Cyberscope

Audit Report

EtherWars

May 2023

EtherWars.sol 3c4ab80c48e3a6178e34679578ac2b055a06b5068e09f78524fe34a529461bbe

EtherWarsQrng.sol 59c3927ee26caaaf0fee7ad1dfdf96dee866f1484babf9190d8279bab1b42dba

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 3
Introduction 4
Roles 5
EtherWars Contract 5
Owner 5
QRNGConsumer 5
SpinToWin 5
User 6
EtherWarsQrng Contract 7
Owner 7
AirnodeRrp 7
EtherWars 7
Test Deployments 8
Findings Breakdown 9
Diagnostics 10
IWF - Inaccurate Withdraw Funds 12
Description 12
Recommendation 13
RED - Redundant Event Declaration 14
Description 14
Recommendation 14
RCS - Redundant Conditional Statement 15
Description 15
Recommendation 15
AOI - Arithmetic Operations Inconsistency 16
Description 16
Recommendation 16
DKO - Delete Keyword Optimization 17
Description 17
Recommendation 17
RC - Redundant Calculations 18
Description 18
Recommendation 18
CR - Code Repetition 19
Description 19
Recommendation 19

MU - Modifiers Usage 20
Description 20
Recommendation 20
SW - Stops Withdrawals 21
Description 21
Recommendation 21
RS - Redundant Subtraction 22
Description 22
Recommendation 22
MC - Missing Check 23
Description 23
Recommendation 23
UCS - Unreachable Code Segments 24
Description 24
Recommendation 24
IDI - Immutable Declaration Improvement 25
Description 25
Recommendation 25
L02 - State Variables could be Declared Constant 26
Description 26
Recommendation 26
L04 - Conformance to Solidity Naming Conventions 27
Description 27
Recommendation 28
L07 - Missing Events Arithmetic 29
Description 29
Recommendation 29
L11 - Unnecessary Boolean equality 30
Description 30
Recommendation 30
L13 - Divide before Multiply Operation 31
Description 31
Recommendation 31
L16 - Validate Variable Setters 32
Description 32
Recommendation 32
Functions Analysis **33**
Inheritance Graph **35**
Flow Graph **36**
Summary **37**
Disclaimer **38**
About Cyberscope **39**

Review

Audit Updates

Initial Audit	18 May 2023

Source Files

Filename	SHA256
EtherWars.sol	3c4ab80c48e3a6178e34679578ac2b055a06b5068e09f78524fe34a529461bbe
EtherWarsQrng.sol	59c3927ee26caaaf0fee7ad1dfdf96dee866f1484babf9190d8279bab1b42dba

Introduction

The SpinToWin ecosystem consists of various contracts. This audit report focuses on the EtherWars and EtherWarsQrng contracts. EtherWars is a decentralized gaming platform implemented as a smart contract. Users can engage in combat battles and compete for rewards. The contract integrates features such as strength, cooldowns, redemption points, and spin points to create an engaging gameplay experience. The contract relies on the OpenZeppelin library for security and access control. An external QRNGConsumer contract is used for random number generation to determine the outcome of battles. Users can enhance their combat abilities by increasing their power through additional Ether contributions. Spin points earned through the SpinToWin contract can be used to participate in the EtherWars game. The contract offers configurable parameters and supports user withdrawals.

Roles

EtherWars Contract

Owner

The Owner role has authority over the following functions:

- `function ownerWithdrawFees()`
- `function setMinimumStrength(uint256 _strength)`
- `function setDevFee(uint256 _devFee)`
- `function setRedemptionPointsCost(uint256 _cost)`
- `function setCooldownReduction(uint256 _time)`
- `function setCooldownTime(uint256 _time)`
- `function setQRNGConsumer(address _qrngConsumer)`
- `function enableArena()`
- `function disableArena()`

QRNGConsumer

The QRNGConsumer role has authority over the following functions:

- `function beginCombat(address _attacker, uint256[] calldata _randomWords)`

SpinToWin

The SpinToWin role has authority over the following functions:

- `function deductSpinPoints(address _user, uint256 _points)`

User

The User role can interact with the following functions:

- `function enterArena()`
- `function attack()`
- `function reduceCooldown()`
- `function userWithdraw(uint256 _amount)`
- `function increasePower(address _contender)`
- `function numberOfContenders()`
- `function getFightList()`

