HashEx
BLOCKCHAIN SECURITY

Poopooville

smart contracts
final audit report

January 2025

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Poopooville team to perform an audit of their smart contract. The audit was conducted between 07/01/2025 and 11/01/2025.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The SHA-1 of the audited files are:`093fc2e8effd38c1134b6db81fee96599ef07ce8` PoocashSale.sol

`199bb4518ae39e675cdc3058abf1176e6f59ef88` Rounds.sol

`0fe212773ec13a4e23190e25814d5208168c36a3` IRounds.sol

`2eb6f89c0a40e7194cca213c7aed5c15b9a7f101` PoocashClaim.sol

Update. The Poopooville team has responded to this report. SHA-1 hashes of the updated files are:

`17b25710651cbd8b6dc01189b385e0ac730e1991` PoocashSale.sol

`db5ec2bee2d9ca1f7c604dac85f401faa297eb62` Rounds.sol

`9647149e5065478b41ca37196c0811b5138fde85` PoocashClaim.sol

2.1 Summary

Project name	Poopooville
URL	https://poopooville.io
Platform	Ethereum, Binance Smart Chain
Language	Solidity
Centralization level	● High
Centralization risk	● High

2.2 Contracts

Name	Address
PoocashClaim.sol	
Rounds.sol	
PoocashSale.sol	
IRounds.sol	
All contracts in scope	

3. Project centralization risks

The token sale is fully centralized, i.e., user's purchased tokens are not minted, transferred, or vested directly, but an additional signature with is required. If such signature is not provided by the project owner, participants will not be able to obtain their purchased tokens or will be granted to claim different amount that expected.

C88CR35 Owner privileges

The owner controls the signer entity, who and only who can permit vesting creation.

The owner can pause claiming process.

C89CR36 Owner privileges

The owner can add and delete rounds.

The owner can update round start and end timestamps at any moment and without safety checks to ensure timeline.

The owner controls the OPERATOR_ROLE that is allowed to directly manipulate user balances and contributions.

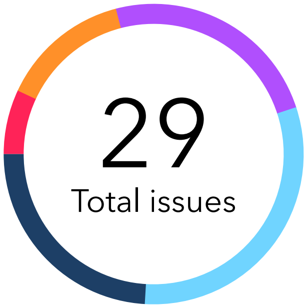
C8aCR37 Owner privileges

The owner can pause sale.

The owner can set an arbitrary address as Rounds contract.

The owner can update referral percentage from 0 to 25%.

4. Found issues



●	Critical	2 (7%)
●	High	4 (14%)
●	Medium	7 (24%)
●	Low	9 (31%)
●	Info	7 (24%)

C88. PoocashClaim.sol

ID	Severity	Title	Status
C88Ia8	Critical	Locked tokens	Resolved
C88Ic3	Critical	Unprotected claim function	Open
C88Ia9	High	No clear source of vested funds	Acknowledged
C88Ib5	High	Possibility of an early unlock for all users	Resolved
C88Ib2	Medium	Possibility of an underflow revert	Acknowledged
C88Ia7	Medium	Vesting schedule update	Acknowledged
C88Ia6	Low	Signature reuse	Acknowledged
C88Ib4	Low	Not enough events	Partially fixed
C88Ia5	Low	Gas optimizations	Partially fixed
C88Ib3	Info	Not used variable	Resolved

C89. Rounds.sol

ID	Severity	Title	Status
C89lba	Medium	Wrong _currentRound modification	Resolved
C89lbc	Low	Not enough checks for a new _min variable	Resolved
C89lbb	Low	Lack of error message	Resolved
C89laa	Low	Gas optimizations	Partially fixed
C89lb6	Info	Not used variables	Acknowledged
C89lb7	Info	Not used events	Resolved
C89lb8	Info	Timestamps are not checked	Acknowledged
C89lac	Info	Inconsistent event parameter	Resolved
C89lab	Info	Inconsistent comment	Resolved
C89lb9	Info	The number of rounds is not limited	Resolved

