Attributes	Complete	No risk
SWC-121	Missing Protection against Signature Replay Attacks	Complete	No risk
SWC-122	Lack of Proper Signature Verification	Complete	No risk
SWC-123	Requirement Violation	Complete	No risk
SWC-124	Write to Arbitrary Storage Location	Complete	No risk
SWC-125	Incorrect Inheritance Order	Complete	No risk
SWC-126	Insufficient Gas Griefing	Complete	No risk
SWC-127	Arbitrary Jump with Function Type Variable	Complete	No risk
SWC-128	DoS With Block Gas Limit	Complete	No risk
SWC-129	Typographical Error	Complete	No risk
SWC-130	Right-To-Left-Override control character (U+202E)	Complete	No risk
SWC-131	Presence of unused variables	Complete	No risk
SWC-132	Unexpected Ether balance	Complete	No risk

SWC-133	Hash Collisions With Multiple Variable Length Arguments	Complete	No risk
SWC-134	Message call with hardcoded gas amount	Complete	No risk
SWC-135	Code With No Effects	Complete	No risk
SWC-136	Unencrypted Private Data On-Chain	Complete	No risk

Contracts Description Table

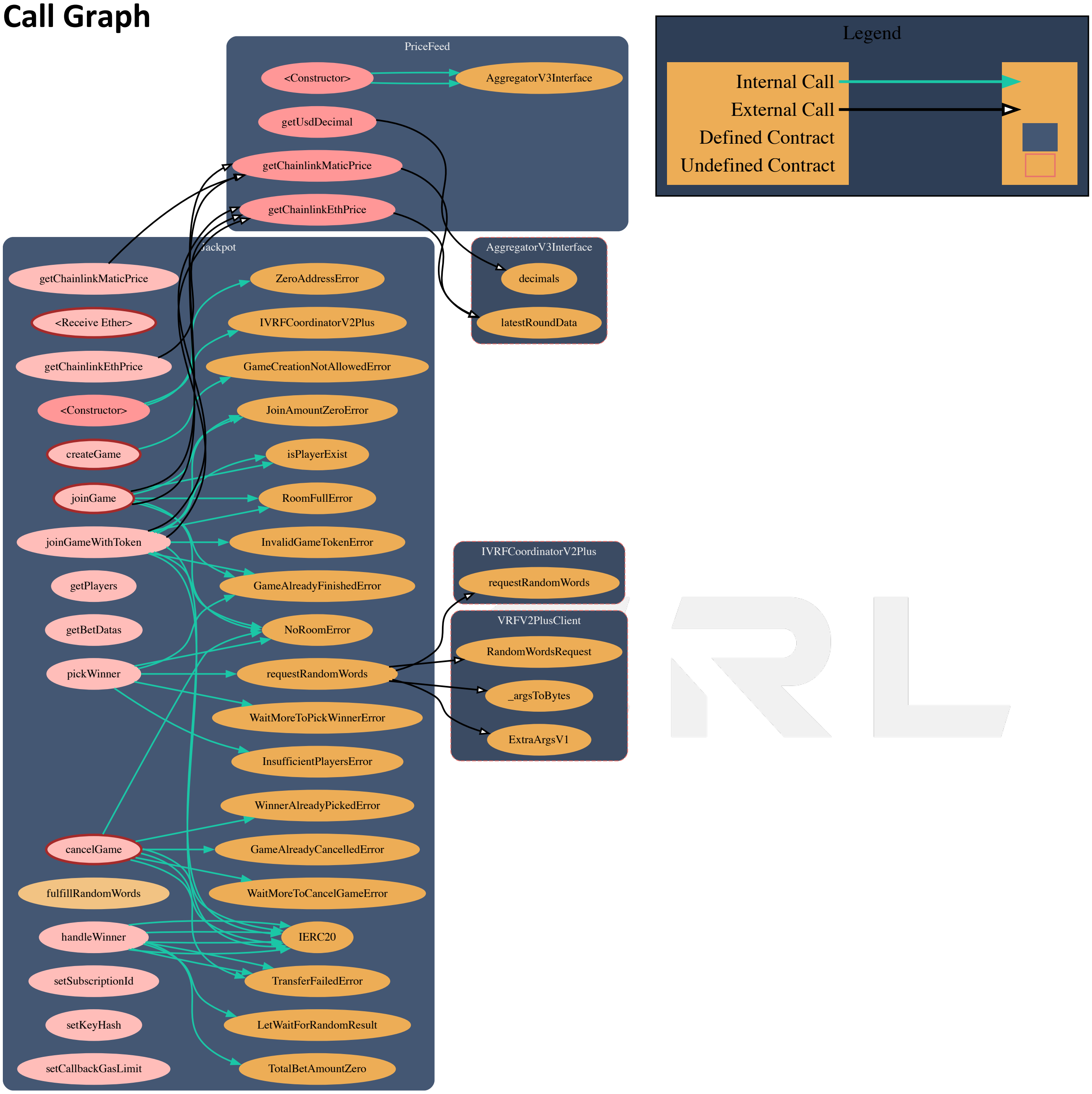
Contract	**Type**	**Bases**		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
Jackpot	Implementation	VRFConsumerBaseV2Plus, ReentrancyGuard		
└		Public ❗	🛑	VRFConsumerBaseV2Plus
└		External ❗	💵	NO ❗
└	createGame	External ❗	💵	NO ❗
└	isPlayerExist	Public ❗		NO ❗
└	joinGame	External ❗	💵	nonReentrant
└	joinGameWithToken	External ❗	🛑	nonReentrant
└	pickWinner	External ❗	🛑	onlyOwner nonReentrant
└	cancelGame	External ❗	💵	onlyOwner nonReentrant
└	getPlayers	External ❗		NO ❗
└	getBetDatas	External ❗		NO ❗
└	getChainlinkMaticPrice	External ❗		NO ❗
└	getChainlinkEthPrice	External ❗		NO ❗
└	requestRandomWords	Internal 🔒	🛑	
└	fulfillRandomWords	Internal 🔒	🛑	