EtherWarsQrng Contract

Owner

The Owner role has authority over the following functions:

- `function setRequestParameters(address _airnode, bytes32 _endpointIdUint256Array, address _sponsorWallet)`
- `function setEtherWarsAddress(address _address)`
- `function setAirnodeAddress(address _address)`
- `function setSponsorWalletAddress(address _address)`
- `function setAirnodeAddress(bytes32 _endpoint)`

AirnodeRrp

The AirnodeRrp role has authority over the following functions:

- `function fulfillUint256Array(bytes32 _requestId, bytes calldata _data)`

EtherWars

The EtherWars role has authority over the following functions:

- `function makeRequestUint256Array(uint256 _size, address _attacker)`

Test Deployments

Contract	Explorer	Address
EtherWars	https://testnet.bscscan.com/address/0xDddd49FFCF872ABf61720cBf6b7908f217a7dFE0	0xDddd49FFCF872ABf61720cBf6b7908f217a7dFE0
EtherWarsQrng	https://testnet.bscscan.com/address/0x23f855D8AE3E753Eb8aBDA8266a10571399Cbe99	0x23f855D8AE3E753Eb8aBDA8266a10571399Cbe99

Findings Breakdown

Critical	1
Medium	0
Minor / Informative	18

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	18	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	IWF	Inaccurate Withdraw Funds	Unresolved
●	RED	Redundant Event Declaration	Unresolved
●	RCS	Redundant Conditional Statement	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	DKO	Delete Keyword Optimization	Unresolved
●	RC	Redundant Calculations	Unresolved
●	CR	Code Repetition	Unresolved
●	MU	Modifiers Usage	Unresolved
●	SW	Stops Withdrawals	Unresolved
●	RS	Redundant Subtraction	Unresolved
●	MC	Missing Check	Unresolved
●	UCS	Unreachable Code Segments	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved

●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved

IWF - Inaccurate Withdraw Funds

Criticality	Critical
Location	EtherWars.sol#L212
Status	Unresolved

Description

All users have the authority to withdraw their funds or a portion of them by calling the `userWithdraw` function. The given amount is subtracted from the user's funds and then sent to the user. Additionally, if the user's remaining funds are less than the minimum amount that is required to participate in the game, then the user is removed from the list. If the user's remaining funds are greater than zero but still less than the minimum amount, the contract will still proceed with the user's removal from the list but does not add the remaining funds to the withdrawal amount. As a result, the user will lose the remaining funds.

```
function userWithdraw(uint256 _amount) external nonReentrant {
    address user = msg.sender;

    if (_amount == 0) revert NoAmountIncluded();
    if (contenderCooldown[user] > block.timestamp)
        revert StillInCooldown(contenderCooldown[user],
block.timestamp);
    if (contenderStrength[user] < _amount) revert NotEnoughFunds();

    contenderStrength[user] = SafeMath.sub(contenderStrength[user],
_amount);

    // Remove user from the arena if strength is less then
minimumAttack
    if (contenderStrength[user] < minimumStrength) {
        removeFromList(user);
    }

    sendViaCall(payable(user), _amount);
    emit UserWithdrawal(user, _amount);
}
```

Recommendation

The team is advised to appropriately handle this case so that users will not lose any of their funds.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	EtherWars.sol#L45
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract declares the event `EmergencyWithdrawal` but it is not used in the contract. As a result, the event is redundant.

```
event EmergencyWithdrawal(uint256 amount);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	EtherWars.sol#L88
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract executes two conditional statements to check if a contender's strength is less than the minimum amount. The contract could achieve the same goal with just one if-statement by just performing the following check `contenderStrength[contender] + msg.value < minimumStrength` .

```
// If contender already has an ETH balance in the contract, then
combine that with the msg.value
if (contenderStrength[contender] > 0 &&
contenderStrength[contender] + msg.value < minimumStrength)
    revert NotEnoughStrength();
// If contender has no ETH balance in the contract, then check the
msg.value only
if (contenderStrength[contender] == 0 && msg.value <
minimumStrength) revert NotEnoughStrength();
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. A suggested implementation would be the following:

```
if (contenderStrength[contender] + msg.value < minimumStrength)
revert NotEnoughStrength();
```

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	EtherWars.sol
Status	Unresolved

Description

The contract demonstrates an inconsistency in arithmetic operations. To ensure consistency and safety, all operations should be executed using one source of truth. The given code segment is just an example of this inconsistency, where the addition operation (`SafeMath.add(attackStrength, defenseStrength)`) is performed using the SafeMath library, while the next operation is using a native arithmetic expression.

```
uint256 totalCombatStrength = SafeMath.add(attackStrength,
defenseStrength);

uint256 attackChance = (attackStrength * 1000) /
totalCombatStrength;
```

Recommendation

The team is advised to use only one source of truth for the arithmetic operations.