C8a. PoocashSale.sol

ID	Severity	Title	Status
C8alad	High	The owner can spend user allowance	Resolved
C8alb1	Medium	All payment tokens assumed to have same price	Acknowledged
C8alaf	Medium	Signature reuse	Partially fixed
C8alb0	Medium	Irreversible action	Resolved

C8albe	● Medium	Possibility of data loss	✓ Resolved
C8albd	● Low	Not enough events	✓ Resolved
C8albf	● Low	Lack of error message	✓ Acknowledged
C8alae	● Low	Gas optimizations	✓ Partially fixed

C8c. All contracts in scope

ID	Severity	Title	Status
C8cla4	● High	Lack of tests and documentation	✓ Acknowledged

5. Contracts

C88. PoocashClaim.sol

Overview

A vesting contract for a pre-defined ERC20 token address with multiple vesting schedules for different rounds. Vesting schedules support instant releasable amount and linear model with cliff.

Issues

C88Ia8 Locked tokens ● Critical ✓ Resolved

The `resetAccount()` function can be used to lock vested tokens of any user. The function is open for public use.

```
/**
 * Resets the vesting information for a given account and claim type.
 */
function resetAccount(address account, ClaimType claimType) external {
    if (account == address(0)) revert ZeroAddress();

    _vesting[account][claimType] = Vesting({
        totalAmount: 0,
        claimedAmount: 0
    });
}
```

Recommendation

Remove the function.

C88Ic3 Unprotected claim function ● Critical ⓘ Open

The `initialClaimScNoSig()` function was introduced in the code update. It allows anyone to claim an arbitrary amount of tokens without any additional safety checks.

```
/**
 * @dev Allows users to make their initial claim without requiring a signature.
 * @param totalAmount Total allocation for the user
 * @param claimType Type of claim (Seed, PreSale, etc.)
 */
function initialClaimScNoSig(
    uint256 totalAmount,
    ClaimType claimType
) external whenNotPaused {
    if (totalAmount == 0) revert InvalidAmount();

    // Cache vesting data from storage
    Vesting storage vesting = _vesting[msg.sender][claimType];

    if (vesting.totalAmount != 0) revert AlreadyClaimedForThisType();

    // Cache vesting schedule data
    VestingSchedule storage schedule = _vestingSchedules[claimType];
    uint256 initialRelease = (totalAmount * schedule.initialRelease) /
        10000;

    // Calculate the vesting start time and vested amount
    uint256 vestingStart = tgeLive + schedule.cliffDuration;
    uint256 vestedAmount = _calculateVestedAmount(
        totalAmount,
        initialRelease,
        vestingStart,
        schedule.vestingDuration
    );

    // Update vesting information directly in storage
    vesting.totalAmount = totalAmount;
    vesting.claimedAmount = vestedAmount;

    // Transfer the vested tokens to the user
    _token.safeTransfer(msg.sender, vestedAmount);
    emit InitialClaim(msg.sender, claimType, vestedAmount);
}
```

Recommendation

Include safety checks or remove the function.

C88la9 No clear source of vested funds ● High ✓ Acknowledged

The vesting is created by calling the `initialClaimScOld()` or `initialClaimSc()` with initial release amount to be transferred immediately, but there's no clear incoming transfer of user's total vested amount. If these tokens meant to be vested haven't been transferred beforehand, then the initial release amount would be unlocked from the shared balance, i.e., from vested funds of other users.

```
/**
 * @dev Allows users to make their initial claim.
 * @param totalAmount Total allocation for the user
 * @param deadline Deadline for the claim
 * @param claimType Type of claim (Seed, PreSale, etc.)
 * @param signature Signature from the backend
 */
function initialClaimScOld(
    uint256 totalAmount,
    uint256 deadline,
    ClaimType claimType,
    bytes memory signature
) external whenNotPaused {
    ...
    uint256 initialRelease = (totalAmount *
        _vestingSchedules[claimType].initialRelease) / 10000;

    _vesting[msg.sender][claimType] = Vesting({
        totalAmount: totalAmount,
        claimedAmount: initialRelease
    });
```

```
        _token.safeTransfer(msg.sender, initialRelease);

        emit InitialClaim(msg.sender, claimType, initialRelease);
    }
```

Recommendation

Ensure that `totalAmount` of vesting is received before creating vesting for the user.

