Cyberscope

Audit Report

EtherWars

July 2023

EtherWars	65e43a08fe68eabbbd27434456fd47a3d4d110e1c179052d3f76cda7a04aafe3
EtherWarsQrng	06f9a022da7cc60a4e59763de700e9c41bc8b2f6799d2ad48d6d2f407b373a76

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	18 May 2023 https://github.com/cyberscope-io/audits/blob/main/1-stw/v1/EtherWars.pdf
Corrected Phase 2	5 Jul 2023

Source Files

Filename	SHA256
EtherWars.sol	65e43a08fe68eabbbd27434456fd47a3d4d110e1c179052d3f76cda7a04aafe3
EtherWarsQrng.sol	06f9a022da7cc60a4e59763de700e9c41bc8b2f6799d2ad48d6d2f407b373a76

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 setMaximumStrength(uint256 _strength)`
- `function setMinimumStrength(uint256 _strength)`
- `function setDevFee(uint256 _devFee)`
- `function setRedemptionPercentage(uint256 _percentage)`
- `function setRedemptionPointsCost(uint256 _cost)`
- `function setCooldownReduction(uint256 _time)`
- `function setCooldownTime(uint256 _time)`
- `function setQRNGConsumer(address _qrngConsumer)`
- `function setAttackMultiplier(uint256 _multiplier)`
- `function setSpinToWinContract(address _address)`
- `function setMaxRandomNum(uint256 _num)`
- `function winChanceToggle(bool _attacker, bool _defender)`
- `function enableArena()`
- `function disableArena()`

QRNGConsumer

The QRNGConsumer role has authority over the following functions:

- `function beginCombat(address _attacker, string memory _username, uint256 _faction, 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(uint256 _faction, string calldata _username)`
- `function attack()`
- `function reduceCooldown()`
- `function userWithdraw(uint256 _amount)`
- `function increasePower(address _contender)`
- `function changeUsername(string calldata _newName)`
- `function getFightList()`
- `function numberOfContenders()`

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

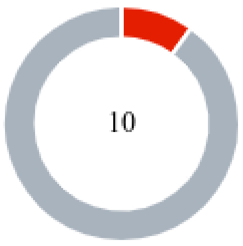
The EtherWars role has authority over the following functions:

- `function makeRequestUint256Array(uint256 _size, address _attacker, string memory _name, uint256 _faction)`

Test Deployments

Contract	Explorer	Address
EtherWars	https://testnet.bscscan.com/address/0x78dF54681A440A75B46498988faE89063322a79A	0x78dF54681A440A75B46498988faE89063322a79A
EtherWarsQrng	https://testnet.bscscan.com/address/0xd93eDdAC471B66E275f22868ff66C54D659130b5	0xd93eDdAC471B66E275f22868ff66C54D659130b5

Findings Breakdown



🟥	Critical	1
🟧	Medium	0
⬜	Minor / Informative	9

	Severity	Unresolved	Acknowledged	Resolved	Other
🟥	Critical	1	0	0	0
🟧	Medium	0	0	0	0
⬜	Minor / Informative	9	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🔴	IWF	Inaccurate Withdraw Funds	Unresolved
⚪	MSC	Missing Sanity Check	Unresolved
⚪	ICR	Invalid Contender Removal	Unresolved
⚪	MU	Modifiers Usage	Unresolved
⚪	VAO	Variable Assignment Optimization	Unresolved
⚪	MEE	Missing Events Emission	Unresolved
⚪	L04	Conformance to Solidity Naming Conventions	Unresolved
⚪	L06	Missing Events Access Control	Unresolved
⚪	L07	Missing Events Arithmetic	Unresolved
⚪	L13	Divide before Multiply Operation	Unresolved

IWF - Inaccurate Withdraw Funds

Criticality	Critical
Location	EtherWars.sol#L262
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 reassigns the amount to the user's entire balance.

However, the user's balance has already been deducted by the amount before checking for the minimum funds. For instance, a simple example is the following:

- A user has a total of 100 strengh (funds) and the `minimumStrength` is 80.
- The user calls the `userWithdraw` function with an `_amount = 40`.
- The `contenderStrength[user] = contenderStrength[user] - _amount;` operation will be executed, thus the user's balance will be deducted by 40. So the user's balance now becomes 60.
- The user's balance now is lower than the minimum, so the operation `_amount = contenderStrength[user];` will assign the user's balance to the amount, which is this case is 60.
- Hence, the user will not receive the full amount for the withdrawal.