DKO - Delete Keyword Optimization

Criticality	Minor / Informative
Location	EtherWars.sol#L158,180,181
Status	Unresolved

Description

The contract resets variables to the default state by setting the initial values. Setting values to state variables increases the gas cost.

```
contenderStrength[defender] = 0;
contenderStrength[_attacker] = 0;
contenderCooldown[_attacker] = 0;
```

Recommendation

The team is advised to use the `delete` keyword instead of setting variables. This can be more efficient than setting the variable to a new value, using delete can reduce the gas cost associated with storing data on the blockchain.

RC - Redundant Calculations

Criticality	Minor / Informative
Location	EtherWars.sol#L134
Status	Unresolved

Description

Redundant calculations can occur in various forms in a smart contract, such as repetitive calculations for the same result, the recalculation of unchanging values, and the repeated execution of complex computations. These redundant calculations can significantly increase the gas cost of executing the smart contract. The contract executes redundant calculations in the segments listed below.

The operation `SafeMath.mul(attackChance, 104) / 100` is evaluated twice if the result of the operation is less than or equal to 1000.

```
if (SafeMath.mul(attackChance, 104) / 100 <= 1000) {
    attackChance = SafeMath.mul(attackChance, 104) / 100;
} else {
    attackChance = 1000; // Must be 1000 for 0.01% chance to win
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

The contract could store the result in a variable first. A recommended approach would be the following:

```
attackChance = SafeMath.mul(attackChance, 104) / 100;
if (attackChance > 1000) {
    attackChance = 1000; // Must be 1000 for 0.01% chance to win
}
```

CR - Code Repetition

Criticality	Minor / Informative
Location	EtherWars.sol#L145,170
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
// Some % fee taken from the defender
devFee = SafeMath.div(SafeMath.mul(contenderStrength[defender],
devFeePercentage), 100);
// Add defender's strength to attacker, and subtract dev fee
contenderStrength[_attacker] = SafeMath.add(
    contenderStrength[_attacker],
    SafeMath.sub(contenderStrength[defender], devFee)
);
// Add 5 spin points to attacker and 3 to defender
spinPoints[_attacker] += 5;
spinPoints[defender] += 3;
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	EtherWars.sol#L88,91,106
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
// If contender already has an ETH balance in the contract, then
combine that with the msg.value
if (contenderStrength[contender] > 0 &&
contenderStrength[contender] + msg.value < minimumStrength)
    revert NotEnoughStrength();
// If contender has no ETH balance in the contract, then check the
msg.value only
if (contenderStrength[contender] == 0 && msg.value <
minimumStrength) revert NotEnoughStrength();

if (contenderStrength[attacker] < minimumStrength) revert
NotEnoughStrength();
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

SW - Stops Withdrawals

Criticality	Minor / Informative
Location	EtherWars.sol#L216
Status	Unresolved

Description

The contract owner has the authority to stop the withdrawals for all users. The owner may take advantage of it by setting the `cooldownTime` to a large value.

```
if (contenderCooldown[user] > block.timestamp)
    revert StillInCooldown(contenderCooldown[user],
block.timestamp);
```

Recommendation

The contract could embody a check for not allowing setting the `cooldownTime` less than a reasonable amount. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

RS - Redundant Subtraction

Criticality	Minor / Informative
Location	EtherWars.sol#L96
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract performs one redundant subtraction by adding the contender to the list before assigning the contender's index to the `contenderListPosition` variable.

```
list.push(contender);
contenderListPosition[contender] = list.length - 1;
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

The contract could swap the execution of these code lines to optimize the code. A possible approach is the following:

```
contenderListPosition[contender] = list.length;
list.push(contender);
```

MC - Missing Check

Criticality	Minor / Informative
Location	EtherWars.sol#L254
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The `devFeePercentage` variable represents the percentage value that is used to calculate the fee amount that is subtracted from the amount given from a user upon participation. The percentage must be less than 100.

```
function setDevFee(uint256 _devFee) external onlyOwner {
    devFeePercentage = _devFee;
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

UCS - Unreachable Code Segments

Criticality	Minor / Informative
Location	EtherWars.sol#L162,184
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The variable `redemptionPercentage` is set to zero and its value never changes. As a result the following code segments will never execute.

Additionally, the function `reduceCooldown` will always revert.