Contract	Type	Bases		
└	handleWinner	External ❗	🛑	onlyOwner nonReentrant
└	setSubscriptionId	External ❗	🛑	onlyOwner
└	setKeyHash	External ❗	🛑	onlyOwner
└	setCallbackGasLimit	External ❗	🛑	onlyOwner

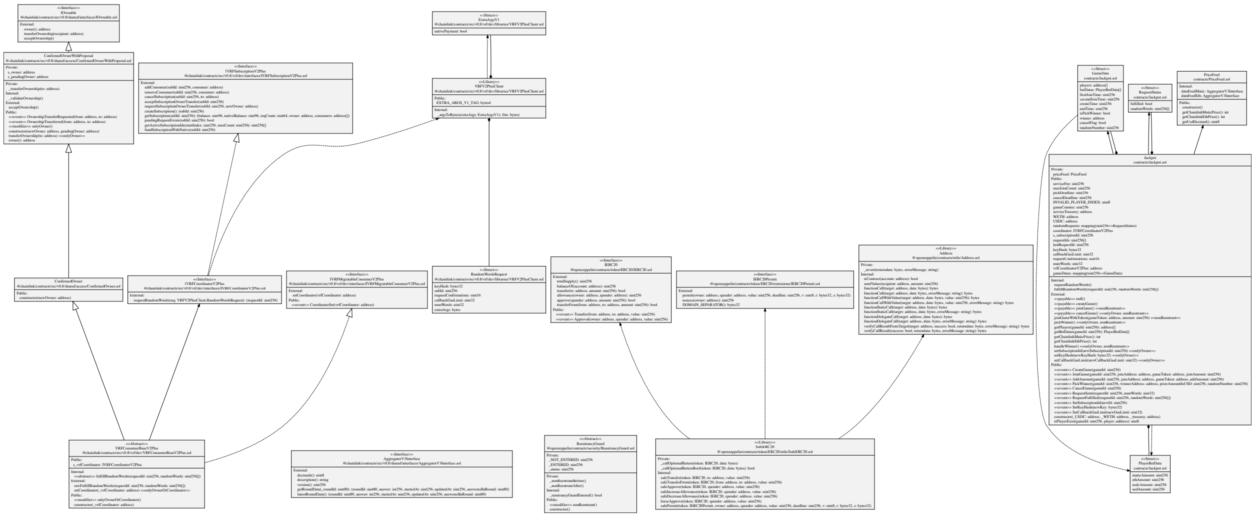
Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Call Graph

