SHERLOCK

Sherlock Security Review For Boost Protocol AA

Public contest prepared for: **Boost Protocol AA**
Lead Security Expert: **0xdeadbeef**
Date Audited: **September 11 - September 20, 2024**

Introduction

The Boost Protocol is a permissionless, trustless, and decentralized growth engine for protocol and application developers. It enables developers to bootstrap their projects by leveraging the power of community and the network effect.

Boosts are individual campaigns that are designed to incentivize and reward users for participation in a specific protocol or application. They are designed to be flexible and can be customized to fit the specific needs of the project.

Scope

Repository: rabbitholegg/boost-protocol

Branch: arthur/boost-4661-adapt-eventaction-to-work-with-contract-signatures

Audited Commit: 5711a9160c51abec01056c5a70501fd13f6ac489

Final Commit: 8b98bdff62ee0dea124ffbbbe141bfc0a1bc4a8d

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
5	2

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

ge6a
oxelmiguel
eLSeR17
sakshamguruji
iamnmt
ke1caM
Atharv
haxagon
0xbranded
0xdeadbeef
denzi_
0xNirix
ZanyBonzy
Pheonix
Japy69
ctf_sec
0xSecuri
dimulski
Galturok
4b
Ragnarok
Trooper
PranavGarg
durov
Okazaki
tinnohofficial
0xloophole
0x539.eth
pwning_dev
SyncCode2017
0xlookman
0rpse
y4y
IvanFitro
KupiaSec
0xDemon
TessKimy
0xbrivan
0xSolus
Hacek00
Aymen0909
Greese
SovaSlava
blutorque
frndz0ne
0xloscar01
scyron6
KungFuPanda
MrCrowNFT
ihtishamsudo
0xsome
AresAudits
Aycozzynfada
nikhilx0111
ParthMandale
tmotfl
DenTonylifer
MSK
MSaptarshi

Issue H-1: Unable to call some functions in the incentive contracts with onlyOwner modifier because of incorrect initialization leading to stuck funds

Source: https://github.com/sherlock-audit/2024-06-boost-aa-wallet-judging/issues/43

Found by

0xDemon, 0xNirix, 0xSecuri, 0xSolus, 0xbranded, 0xbrivan, 0xdeadbeef, 0xloscar01, Atharv, Aymen0909, Galturok, Greese, Hacek00, IvanFitro, Japy69, KupiaSec, PranavGarg, Ragnarok, SovaSlava, TessKimy, Trooper, ZanyBonzy, blutorque, ctf_sec, dimulski, durov, frndz0ne, ge6a, haxagon, iamnmt, ke1caM, oxelmiguel, sakshamguruji, scyron6, y4y

Summary

`BoostCore.sol` will always be set as the owner of Boost provided incentive contracts because the initializer is called here within _makeIncentives. Therefore any function using the onlyOwner modifier within the incentive contracts must be called by `BoostCore`. For example, there is no way to call drawRaffle or clawback from the BoostCore contract.

Root Cause

createBoost is called to create a new boost. Each incentive is initialized by the call to _makeIncentives. Within `_makeIncentives` the initializer is called for each incentive. The initializer function within each incentive contract sets the owner as msg.sender which would be the `BoostCore` contract.

Internal pre-conditions

1. Boost is created using the out of the box incentive contract as one of the incentives including: ERC20Incentive, CGDAIncentive, ERC20VariableIncentive, and ERC1155Incentive

External pre-conditions

No response

Attack Path

1. User calls `createBoost` to create a new Boost
2. They choose to use an out of the box incentive contract listed above
3. They are initialized with `BoostCore` as the owner

Impact

- No winner can be drawn for raffle contests through ERC20Incentive contract
- Any funds in the contract that need to be rescued cannot be retrieved through clawback

PoC

No response

Mitigation

Owner should be specified in the init payload by the user similarly to how its done for the budget contracts here

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/boostxyz/boost-protocol/pull/192

Issue H-2: IncentiveBits.setOrThrow() will revert, leading to a DoS

Source: https://github.com/sherlock-audit/2024-06-boost-aa-wallet-judging/issues/263

Found by

eLSeR17, ge6a, oxelmiguel

Summary

IncentiveBits.setOrThrow() will revert, leading to a DoS.