C88Ib5 Possibility of an early unlock for all users ● High ✓ Resolved

If the owner of the contract won't set the `tgeLive` variable by calling the `setTgeLive()` function, then this variable will be equal to zero. In this case, all users will be able to claim all tokens immediately.

```
    /**
     * @dev Sets the TGE live timestamp. Must be in the future.
     * @param _tgeLive The new TGE live timestamp
     */
    function setTgeLive(uint256 _tgeLive) external onlyOwner {
        require(
            _tgeLive > block.timestamp,
            "TGE live date must be in the future"
        );
        tgeLive = _tgeLive;

        emit TgeLiveDate(_tgeLive);
    }

    /**
     * @dev Calculates the releasable amount of tokens based on vesting.
     */
    function _releasableAmount(
        uint256 totalAmount,
        uint256 claimedAmount,
        ClaimType claimType
    ) internal view returns (uint256) {
        ...
        uint256 vestingStart = tgeLive + schedule.cliffDuration;
        ...
        uint256 elapsedTime = block.timestamp - vestingStart;
        ...
    }
```

Recommendation

Add to the constructor of the contract this line:

```
tgeLive = block.timestamp;
```

C88Ib2 Possibility of an underflow revert ● Medium ✓ Acknowledged

In the function `_releasableAmount()` there is a possibility of an underflow on this line `return vestedAmount - claimedAmount;` after possible changes in vesting schedule variables in the `addClaimType()` function (it is possible only for the `SuperPoo` claim type).

If an owner of the contract decreases the number of tokens that a user can claim, then some users will face an error when they try to call the `getAvailableClaimTokens()` and the `claimTokens()` functions.

Recommendation

Replace the return statement with this logic:

```
if (vestedAmount >= claimedAmount) {
    return vestedAmount - claimedAmount;
} else {
    return 0;
}
```

C88Ia7 Vesting schedule update ● Medium ✓ Acknowledged

The `addClaimType()` can be used only to change the `ClaimType.SuperPoo` vesting schedule, and if new value of the `initialRelease` parameters is non-zero, then the function become unusable.

```
/**
 * @dev Adds a new claim type with its vesting schedule.
 */
function addClaimType(
    uint256 claimTypeId,
    uint256 initialRelease,
    uint256 cliffDuration,
    uint256 vestingDuration
) external onlyOwner {
    require(
        _vestingSchedules[ClaimType(claimTypeId)].initialRelease == 0,
        "Claim type already exists"
    );

    _vestingSchedules[ClaimType(claimTypeId)] = VestingSchedule({
        initialRelease: initialRelease,
        cliffDuration: cliffDuration,
        vestingDuration: vestingDuration
    });

    emit ClaimTypeAdded(
        claimTypeId,
        initialRelease,
        cliffDuration,
        vestingDuration
    );
}
```

Recommendation

Remove the function or update its logic.

C88Ia6 Signature reuse ● Low ✓ Acknowledged

A signature from backend is required for vesting creation. Signed data includes vesting data, deadline, verification contract address, but doesn't include chain-specific parameters, such as `chainId`. This allows possible re-use of signature in different chain.

```
        bytes32 hash = keccak256(
            abi.encodePacked(
                address(this),
                msg.sender,
                totalAmount,
                deadline,
                claimType
            )
        );
        bytes32 message = ECDSA.toEthSignedMessageHash(hash);

        if (ECDSA.recover(message, signature) != _claimSigner)
            revert InvalidSignature();
```

Recommendation

Consider securing the signature by including chain parameter.