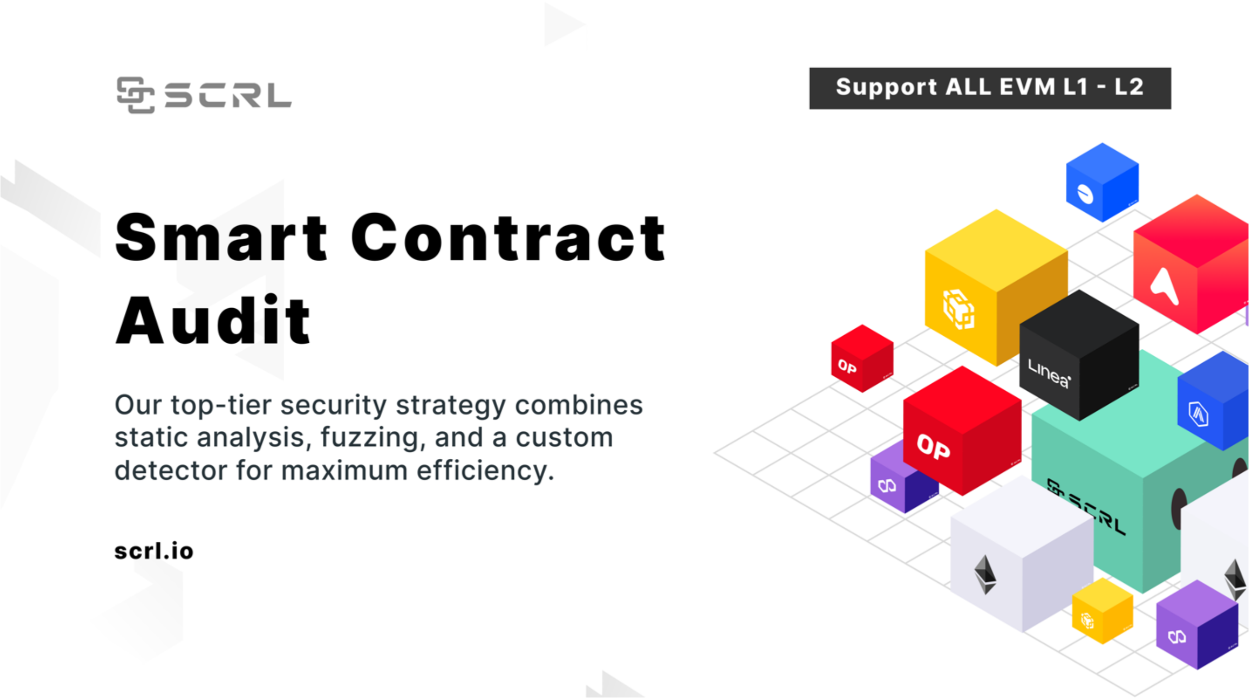

Legend
Internal Call
External Call
Defined Contract
Undefined Contract
PriceFeed
<Constructor>
AggregatorV3Interface
getUsdDecimal
getChainlinkMaticPrice
getChainlinkEthPrice
AggregatorV3Interface
decimals
latestRoundData
Jackpot
getChainlinkMaticPrice
<Receive Ether>
getChainlinkEthPrice
<Constructor>
createGame
joinGame
joinGameWithToken
getPlayers
getBetDatas
pickWinner
cancelGame
fulfillRandomWords
handleWinner
setSubscriptionId
setKeyHash
setCallbackGasLimit
ZeroAddressError
IVRFCoordinatorV2Plus
GameCreationNotAllowedError
JoinAmountZeroError
isPlayerExist
RoomFullError
InvalidGameTokenError
GameAlreadyFinishedError
NoRoomError
requestRandomWords
WaitMoreToPickWinnerError
InsufficientPlayersError
WinnerAlreadyPickedError
GameAlreadyCancelledError
WaitMoreToCancelGameError
IERC20
TransferFailedError
LetWaitForRandomResult
TotalBetAmountZero
IVRFCoordinatorV2Plus
requestRandomWords
VRFV2PlusClient
RandomWordsRequest
_argsToBytes
ExtraArgsV1