Vulnerability Detail

setOrThrow() expects each incentive from 0 to 7 to be used once per hash, reverting in case that for a given hash, an already used incentive is used again. However the mechanism that checks already used incentives does not work as expected: `alreadySet:=xor(1,shr(incentive,updatedStorageValue))`, reverting if incentiveIds are not used in increasing order.

The external call will come from BoostCore.claimIncentiveFor(), which calls SignedValidator.validate() and therefore setOrThrow(). The value of the incentiveId parameter used is arbitrary and valid as long as `uint256(validatorData.incentiveQuantity)<=incentiveId` is not fulfilled, which does not guarantee that calls will necessarily be in increasing order.

Example: Imagine setOrThrow() function is used with incentiveId = 5, in that case updatedStorageValue will be set to XOR (00000000, 00100000) = 00100000. Therefore, the resulting value for alreadySet is: alreadySet = XOR (1, shr(5, 00100000)) = XOR (00000001, 00000001) = 0 => Does NOT revert.

Now setOrThrow() function is called again for incentiveId = 2, so that updatedStorageValue will be: XOR (00100000, 00000100) = 00100100. Therefore, the new resulting value for alreadySet is: alreadySet = XOR (1, shr(2, 00100100)) = XOR (00000001, 00001001) = 00001000 => Reverts as alreadySet != 0

Impact

Claiming incentive for a given hash will be no longer possible, or fewer claims will be allowed depending on the last incentiveId used. This could be performed by accident by a normal user or on purpose by a malicious attacker to DoS and prevent other users from claiming from this hash.

Code Snippet

https://github.com/sherlock-audit/2024-06-boost-aa-wallet/blob/main/boost-protocol/packages/evm/contracts/validators/SignerValidator.sol#L126-L154

Tool used

Manual Review

Recommendation

For correctly comparing if the incentiveId index has been used, that bit must be totally isolated and XOR it with 1. For this, first shift left until we get 10000000 and then shift 7 times to right to get 1.

```
function setOrThrow(IncentiveMap storage bitmap, bytes32 hash, uint256 incentive)
 ↪  internal {
        bytes4 invalidSelector = BoostError.IncentiveToBig.selector;
        bytes4 claimedSelector = BoostError.IncentiveClaimed.selector;
        /// @solidity memory-safe-assembly
        assembly {
            if gt(incentive, 7) {
                // if the incentive is larger the 7 (the highest bit index)
                // we revert
                mstore(0, invalidSelector)
                mstore(4, incentive)
                revert(0x00, 0x24)
            }
            mstore(0x20, bitmap.slot)
            mstore(0x00, hash)
            let storageSlot := keccak256(0x00, 0x40)
            // toggle the value that was stored inline on stack with xor
            let updatedStorageValue := xor(sload(storageSlot), shl(incentive, 1))
            // isolate the toggled bit and see if it's been unset back to zero
-           let alreadySet := xor(1, shr(incentive, updatedStorageValue))
+          let alreadySet := xor(1, shr(7, shl(incentive - 1, updatedStorageValue)))
.
.
.
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/rabbitholegg/boost-protocol/pull/138

Issue M-1: Both block.prevrandao and block.timestamp are not reliably source of randonness

Source:
https://github.com/sherlock-audit/2024-06-boost-aa-wallet-judging/issues/106

The protocol has acknowledged this issue.

Found by

0x539.eth, 0xSecuri, 0xloophole, 4b, Atharv, Japy69, Okazaki, Pheonix, ctf_sec, denzi_, ge6a, haxagon, oxelmiguel, pwning_dev, sakshamguruji, tinnohofficial

Summary

Both block.prevrandao and block.timestamp are not reliably source of randonness

Vulnerability Detail

In the ERC20Incentive.sol,

```
function drawRaffle() external override onlyOwner {
    if (strategy != Strategy.RAFFLE) revert BoostError.Unauthorized();

