



Full Audit Report

DoubleUp Coinflip Security Assessment



DoubleUp Coinflip Security Assessment

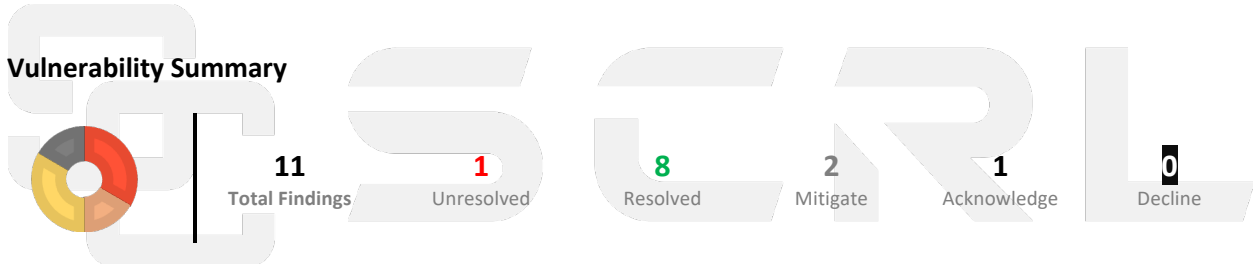
FULL AUDIT REPORTSecurity Assessment by SCRL on **Wednesday, June 19, 2024**

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 Coinflip	Solidity	Whitebox	Public	Polygon	0x86773A0eA8a7d8867aDA4E55F12278e408Ff283E
Report Version	Twitter	Telegram	Website		
1.2	https://twitter.com/doubleup_org	https://t.me/doubleup_org	https://doubleup.org/		

Scoring:**Vulnerability Summary**▪ **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****1 Mitigate, 1 Resolved**

High-severity issues should be addressed quickly to reduce the risk of exploitation and protect users' funds and data.

▪ **1 Medium****1 Mitigate**

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

▪ **5 Gas-optimization****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/Coinflip.sol	f80d179a4ec4a1e3751c902ded2c5dd9fe1aefe9

Audit Version History:

Version	Date	Description
1.0	Tuesday, May 21, 2024	Preliminary Report
1.1	Sunday, June 2, 2024	Update with re-assessment on github commit bca6bb66faeac193897a771a9f3e9bff09c247d0
1.2	Wednesday, 19 June R 2024	Update with re-assessment on deployed contrct address 0x86773A0eA8a7d8867aDA4E55F12278e408Ff283E


Audit information:

Request Date	Audit Date	Re-assessment Date
Thursday, May 16, 2024	Tuesday, May 21, 2024	Wednesday, June 19, 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 June 19, 2024



Security Assessment Author

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

Digital Sign

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.

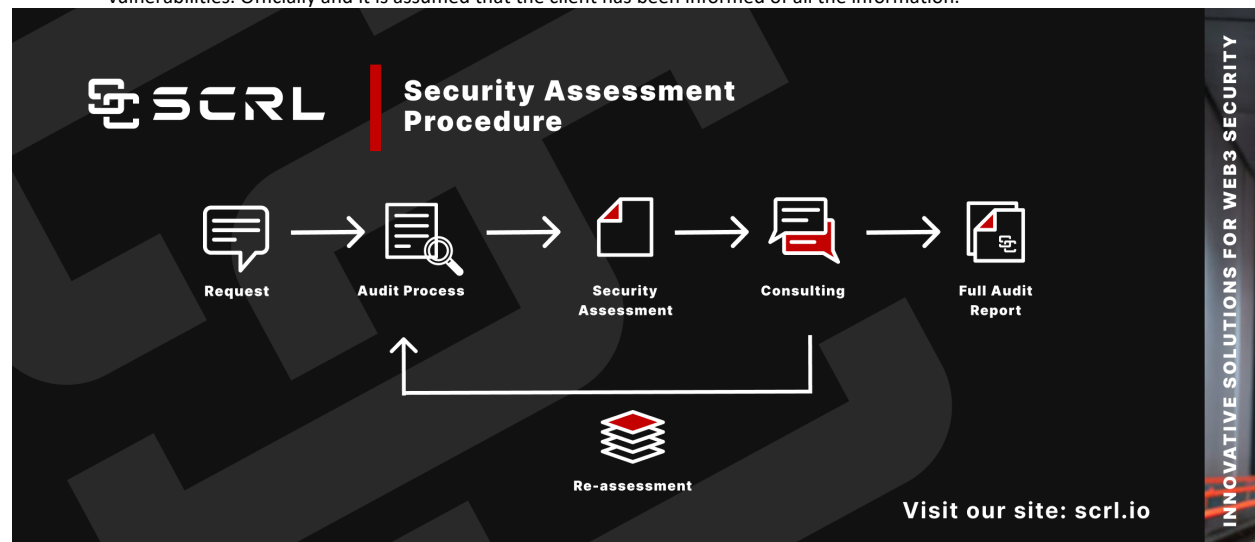
If the **service provider** finds a vulnerability The **service provider** will notify the **service user** via the Preliminary Report, which will be kept confidential for security. The **service provider** disclaims responsibility in the event of any attacks occurring whether before conducting a security 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	Low	Medium	High
Impact [Likelihood]			
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