UML Class Diagram


<<Interface>>
IOwnable
@chainlink/contracts/src/v0.8/shared/interfaces/IOwnable.sol
External:
owner(): address
transferOwnership(recipient: address)
acceptOwnership()
ConfirmedOwnerWithProposal
@chainlink/contracts/src/v0.8/shared/access/ConfirmedOwnerWithProposal.sol
Private:
s_owner: address
s_pendingOwner: address
Private:
_transferOwnership(to: address)
Internal:
_validateOwnership()
External:
acceptOwnership()
Public:
<<event>> OwnershipTransferRequested(from: address, to: address)
<<event>> OwnershipTransferred(from: address, to: address)
<<modifier>> onlyOwner()
constructor(newOwner: address, pendingOwner: address)
transferOwnership(to: address) <<onlyOwner>>
owner(): address
<<Interface>>
IVRFSubscriptionV2Plus
@chainlink/contracts/src/v0.8/vrf/dev/interfaces/IVRFSubscriptionV2Plus.sol
External:
addConsumer(subId: uint256, consumer: address)
removeConsumer(subId: uint256, consumer: address)
cancelSubscription(subId: uint256, to: address)
acceptSubscriptionOwnerTransfer(subId: uint256)
requestSubscriptionOwnerTransfer(subId: uint256, newOwner: address)
createSubscription(): (subId: uint256)
getSubscription(subId: uint256): (balance: uint96, nativeBalance: uint96, reqCount: uint64, owner: address, consumers: address[])
pendingRequestExists(subId: uint256): bool
getActiveSubscriptionIds(startIndex: uint256, maxCount: uint256): uint256[]
fundSubscriptionWithNative(subId: uint256)
<<Struct>>
ExtraArgsV1
@chainlink/contracts/src/v0.8/vrf/dev/libraries/VRFV2PlusClient.sol
nativePayment: bool
<<Library>>
VRFV2PlusClient
@chainlink/contracts/src/v0.8/vrf/dev/libraries/VRFV2PlusClient.sol
Public:
EXTRA_ARGS_V1_TAG: bytes4
Internal:
_argsToBytes(extraArgs: ExtraArgsV1): (bts: bytes)
ConfirmedOwner
@chainlink/contracts/src/v0.8/shared/access/ConfirmedOwner.sol
Public:
constructor(newOwner: address)
<<Interface>>
IVRFCoordinatorV2Plus
@chainlink/contracts/src/v0.8/vrf/dev/interfaces/IVRFCoordinatorV2Plus.sol
External:
requestRandomWords(req: VRFV2PlusClient.RandomWordsRequest): (requestId: uint256)
<<Interface>>
IVRFMigratableConsumerV2Plus
@chainlink/contracts/src/v0.8/vrf/dev/interfaces/IVRFMigratableConsumerV2Plus.sol
External:
setCoordinator(vrfCoordinator: address)
Public:
<<event>> CoordinatorSet(vrfCoordinator: address)
<<Struct>>
RandomWordsRequest
@chainlink/contracts/src/v0.8/vrf/dev/libraries/VRFV2PlusClient.sol
keyHash: bytes32
subId: uint256
requestConfirmations: uint16
callbackGasLimit: uint32
numWords: uint32
extraArgs: bytes
<<Interface>>
IERC20
@openzeppelin/contracts/token/ERC20/IERC20.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(from: address, to: address, amount: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
<<Interface>>
IERC20Permit
@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol
External:
permit(owner: address, spender: address, value: uint256, deadline: uint256, v: uint8, r: bytes32, s: bytes32)
nonces(owner: address): uint256
DOMAIN_SEPARATOR(): bytes32
<<Library>>
Address
@openzeppelin/contracts/utils/Address.sol
Private:
_revert(returndata: bytes, errorMessage: string)
Internal:
isContract(account: address): bool