    LibPRNG.PRNG memory _prng = LibPRNG.PRNG({state: block.prevrandao +
↪   block.timestamp});

    address winnerAddress = entries[_prng.next() % entries.length];

    asset.safeTransfer(winnerAddress, reward);
    emit Claimed(winnerAddress, abi.encodePacked(asset, winnerAddress, reward));
}
```

the code use block.prevrandao and block.timestamp as source of randoness to determine who is lucky to win the raffle.

However, both op code are not good source of randonness.

https://eips.ethereum.org/EIPS/eip-4399

> Security Considerations The PREVRANDAO (0x44) opcode in PoS Ethereum (based on the beacon chain RANDAO implementation) is a source of randomness with different properties to the randomness supplied by BLOCKHASH (0x40) or DIFFICULTY (0x44) opcodes in the PoW network.

> Biasability The beacon chain RANDAO implementation gives every block proposer 1 bit of influence power per slot. Proposer may deliberately refuse to propose a block on the opportunity cost of proposer and transaction fees to prevent beacon chain randomness (a RANDAO mix) from being updated in a particular slot.

Impact

Miner can manipulate the block.prevrandao and block.timestamp to let specific address win the raffle

Code Snippet

https://github.com/sherlock-audit/2024-06-boost-aa-wallet/blob/78930f2ed6570f30e356b5529bd4bcbe5194eb8b/boost-protocol/packages/evm/contracts/incentives/ERC20Incentive.sol#L137

Tool used

Manual Review

Recommendation

change randon generate method (can use chainlink VRF, etc...)

Issue M-2: Boost creator can collect all the fees by setting referralFee to 9_000 and give claimants his address as referrer_ address

Source:
https://github.com/sherlock-audit/2024-06-boost-aa-wallet-judging/issues/158

Found by

0rpse, 0xbranded, 0xdeadbeef, 0xlookman, Atharv, Galturok, Pheonix, PranavGarg, Ragnarok, SyncCode2017, Trooper, dimulski, durov, ge6a, iamnmt, ke1caM, oxelmiguel, sakshamguruji

Summary

The boost creator can set the value of referralFee to 9_000 when creating the boost. The `BoostCore::referralFee` (the base fee) is set to 1000 in line 70,

https://github.com/sherlock-audit/2024-06-boost-aa-wallet/blob/main/boost-protocol/packages/evm/contracts/BoostCore.sol#L70

and added to the boost creator input in line 122,

https://github.com/sherlock-audit/2024-06-boost-aa-wallet/blob/main/boost-protocol/packages/evm/contracts/BoostCore.sol#L122

This will make the `BoostCore::referralFee` to be 10_000 (equal to the `BoostCore::FEE_DENOMINATOR`) ensuring that 100% of the fees collected when claimants claim their incentives are sent to the referrer address. To get the fees, the boost creator just need to ensure claimants use his address as referrer_ address. The protocol will never receive any fee for this particular boost.

Root Cause

Maximum value for `BoostCore::referralFee` was not set, allowing boost creators to allocate unlimited fraction of the fees to the referrer.

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

The protocol will receive no fees as all the fees will continuously be sent to the referrer_ address.

PoC

Please copy the code below into BoostCore.t.sol and run the test.

```
uint64 public constant boostAdditionalReferralFee = 9_000; // additional 90%
uint256 public constant PRECISION = 10_000;
uint256 public constant BASE_FEE = 1_000; // 10%
bytes invalidCreateCalldata =
    LibZip.cdCompress(
        abi.encode(
            BoostCore.InitPayload({
                budget: budget,
                action: action,
                validator: BoostLib.Target({
                    isBase: true,
                    instance: address(0),
                    parameters: ""
                }),
                allowList: allowList,
                incentives: _makeIncentives(1),
                protocolFee: 500, // 5%
                referralFee: boostAdditionalReferralFee, // 90%
                maxParticipants: 10_000,
                owner: address(1)
            })
        )
    );

function testClaimIncentive_ReferralTakesAllFees_audit() public {
    uint256 claimFee = 0.000075 ether;
    // Create a Boost first
    boostCore.createBoost(invalidCreateCalldata);
```