For **Informational & Non-class/Optimization/Best-practices** will not be counted 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 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

Summary

- Executive Summary
- CVSS Scoring
- Vulnerability Summary
- Audit Scope
- Audit Version History
- Audit Information
- Smart Contract Audit Summary
- Security Assessment Author
- Digital Sign
- Disclaimer
- Security Assessment Procedure
- Risk Rating
- Category

Source Code Detail

- Dependencies / External Imports
- Visibility, Mutability, Modifier function testing





Vulnerability Finding

- Vulnerability
- SWC Findings
- Contract Description
- Inheritance Relational Graph
- UML Diagram

About SCRL

Source Units Analyzed: 1

Source Units in Scope: 1 (100%)

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/Coinflip.sol	1		492	477	378	26	211	
	Totals	1		492	477	378	26	211	

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				
12	4				
External	Internal	Private	Pure	View	
12	14	0	0	1	

StateVariables

Total	 Public
19	11

Capabilities

Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
<code>^0.8.20</code>		<code>yes</code>			
 Transfers ETH	 Low-Level Calls	 DelegateC all	 Uses Hash Functions	 ECRecover	 New/Create/Create2
 TryCatch	Σ Unchecked				

Dependencies / External Imports

Dependency / Import Path	Count
@chainlink/contracts/src/v0.8/vrf/dev/VRFCConsumerBaseV2Plus.sol	1
@chainlink/contracts/src/v0.8/vrf/dev/interfaces/IVRFCCoordinatorV2Plus.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	<code>fulfillRandomWords</code> Function must not revert	High	Logical Issue	Mitigate
CEN-01	Centralization Risk	Medium	Centralization	Mitigate
OPN-01	Unsafe ERC20 operation(s)	Low	Best Practices	Resolved
SEC-01	Missing Zero Address Validation (missing-zero-check)	Low	Best Practices	Resolved
SEC-02	Conformity to Solidity naming conventions (naming-convention)	Informational	Naming Conventions	Acknowledge
GAS-01	Use Custom Errors	Gas-optimization	Gas Optimization	Resolved
GAS-02	Long revert strings	Gas-optimization	Gas Optimization	Resolved
GAS-03	Functions guaranteed to revert when called by normal users can be marked <code>payable</code>	Gas-optimization	Gas Optimization	Resolved
GAS-04	<code>++i</code> costs less gas than <code>i++</code> , especially when it's used in <code>for</code> -loops (<code>--i</code> / <code>i--</code> too)	Gas-optimization	Gas Optimization	Resolved
GAS-05	Use <code>!= 0</code> instead of <code>> 0</code> for unsigned integer comparison	Gas-optimization	Gas Optimization	Resolved

REG-01: Potential Reentrancy Attack

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

Finding:

```
Function pickWinner(uint256 gameId, uint256 randomWord) (Coinflip.sol:227-276)
```

Description:

The `pickWinner` function in the Coinflip contract uses the `call` method with `value` to transfer Ether. This operation is susceptible to reentrancy attacks, where an attacker could re-enter the contract and manipulate its state before the original function call is completed.