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] = contenderStrength[user] - _amount;

    // Remove user from the arena if strength is less then minimumAttack
    if (contenderStrength[user] < minimumStrength) {
        removeFromList(user);
        // Use the player's entire balance for the withdrawal amount
        _amount = contenderStrength[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.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	EtherWars.sol#L316,322
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 `minimumStrength` should always be less than the `maximumStrength`.

```
function setMaximumStrength(uint256 _strength) external onlyOwner {
    require(_strength != 0, "Strength 0");
    maximumStrength = _strength;
    emit MaxStrengthSet(_strength);
}

function setMinimumStrength(uint256 _strength) external onlyOwner {
    require(_strength != 0, "Strength 0");
    minimumStrength = _strength;
    emit MinStrengthSet(_strength);
}
```

Recommendation

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

ICR - Invalid Contender Removal

Criticality	Minor / Informative
Location	EtherWars.sol#L230
Status	Unresolved

Description

As part of the `beginCombat` function flow a contender is removed from the list. If the attacker's stength is over the maximum threshold then the defender is removed. However, when the defender's strength is over the maximum threshold, the contract removes the defender from the list, not the attacker. As a result, the contract may not operate as expected.

```
if (contenderStrength[defender] >= maximumStrength) {
    removeFromList(defender);
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so that the `beginCombat` function works as expected.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	EtherWars.sol#L125,150
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 (contenderStrength[contender] + 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.

VAO - Variable Assignment Optimization

Criticality	Minor / Informative
Location	EtherWars.sol#L25
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 initializes the `maximumStrength` variable to the maximum value of the `uint256` variable type. Solidity already has a cleaner and safer way to use this value built-in.

```
uint256 public maximumStrength = 2 ** 256 - 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 team could modify the code segment to the following:

```
uint256 public maximumStrength = type(uint256).max;
```

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	EtherWars.sol#L358,362,367,382,386
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
qrngConsumer = IEtherWarsQrng(_qrngConsumer);
attackMultiplier = _multiplier;
spinToWinContract = _address;
isOnline = true;
isOnline = false;
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	EtherWarsQrng.sol#L53,54,55,56,80,102,103,104,115,120,125,130 EtherWars.sol#L122,159,160,161,162,262,283,291,305,316,322,328,334,340,345,350,356,361,365,370,375,397,411,425
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
string memory _name
uint256 _faction
bytes32 _requestId
bytes calldata _data
address _airnode
bytes32 _endpointIdUint256Array
address _sponsorWallet
address _address
bytes32 _endpoint
string calldata _username
string memory _username
uint256[] calldata _randomWords

...
```

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.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	EtherWars.sol#L367
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. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
spinToWinContract = _address
```

Recommendation

To avoid this issue, 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.