```
    // Mint an ERC721 token to the claimant (this contract)
    uint256 tokenId = 1;
    mockERC721.mint{value: 0.1 ether}(address(this));
    mockERC721.mint{value: 0.1 ether}(address(this));
    mockERC721.mint{value: 0.1 ether}(address(this));

    // Prepare the data payload for validation
    bytes memory data = abi.encode(address(this), abi.encode(tokenId));
    address referralAddress = makeAddr("referral");
    address protocolFeeReceiver = boostCore.protocolFeeReceiver();
    uint256 initialProtocolFeeReceiverBalance = protocolFeeReceiver.balance;
    // Claim the incentive
    boostCore.claimIncentive{value: claimFee}(0, 0, referralAddress, data);

    uint256 actualReferrerBalance = referralAddress.balance;
    uint256 finalProtocolFeeReceiverBalance = protocolFeeReceiver.balance;
    // check referral balance
    assertEq(actualReferrerBalance, claimFee);
    // check protocol fee receiver balance
    assertEq(
        (finalProtocolFeeReceiverBalance -
            initialProtocolFeeReceiverBalance),
        0
    );
    // Check the claims
    BoostLib.Boost memory boost = boostCore.getBoost(0);
    ERC20Incentive _incentive = ERC20Incentive(
        address(boost.incentives[0])
    );
    assertEq(_incentive.claims(), 1);
}
```

Mitigation

Set a maximum value for `BoostCore::referralFee` and refactor `BoostCore::createBoost` as shown below.

```
+ uint64 public constant MAX_REFERRER_FEE = 5000; // should be any value below
 ↪  10_000
 function createBoost(bytes calldata data_)
        external
        canCreateBoost(msg.sender)
        nonReentrant
        returns (BoostLib.Boost memory)
    {
        InitPayload memory payload_ = abi.decode(data_.cdDecompress(),
 ↪  (InitPayload));
```

```
        // Validate the Budget
        _checkBudget(payload_.budget);

        // Initialize the Boost
        BoostLib.Boost storage boost = _boosts.push();
        boost.owner = payload_.owner;
        boost.budget = payload_.budget;
        boost.protocolFee = protocolFee + payload_.protocolFee;
        boost.referralFee = referralFee + payload_.referralFee;
+       require(boost.referralFee <= MAX_REFERRER_FEE, "referralFee is too high");
        boost.maxParticipants = payload_.maxParticipants;

        // Setup the Boost components
        boost.action = AAction(_makeTarget(type(AAction).interfaceId,
 ↪  payload_.action, true));
        boost.allowList = AAllowList(_makeTarget(type(AAllowList).interfaceId,
 ↪  payload_.allowList, true));
        boost.incentives = _makeIncentives(payload_.incentives, payload_.budget);
        boost.validator = AValidator(
            payload_.validator.instance == address(0)
                ? boost.action.supportsInterface(type(AValidator).interfaceId) ?
 ↪  address(boost.action) : address(0)
                : _makeTarget(type(AValidator).interfaceId, payload_.validator,
 ↪  true)
        );
        emit BoostCreated(
            _boosts.length - 1,
            boost.owner,
            address(boost.action),
            boost.incentives.length,
            address(boost.validator),
            address(boost.allowList),
            address(boost.budget)
        );
        return boost;
    }
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/boostxyz/boost-protocol/pull/197

Issue M-3: claimIncentiveFor Might Lead To Loss Of Funds For CGDA Incentive

Source:
https://github.com/sherlock-audit/2024-06-boost-aa-wallet-judging/issues/178

The protocol has acknowledged this issue.

Found by

iamnmt, ke1caM, sakshamguruji

Summary

The protocol has introduces a functionality to claim an incentive for other claimants , this would require data for the claim and according to the sponsor (asked in thread) -> `Signaturesareavailablepubliclybywayofapi` , so this way I can claim for someone else , it's a neat feature but for CGDA incentive can be disastrous.