```
if (redemptionPercentage > 0) {
    contenderRedemptionPoints[defender] = SafeMath.add(
        contenderRedemptionPoints[defender],
        SafeMath.div(attackChance, redemptionPercentage)
    );
}

if (redemptionPercentage > 0) {
    contenderRedemptionPoints[_attacker] = SafeMath.add(
        contenderRedemptionPoints[_attacker],
        SafeMath.div(SafeMath.sub(1000, attackChance),
redemptionPercentage)
    );
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	EtherWars.sol#L67
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
SPINTOWIN
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	EtherWars.sol#L24
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 public redemptionPercentage = 0
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	EtherWarsQrng.sol#L45,65,85,86,87,98,103,108,113EtherWars.sol#L16,113,114,212,231,238,249,254,258,262,266,270,291,301
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint256 _size
address _attacker
bytes32 _requestId
bytes calldata _data
address _airnode
bytes32 _endpointIdUint256Array
address _sponsorWallet
address _address
bytes32 _endpoint
address public SPINTOWIN
uint256[] calldata _randomWords
uint256 _amount
address _contender
uint256 _points

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	EtherWars.sol#L251,255,259,263,267
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
minimumStrength = _strength
devFeePercentage = _devFee
redemptionPointsCost = _cost
cooldownReduction = _time
cooldownTime = _time
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	EtherWars.sol#L81
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
isOnline == false
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	EtherWars.sol#L132,134,135
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 attackChance = (attackStrength * 1000) /
totalCombatStrength
attackChance = SafeMath.mul(attackChance, 104) / 100
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	EtherWars.sol#L67
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
SPINTOWIN = _spinToWin
```

Recommendation

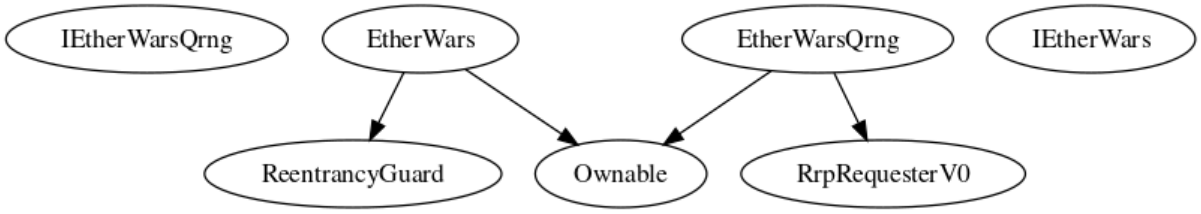
By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

Functions Analysis

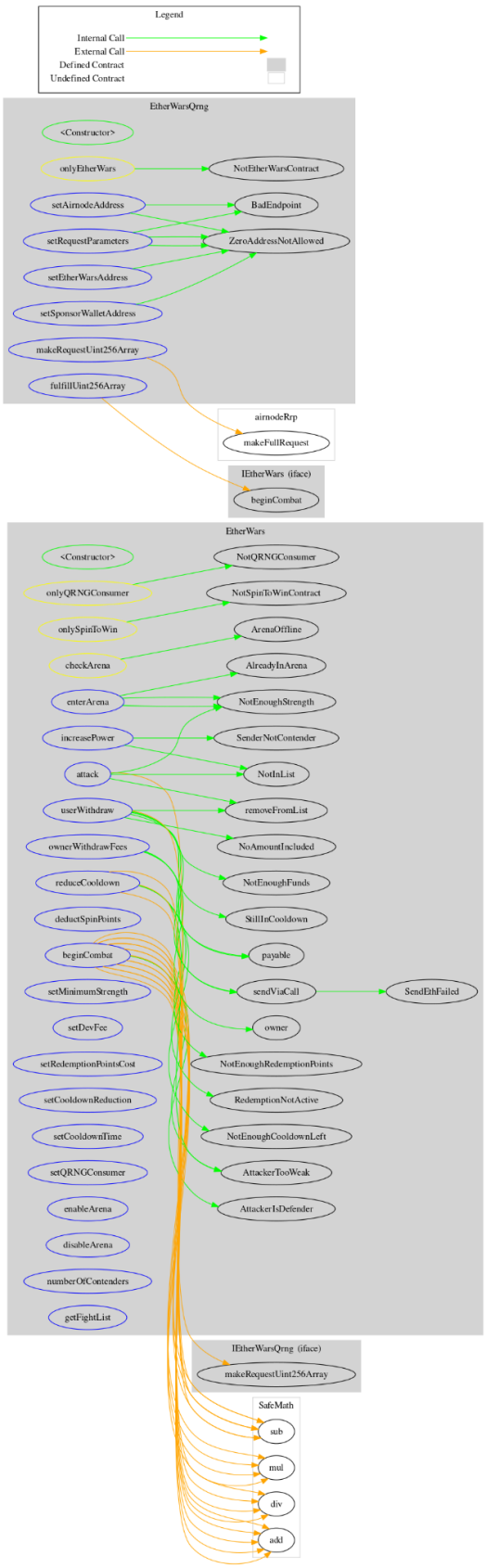
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IEtherWarsQrng	Interface			
	makeRequestUint256Array	External	✓	-
EtherWars	Implementation	Ownable, ReentrancyGuard		
		Public	✓	-
	enterArena	External	Payable	checkArena
	attack	External	✓	nonReentrant checkArena
	beginCombat	External	✓	onlyQRNGConsumer nonReentrant
	reduceCooldown	External	✓	nonReentrant checkArena