Impact:

An attacker could exploit this vulnerability to re-enter the contract and modify its state, potentially draining the contract's funds or causing other unintended behavior.

Recommendation:

To mitigate this issue, consider using OpenZeppelin's ReentrancyGuard or ensuring that all state changes occur before any external calls.

References: SWC-107: Reentrancy: <https://swcregistry.io/docs/SWC-107>
OpenZeppelin ReentrancyGuard:
<https://docs.openzeppelin.com/contracts/4.x/api/security#ReentrancyGuard>

Alleviation:

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	Mitigate

Finding:

```
Function fulfillRandomWords(uint256 _requestId, uint256[] memory _randomWords)
(Coinflip.sol:341-356)
```

Description:

The fulfillRandomWords function in the Coinflip contract serves as a callback from the Chainlink VRF (Verifiable Random Function) to provide randomness for determining the game winner. If this function reverts, it can prevent the randomness from being generated and potentially lock the game funds indefinitely.

We recommend splitting the pickWinner 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.

Impact:

If the fulfillRandomWords function reverts, the game funds could be locked indefinitely, preventing participants from finalizing the game and receiving their winnings.

Recommendation:

To enhance the contract's resilience and reliability, we recommend splitting the pickWinner function from fulfillRandomWords. The fulfillRandomWords function should only store the generated random word from Chainlink VRF, and the pickWinner function should be made external to allow either participant to finalize the game using the stored random word.

References: Chainlink VRF: <https://docs.chain.link/vrf/v2/introduction>

Alleviation:

The doubleup team already mitigate this issue if there are any issue with random result from chainlink, that cancel game will auto cancelled after 1 hour or user cancel it after 3 minutes

CEN-01: Centralization Risk

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

Finding:

File: Coinflip.sol

```
278:     function cancelGame(uint256 gameId) external onlyOwner {  
309:     } external onlyOwner {
```

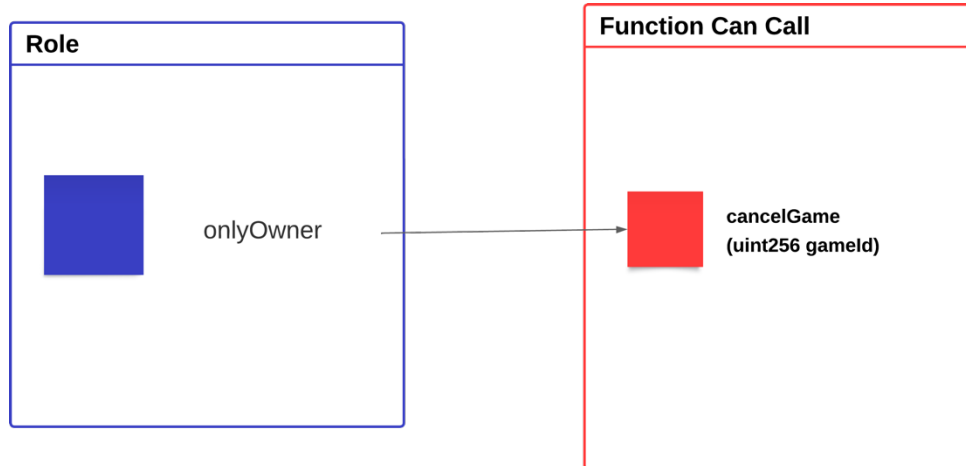
Explain Function Capability:

The contract provides several functions:

1. `cancelGame(uint256 gameId)`

The `cancelGame` function enables the contract owner to cancel a game under certain conditions and return the staked amount to the initial player who created the game. The `onlyOwner` modifier ensures that only the contract owner can execute this function.