sendValue(recipient: address, amount: uint256)
functionCall(target: address, data: bytes): bytes
functionCall(target: address, data: bytes, errorMessage: string): bytes
functionCallWithValue(target: address, data: bytes, value: uint256): bytes
functionCallWithValue(target: address, data: bytes, value: uint256, errorMessage: string): bytes
functionStaticCall(target: address, data: bytes): bytes
functionStaticCall(target: address, data: bytes, errorMessage: string): bytes
functionDelegateCall(target: address, data: bytes): bytes
functionDelegateCall(target: address, data: bytes, errorMessage: string): bytes
verifyCallResultFromTarget(target: address, success: bool, returndata: bytes, errorMessage: string): bytes
verifyCallResult(success: bool, returndata: bytes, errorMessage: string): bytes
<<Struct>>
GameData
contracts/Jackpot.sol
players: address[]
betDatas: PlayerBetData[]
firstJoinTime: uint256
secondJoinTime: uint256
createTime: uint256
endTime: uint256
isPickWinner: bool
winner: address
cancelFlag: bool
randomNumber: uint256
<<Struct>>
RequestStatus
contracts/Jackpot.sol
fulfilled: bool
randomWords: uint256[]
PriceFeed
contracts/PriceFeed.sol
Internal:
dataFeedMatic: AggregatorV3Interface
dataFeedEth: AggregatorV3Interface
Public:
constructor()
getChainlinkMaticPrice(): int
getChainlinkEthPrice(): int
getUsdDecimal(): uint8
Jackpot
contracts/Jackpot.sol
Private:
priceFeed: PriceFeed
Public:
serviceFee: uint256
maxJoinCount: uint256
pickDeadline: uint256
cancelDeadline: uint256
INVALID_PLAYER_INDEX: uint8
gameCounter: uint256
serviceTreasury: address
WETH: address
USDC: address
randomRequests: mapping(uint256=>RequestStatus)
coordinator: IVRFCoordinatorV2Plus
s_subscriptionId: uint256
requestIds: uint256[]
lastRequestId: uint256
keyHash: bytes32
callbackGasLimit: uint32
requestConfirmations: uint16
numWords: uint32
vrfCoordinatorV2Plus: address
gameDatas: mapping(uint256=>GameData)
Internal:
requestRandomWords()
fulfillRandomWords(requestId: uint256, randomWords: uint256[])
External:
<<payable>> null()
<<payable>> createGame()
<<payable>> joinGame() <<nonReentrant>>
<<payable>> cancelGame() <<onlyOwner, nonReentrant>>
joinGameWithToken(gameToken: address, amount: uint256) <<nonReentrant>>
pickWinner() <<onlyOwner, nonReentrant>>
getPlayers(gameId: uint256): address[]
getBetDatas(gameId: uint256): PlayerBetData[]
getChainlinkMaticPrice(): int
getChainlinkEthPrice(): int
handleWinner() <<onlyOwner, nonReentrant>>
setSubscriptionId(newSubscriptionId: uint256) <<onlyOwner>>
setKeyHash(newKeyHash: bytes32) <<onlyOwner>>
setCallbackGasLimit(newCallbackGasLimit: uint32) <<onlyOwner>>
Public:
<<event>> CreateGame(gameId: uint256)
<<event>> JoinGame(gameId: uint256, joinAddress: address, gameToken: address, joinAmount: uint256)
<<event>> AddAmount(gameId: uint256, joinAddress: address, gameToken: address, addAmount: uint256)
<<event>> PickWinner(gameId: uint256, winnerAddress: address, prizeAmountInUSD: uint256, randomNumber: uint256)
<<event>> CancelGame(gameId: uint256)
<<event>> RequestSent(requestId: uint256, numWords: uint32)
<<event>> RequestFulfilled(requestId: uint256, randomWords: uint256[])