Vulnerability Detail

1.) Alice completes an action and for this action the incentive was a CGDAIncentive , the off chain mechanism verifies that Alice has performed the action successfully and grants her the claim , the claim as mentioned `Signaturesareavailablepubliclybywayofapi`

2.) Alice has a valid claim now for the CGDAIncentive , but she wants to wait for some time to claim since CGDA is dependent on lastClaimTime and she wants to maximise her gains , she wants to wait for 5 more blocks.

https://github.com/sherlock-audit/2024-06-boost-aa-wallet/blob/main/boost-protocol/packages/evm/contracts/incentives/CGDAIncentive.sol#L124

3.) Bob comes and claims the incentive for Alice earlier ->

https://github.com/sherlock-audit/2024-06-boost-aa-wallet/blob/main/boost-protocol/packages/evm/contracts/BoostCore.sol#L164

He does it such that Alice would get lesser incentive due to `uint256timeSinceLastClaim=block.timestamp-cgdaParams.lastClaimTime;` being smaller than Alice intended and hence the rewards sent would be lesser

https://github.com/sherlock-audit/2024-06-boost-aa-wallet/blob/main/boost-protocol/packages/evm/contracts/incentives/CGDAIncentive.sol#L123-L130

4.) Alice lost her incentives , she wanted to claim after 5 blocks and make maximum gains , but Bob ruined her returns.

Impact

Alice will get way lesser incentives than intended due to Bob claiming on her behalf.

Code Snippet

https://github.com/sherlock-audit/2024-06-boost-aa-wallet/blob/main/boost-protocol/packages/evm/contracts/BoostCore.sol#L164

Tool used

Manual Review

Recommendation

Done let users cliam for other for such time dependent incentives.

Issue M-4: Budget allocation will break in case of a fee on transfer ERC 20 token

Source:
https://github.com/sherlock-audit/2024-06-boost-aa-wallet-judging/issues/325

The protocol has acknowledged this issue.

Found by

0xDemon, 0xbranded, 0xbrivan, 0xsome, 4b, AresAudits, Atharv, Aycozzynfada, DenTonylifer, Galturok, IvanFitro, Japy69, KungFuPanda, KupiaSec, MSK, MSaptarshi, MrCrowNFT, ParthMandale, Pheonix, TessKimy, dimulski, ge6a, haxagon, iamnmt, ihtishamsudo, nikhilx0111, oxelmiguel, sakshamguruji, tmotfl, y4y

Summary

Docs mention that the protocol should work with all kinds of weird tokens but a fee on transfer token won't be allocated to the budget since the allocate function in ManagedBudget.sol reverts when the balance of the asset is lesser than the amount mentioned in the payload.

Root Cause

: This check prevents the use of fee on transfer tokens since the allocated tokens actually transferred to the contract's balance will always be lesser than the payload amount owing to the fee component.

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

The protocol will not be able to use fee on transfer tokens which they clearly want to use according to the questionnaire they answered.

PoC

This is a mock ERC20 fee on transfer token used for the POC

```
contract FeeOnTransferMockERC20 is ERC20("MOCK","MOCK"){
    uint FEE = 10000;
    function mint(address to, uint256 amount) public {
        _balances[to] += amount;
    }

    function transfer(address to, uint256 amount) public override returns (bool) {
        require(amount > FEE);
        _balances[msg.sender] = _balances[msg.sender] - amount;
        _balances[to] = _balances[msg.sender] + amount - FEE;
        return true;
    }

    function safeTransferFrom(address from, address to, uint amount) public {
        transferFrom(from, to, amount);
    }
    function transferFrom(address from, address to, uint amount)  public override
↪   returns(bool){
        require(amount > FEE);
        _balances[from] = _balances[from] - amount;
        _balances[to] = _balances[from] + amount - FEE;
        return true;
    }

    function mintPayable(address to, uint256 amount) public payable {
        require(msg.value >= amount / 100, "MockERC20: gimme more money!");
        mint(to, amount);
    }
}
```