	userWithdraw	External	✓	nonReentrant
	increasePower	External	Payable	checkArena
	deductSpinPoints	External	✓	onlySpinToWin
	ownerWithdrawFees	External	✓	onlyOwner
	setMinimumStrength	External	✓	onlyOwner
	setDevFee	External	✓	onlyOwner
	setRedemptionPointsCost	External	✓	onlyOwner

	setCooldownReduction	External	✓	onlyOwner
	setCooldownTime	External	✓	onlyOwner
	setQRNGConsumer	External	✓	onlyOwner
	enableArena	External	✓	onlyOwner
	disableArena	External	✓	onlyOwner
	numberOfContenders	External		-
	getFightList	External		-
	removeFromList	Private	✓	
	sendViaCall	Private	✓	
IEtherWars	Interface			
	beginCombat	External	✓	-
EtherWarsQrng	Implementation	RrpRequesterV0, Ownable		
		Public	✓	RrpRequesterV0
	makeRequestUint256Array	External	✓	onlyEtherWars
	fulfillUint256Array	External	✓	onlyAirnodeRrp
	setRequestParameters	External	✓	onlyOwner
	setEtherWarsAddress	External	✓	onlyOwner
	setAirnodeAddress	External	✓	onlyOwner
	setSponsorWalletAddress	External	✓	onlyOwner
	setAirnodeAddress	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
EtherWarsQrng
<Constructor>
onlyEtherWars
NotEtherWarsContract
setAirnodeAddress
BadEndpoint
setRequestParameters
ZeroAddressNotAllowed
setEtherWarsAddress
setSponsorWalletAddress
makeRequestUint256Array
fulfillUint256Array
airnodeRrp
makeFullRequest
IEtherWars (iface)
beginCombat
EtherWars
<Constructor>
NotQRNGConsumer
onlyQRNGConsumer
NotSpinToWinContract
onlySpinToWin
ArenaOffline
checkArena
AlreadyInArena
enterArena
NotEnoughStrength
increasePower
SenderNotContender
attack
NotInList
userWithdraw
removeFromList
ownerWithdrawFees
NoAmountIncluded
reduceCooldown
NotEnoughFunds
deductSpinPoints
StillInCooldown
beginCombat
payable
setMinimumStrength
sendViaCall
SendEthFailed
setDevFee
owner
setRedemptionPointsCost
NotEnoughRedemptionPoints
setCooldownReduction
RedemptionNotActive
setCooldownTime
NotEnoughCooldownLeft
setQRNGConsumer
AttackerTooWeak
enableArena
AttackerIsDefender
disableArena
numberOfContenders
getFightList
IEtherWarsQrng (iface)
makeRequestUint256Array
SafeMath
sub
mul
div
add

Summary

EtherWars contract implements a game and rewards mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope'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 regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document 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 represents an extensive assessment 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 Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope'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. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io