<<event>> SetSubscriptionId(newId: uint256)
<<event>> SetKeyHash(newKey: bytes32)
<<event>> SetCallbackGasLimit(newGasLimit: uint32)
constructor(_USDC: address, _WETH: address, _treasury: address)
isPlayerExist(gameId: uint256, player: address): uint8
<<Abstract>>
VRFConsumerBaseV2Plus
@chainlink/contracts/src/v0.8/vrf/dev/VRFConsumerBaseV2Plus.sol
Public:
s_vrfCoordinator: IVRFCoordinatorV2Plus
Internal:
<<abstract>> fulfillRandomWords(requestId: uint256, randomWords: uint256[])
External:
rawFulfillRandomWords(requestId: uint256, randomWords: uint256[])
setCoordinator(_vrfCoordinator: address) <<onlyOwnerOrCoordinator>>
Public:
<<modifier>> onlyOwnerOrCoordinator()
constructor(_vrfCoordinator: address)
<<Interface>>
AggregatorV3Interface
@chainlink/contracts/src/v0.8/shared/interfaces/AggregatorV3Interface.sol
External:
decimals(): uint8
description(): string
version(): uint256
getRoundData(_roundId: uint80): (roundId: uint80, answer: int256, startedAt: uint256, updatedAt: uint256, answeredInRound: uint80)
latestRoundData(): (roundId: uint80, answer: int256, startedAt: uint256, updatedAt: uint256, answeredInRound: uint80)
<<Abstract>>
ReentrancyGuard
@openzeppelin/contracts/security/ReentrancyGuard.sol
Private:
_NOT_ENTERED: uint256
_ENTERED: uint256
_status: uint256
Private:
_nonReentrantBefore()
_nonReentrantAfter()
Internal:
_reentrancyGuardEntered(): bool
Public:
<<modifier>> nonReentrant()
constructor()
<<Library>>
SafeERC20
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol
Private:
_callOptionalReturn(token: IERC20, data: bytes)
_callOptionalReturnBool(token: IERC20, data: bytes): bool
Internal:
safeTransfer(token: IERC20, to: address, value: uint256)
safeTransferFrom(token: IERC20, from: address, to: address, value: uint256)
safeApprove(token: IERC20, spender: address, value: uint256)
safeIncreaseAllowance(token: IERC20, spender: address, value: uint256)
safeDecreaseAllowance(token: IERC20, spender: address, value: uint256)
forceApprove(token: IERC20, spender: address, value: uint256)
safePermit(token: IERC20Permit, owner: address, spender: address, value: uint256, deadline: uint256, v: uint8, r: bytes32, s: bytes32)
<<Struct>>
PlayerBetData
contracts/Jackpot.sol
maticAmount: uint256
ethAmount: uint256
usdcAmount: uint256
usdAmount: uint256

About SCRL

SCRL (Previously name SECURI LAB) was established in 2020, and its goal is to deliver a security solution for Web3 projects by expert security researchers. To verify the security of smart contracts, they have developed internal tools and KYC solutions for Web3 projects using industry-standard technology. SCRL was created to solve security problems for Web3 projects. They focus on technology for conciseness in security auditing. They have developed Python-based tools for their internal use called WAS and SCRL. Their goal is to drive the crypto industry in Thailand to grow with security protection technology.



Our top-tier security strategy combines static analysis, fuzzing, and a custom detector for maximum efficiency.

Follow Us On:

Website	https://scrl.io/
Twitter	https://twitter.com/scrl_io
Telegram	https://t.me/scrl_io
Medium	https://scrl.medium.com/