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 use multi-signature; it will mitigate this centralization risk, but still remember **doubleup team still can call this centralized function.**

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: Coinflip.sol

```

156:     IERC20(gameToken).transferFrom(msg.sender, address(this), amount)
213:     IERC20(gameData.gameToken).transferFrom(
256:     IERC20(gameData.gameToken).transfer(
262:     IERC20(gameData.gameToken).transfer(
297:     IERC20(gameData.gameToken).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-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:

- ✗ Coinflip.constructor(address,address,address)._USDC (src/Coinflip.sol:104) lacks a zero-check on :
 - USDC = _USDC (src/Coinflip.sol#109)
- ✗ Coinflip.constructor(address,address,address)._WETH (src/Coinflip.sol:105) lacks a zero-check on :
 - WETH = _WETH (src/Coinflip.sol#110)
- ✗ Coinflip.constructor(address,address,address)._treasury (src/Coinflip.sol:106) lacks a zero-check on :
 - serviceTreasury = _treasury (src/Coinflip.sol#111)

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.

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 Coinflip.fulfillRandomWords(uint256,uint256[])._randomWords (src/Coinflip.sol:343) is not in mixedCase
✗ Parameter Coinflip.fulfillRandomWords(uint256,uint256[])._requestId (src/Coinflip.sol:342) is not in mixedCase
✗ Parameter Coinflip.getRequestStatus(uint256)._requestId (src/Coinflip.sol:359) is not in mixedCase
✗ Parameter Coinflip.updateMaxGameOpenTime(uint256)._maxGameOpenTime (src/Coinflip.sol:308) is not in mixedCase
✗ Variable Coinflip.COORDINATOR (src/Coinflip.sol:70) is not in mixedCase
✗ Variable Coinflip.USDC (src/Coinflip.sol:17) is not in mixedCase
✗ Variable Coinflip.WETH (src/Coinflip.sol:16) is not in mixedCase
✗ Variable Coinflip.s_requests (src/Coinflip.sol:68-69) 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: Use Custom Errors

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

Finding:

File: Coinflip.sol

```
117:         require(msg.value > 0, "Amount should be greater than 0.");
149:         require(amount > 0, "Amount should be greater than 0.");
185:         require(gameId < gameDatas.length, "Invalid game id");
188:         require(gameData.player2 == address(0), "Already other player joined");
193:         require(msg.value == gameData.amount, "Amount should be same.");
202:         require(gameId < gameDatas.length, "Invalid game id");
205:         require(gameData.player2 == address(0), "Already other player joined");
210:         require(amount == gameData.amount, "Amount should be same.");
228:         require(gameId < gameDatas.length, "Invalid game id");
234:         require(gameData.player2 != address(0), "Other player didn't join yet");
250:         require(send1, "Transfer failed.");
253:         require(send2, "Transfer failed.");
279:         require(gameId < gameDatas.length, "Invalid game id");
282:         require(!gameData.cancelFlag, "This game has already been cancelled.");
283:         require(gameData.player2 == address(0), "Already other player joined");
294:         require(send, "Transfer failed.");
345:         require(s_requests[_requestId].exists, "request not found");
```

```
346:         require(requestIDGames[_requestId] != 0, "request id game not found");  
361:         require(s_requests[_requestId].exists, "request not found");  
...  

```

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-02: Long revert strings

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

Finding:

File: Coinflip.sol

```
282:         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-03: 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 <code>payable</code>	Gas-optimization	Check on finding	Gas Optimization	Resolved

Finding:

File: Coinflip.sol

```
278:     function cancelGame(uint256 gameId) external onlyOwner {
    ...
```

Recommendation:

The `cancelGame` function **can be marked as payable** to optimize gas usage since it is restricted to the contract owner using the `onlyOwner` modifier. This avoids unnecessary checks for zero Ether being sent.

Alleviation:

The doubleup team has already resolved this issue.

GAS-04: `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)

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

Finding:

File: Coinflip.sol

```
141:         gameCounter++;
181:         gameCounter++;
...
```

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.

```
141:         ++gameCounter;
181:         ++gameCounter;
...
```

Alleviation:

The doubleup team has already resolved this issue.