The test to be added to ManagedBudget.t.sol to replicate the results

```
function testFeeOnTransfer() public{
        //deploy a feeon transfer mock token and mint tokens to this address
        mockFeeOnTransferERC20 = new FeeOnTransferMockERC20();
        mockFeeOnTransferERC20.mint(address(this), 100 ether);

        managedBudget = ManagedBudget(payable(LibClone.clone(address(new
↪   ManagedBudget()))));
        managedBudget.initialize(
```

```
            abi.encode(
                ManagedBudget.InitPayload({owner: address(this), authorized: new
↪  address[](0), roles: new uint256[](0)})
            )
        );
        mockFeeOnTransferERC20.approve(address(managedBudget), 100 ether);
        bytes memory data = _makeFungibleTransfer(ABudget.AssetType.ERC20,
↪  address(mockFeeOnTransferERC20), address(this), 100 ether);
        vm.expectRevert(abi.encodeWithSelector(ABudget.InvalidAllocation.selector,
↪  address(mockFeeOnTransferERC20), uint256(100 ether)));
        managedBudget.allocate(data);

    }
```

The test passes which means the `managedBudget.allocate(data)` call reverts with an `InvalidAllocation` error

Mitigation

This one is tricky since there are 2 paths the sponsor can take:

1. Remove the support for fee on transfer tokens and mention this explicitly
2. Keep supporting fee on transfer tokens and remove the aforementioned check.

Issue M-5: The incentive contracts are not compatible with rebasing/deflationary/inflationary tokens

Source: https://github.com/sherlock-audit/2024-06-boost-aa-wallet-judging/issues/460

The protocol has acknowledged this issue.

Found by

0xNirix, 0xbranded, 0xdeadbeef, Atharv, ZanyBonzy, denzi_, ge6a, haxagon

Summary

The protocol wants to work with all kind of tokens including rebasing tokens. From weirdERC20 we can read more about Balance Modfications Outisde of Transfers (rebasing/airdrops) section which states

> Some tokens may make arbitrary balance modifications outside of transfers (e.g. Ampleforth style rebasing tokens, Compound style airdrops of governance tokens, mintable/burnable tokens).
>
> Some smart contract systems cache token balances (e.g. Balancer, Uniswap-V2), and arbitrary modifications to underlying balances can mean that the contract is operating with outdated information.

Vulnerability Detail

One such example of not supporting in the code is the `ERC20Incentive::clawback()` function

```
function clawback(bytes calldata data_) external override onlyOwner returns (bool) {
        ClawbackPayload memory claim_ = abi.decode(data_, (ClawbackPayload));
        (uint256 amount) = abi.decode(claim_.data, (uint256));

        if (strategy == Strategy.RAFFLE) {
            // Ensure the amount is the full reward and there are no raffle
↪   entries, then reset the limit
            if (amount != reward || claims > 0) revert
↪   BoostError.ClaimFailed(msg.sender, abi.encode(claim_));
            limit = 0;
        } else {
            // Ensure the amount is a multiple of the reward and reduce the max
↪   claims accordingly
```

```
            if (amount % reward != 0) revert BoostError.ClaimFailed(msg.sender,
↪  abi.encode(claim_));
            limit -= amount / reward;
        }
```

The variable `reward` is being used in these if conditions, reward is set during initialization of the contract. It is either set as the full amount for raffles or the amount of reward per person for pools.

Lets consider the raffle situation for this report.

In the `initialize()` function, suppose that the reward amount in the data is sent as `10e18`, this is set as reward for the raffle after confirming by checking the balance of the contract.

Now suppose after some time the balance has changed due to rebasing. The reward variable is still 10e18 but the actual balance of the contract is different.

In the `clawback()` function, the owner wants to withdraw the full amount of the raffle. If they provide the rebased balance of the contract, the function will revert due to the following if condition

```
if (amount != reward || claims > 0) revert BoostError.ClaimFailed(msg.sender,
↪  abi.encode(claim_));
```

If they provide 10e18 as amount which was the original amount and the current balance of the contract is lower then the following line will cause a revert

```
asset.safeTransfer(claim_.target, amount);
```

This is only one instance of an issue, these issues are present in the Incentive contracts which use ERC20s.

Similarly `ERC20Incentive::drawRaffle()` will also not work if the actual balance of the contract has changed to a lower amount.

Impact

The balances are outdated and will cause hindrances for all parties involved. Denial of Service when the balances rebase.

Code Snippet

ERC20VariableIncentive.sol

ERC20Incentive.sol

CGDAIncentive.sol

Tool used

Manual Review

Recommendation

Track the balances after each transfer in/out to keep updated data in the contracts.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.