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	EtherWars.sol#L362
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.

```
attackMultiplier = _multiplier
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	EtherWars.sol#L178,181,219
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 = (attackCalculation * (100 + attackMultiplier)) / 100
contenderRedemptionPoints[defender] =
                    contenderRedemptionPoints[defender] +
                    ((attackChance * redemptionPercentage) / 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.

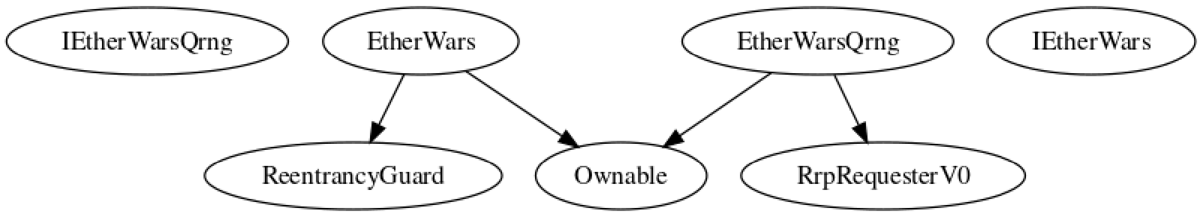
Functions Analysis

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
	changeUsername	External	✓	-
	deductSpinPoints	External	✓	onlySpinToWin
	ownerWithdrawFees	External	✓	onlyOwner
	setMaximumStrength	External	✓	onlyOwner
	setMinimumStrength	External	✓	onlyOwner

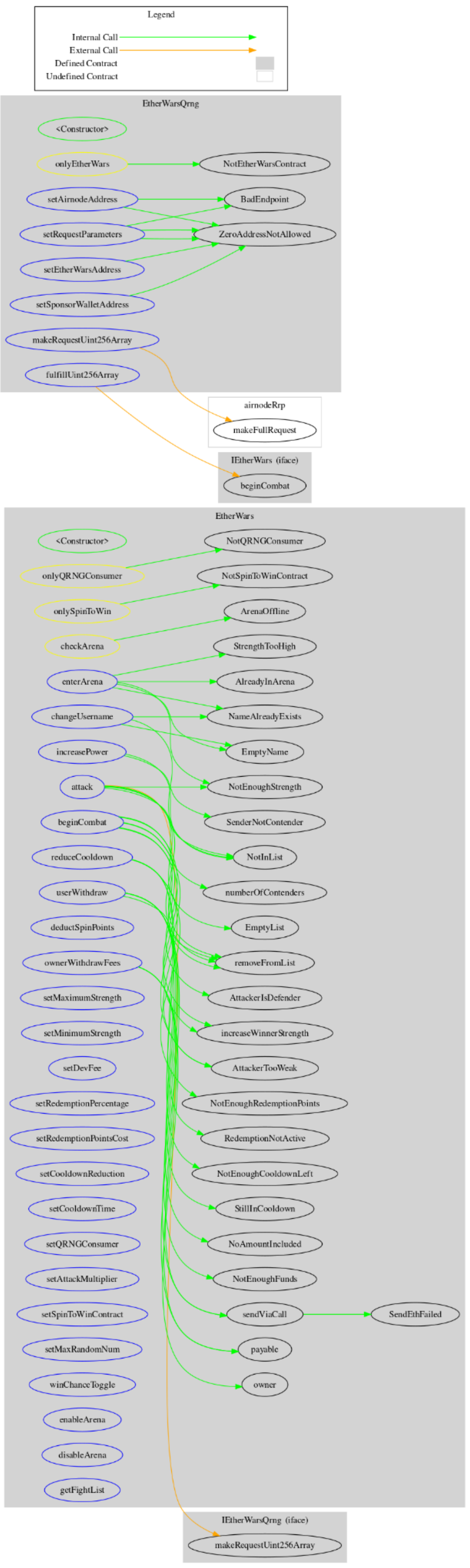
	setDevFee	External	✓	onlyOwner
	setRedemptionPercentage	External	✓	onlyOwner
	setRedemptionPointsCost	External	✓	onlyOwner
	setCooldownReduction	External	✓	onlyOwner
	setCooldownTime	External	✓	onlyOwner
	setQRNGConsumer	External	✓	onlyOwner
	setAttackMultiplier	External	✓	onlyOwner
	setSpinToWinContract	External	✓	onlyOwner
	setMaxRandomNum	External	✓	onlyOwner
	winChanceToggle	External	✓	onlyOwner
	enableArena	External	✓	onlyOwner
	disableArena	External	✓	onlyOwner
	getFightList	External		-
	numberOfContenders	Public		-
	removeFromList	Private	✓	
	increaseWinnerStrength	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
StrengthTooHigh
enterArena
AlreadyInArena
changeUsername
NameAlreadyExists
increasePower
EmptyName
attack
NotEnoughStrength
beginCombat
SenderNotContender
reduceCooldown
NotInList
userWithdraw
numberOfContenders
deductSpinPoints
EmptyList
ownerWithdrawFees
removeFromList
setMaximumStrength
AttackerIsDefender
setMinimumStrength
increaseWinnerStrength
setDevFee
AttackerTooWeak
setRedemptionPercentage
NotEnoughRedemptionPoints
setRedemptionPointsCost
RedemptionNotActive
setCooldownReduction
NotEnoughCooldownLeft
setCooldownTime
StillInCooldown
setQRNGConsumer
NoAmountIncluded
setAttackMultiplier
NotEnoughFunds
setSpinToWinContract
sendViaCall
SendEthFailed
setMaxRandomNum
payable
winChanceToggle
owner
enableArena
disableArena
getFightList
IEtherWarsQrng (iface)
makeRequestUint256Array

Summary

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

Disclaimer

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