GAS-05: 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	Resolved

Finding:

File: Coinflip.sol

```
117:         require(msg.value > 0, "Amount should be greater than 0.");  
  
149:         require(amount > 0, "Amount should be greater than 0.");  
  
...
```

Recommendation:

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

Replace > 0 with != 0 for gas optimization.

```
117:         require(msg.value != 0, "Amount should be greater than 0.");  
  
149:         require(amount != 0, "Amount should be greater than 0.");  
  
...
```

Alleviation:

The doubleup team has already resolved this issue.















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

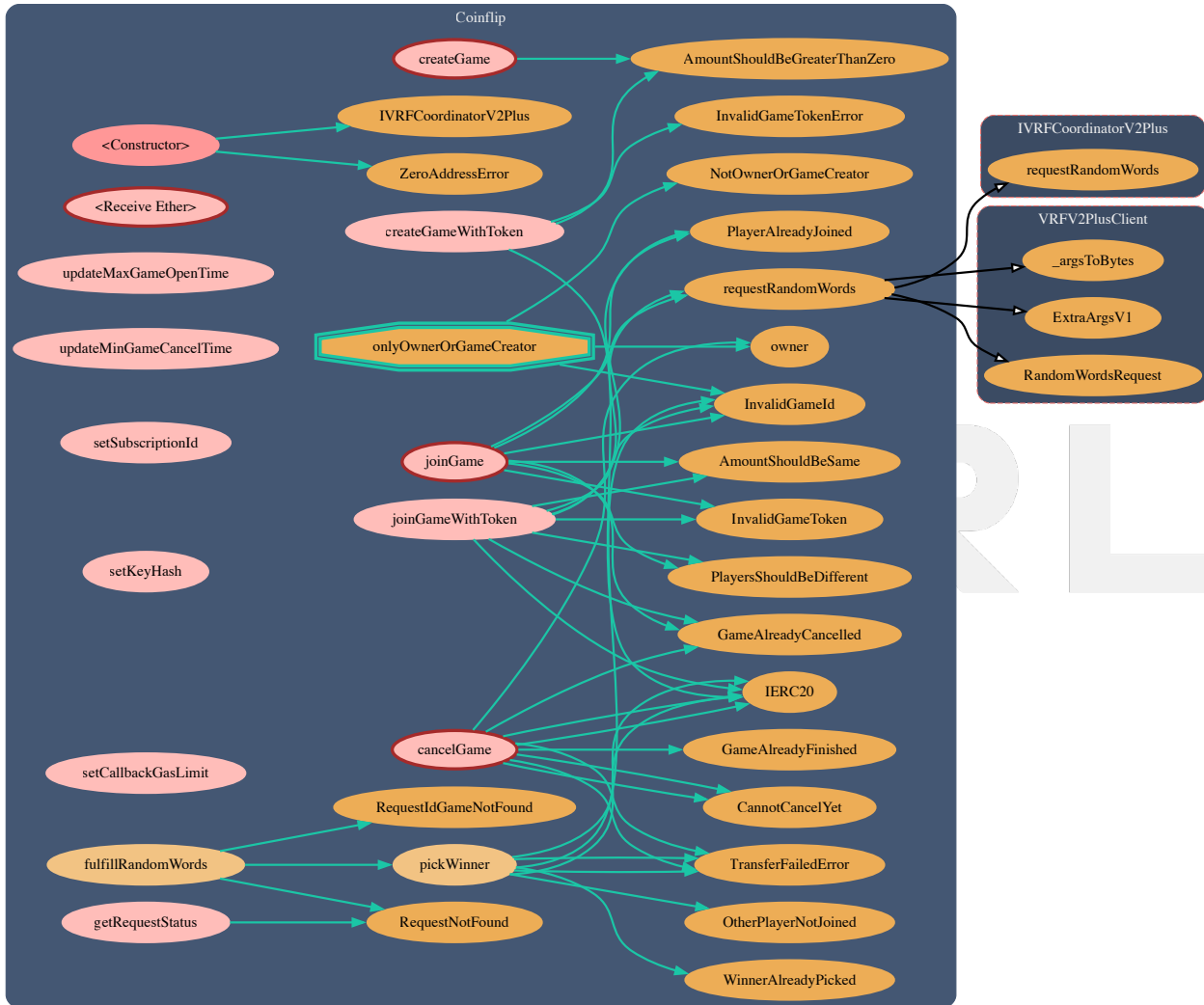
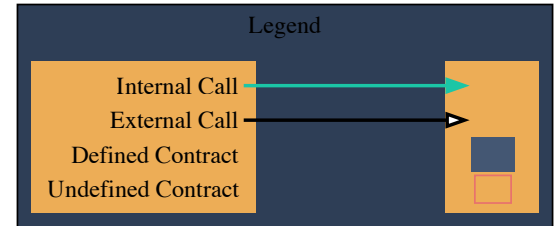
Contr act	Type	Bases		
L	Function Name	Visibility	Mutabi lity	Modifiers
Coinfl ip	Implementation	VRFConsumerBase V2Plus, ReentrancyGuard		
L		Public !		VRFConsumerBas eV2Plus
L		External !		NO !
L	createGame	External !		NO !
L	createGameWithTok en	External !		NO !
L	joinGame	External !		NO !
L	joinGameWithToken	External !		NO !
L	pickWinner	Internal 		nonReentrant
L	cancelGame	External !		onlyOwnerOrGame Creator
L	updateMaxGameOp enTime	External !		onlyOwner
L	updateMinGameCan celTime	External !		onlyOwner
L	setSubscriptionId	External !		onlyOwner
L	setKeyHash	External !		onlyOwner
L	setCallbackGasLimit	External !		onlyOwner
L	requestRandomWor ds	Internal 		
L	fulfillRandomWords	Internal 		

Contr act	Type	Bases		
L	getRequestStatus	External !		NO !

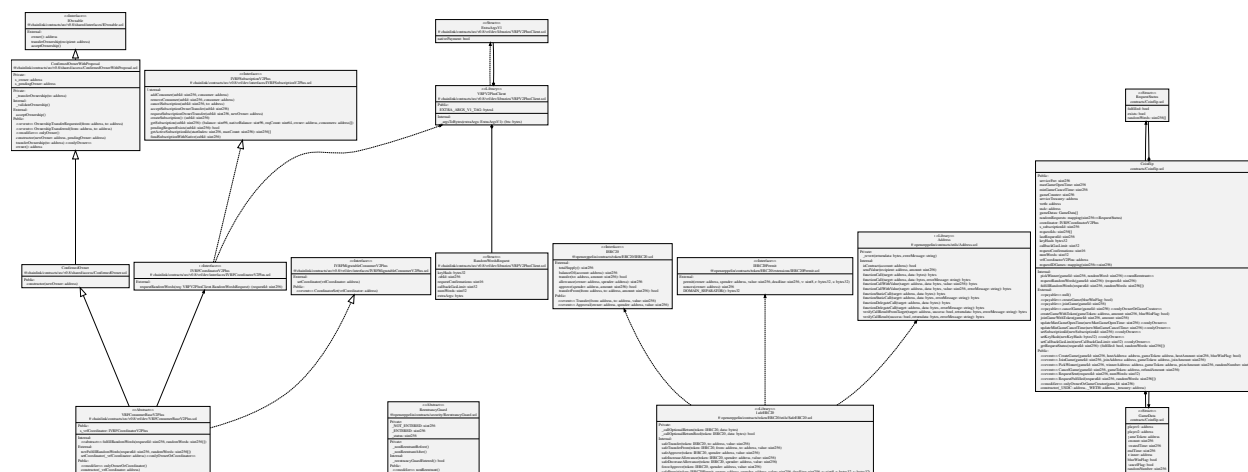
Legend

Symbol	Meaning
	Function can modify state
	Function is payable

Call Graph



UML Class Diagram



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.



Support ALL EVM L1 - L2

Smart Contract Audit

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

scrl.io



Follow Us On:

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