C88Ib4 Not enough events

● Low ✓ Partially fixed

Functions `resetAccount()`, `withdrawETH()` and the constructor of the contract doesn't have any events.

Update

The constructor section still lacks for events. The severity of the issue was decreased.

C88Ia5 Gas optimizations

● Low ✓ Partially fixed

1. Multiple reads from storage in the `initialClaimSc()` function: `schedule.initialRelease`, `schedule.cliffDuration` variables.

2. Multiple reads from storage in the `claimTokens()` function: `vesting.totalAmount`, `vesting.claimedAmount`, `tgeLive` variables.

3. Multiple reads from storage in the `_releasableAmount()` function: `schedule.vestingDuration` variable.

4. Multiple reads from storage in the `withdrawETH()` function: `owner` address.

5. Multiple reads from storage in the `setClaimSigner()` function: `_claimSigner` variable.

6. The ECDSA library has been optimized after the 4.9 release, imported version is 4.2.

C88Ib3 Not used variable ● Info ✓ Resolved

The variable `_nonces` isn't used anywhere.

C89. Rounds.sol

Overview

A config contract to store all rounds of sale as weel as the current round info. The Rounds contract also regulates user's limits and stores sale participants data.

Issues

C89Iba Wrong _currentRound modification ● Medium ✓ Resolved

In the `deleteRound()` function the `_currentRound` variable is modified:

```
if (_currentRound >= _rounds.length) {
    _currentRound = _rounds.length - 1;
}
```

But this is an error for the case when `_currentRound < _rounds.length` and `_currentRound >= index_`. For example, there were 5 rounds, and the `_currentRound` variable has a value of 3, and in case we delete the 2 index, then the actual index of the current round will be 2, but the `_currentRound` variable still has a value of 3, which is wrong.

Recommendation

Replace the original if with this one:

```
if (_currentRound >= index_) {
    _currentRound -= 1;
}
```

C89lbc Not enough checks for a new _min variable ● Low ✓ Resolved

In the function `setMin()` the argument `amount_` isn't checked against the `MIN` global constant.

C89lbb Lack of error message ● Low ✓ Resolved

In functions `updateRoundPrice()`, `updateRoundSupply()`, `updateRoundStartTime()` and `updateRoundEndTime()` there is no custom error for the case when the `index_` argument is `>= _rounds.length`.

Should add the same check that exists in the `deleteRound()` function (error `ErrRoundUndefined`).

C89laa Gas optimizations ● Low ✓ Partially fixed

1. Multiple reads from storage in the `deleteRound()` function: `_rounds.length` variable.

2. Multiple reads from storage in the `openRound()` function: `_currentRound` variable.

3. Unnecessary reads from storage in the `setMax()` function: `_max` variable.

4. Unnecessary reads from storage in the `setMin()` function: `_min` variable.

5. Unnecessary reads from storage in the `limitOf()` function: `_authLimit` variable.

6. Multiple reads from storage in the `maxLimitOf()` function: `_max` variable.

7. Multiple reads from storage in the `getPrice()` function: `_currentRound` variable.

8. Ineffective removing element from the `_rounds[]` array may exceed block gas limit.

9. Multiple reads from storage in the `deleteRound()` function: `_currentRound` variable.

C89Ib6 Not used variables ● Info ✓ Acknowledged

The structure `Round` has fields `startTime` and `endTime` that are not used anywhere.

C89Ib7 Not used events ● Info ✓ Resolved

There are events `Erc20Recovered` and `CoinRecovered` that are not used anywhere.

C89Ib8 Timestamps are not checked ● Info ✓ Acknowledged

In the function `setRound()` there are no checks for the `startTime_` and the `endTime_` arguments, they can be of any value.

C89Iac Inconsistent event parameter ● Info ✓ Resolved

The `AuthUserUpdated` event contains an indexed user address, but the `AuthBatchUpdated` event is emitted without indexed parameters.

```
event AuthUserUpdated(address indexed user, bool value);
event AuthBatchUpdated(address[] users, bool[] values);
```

C89Iab Inconsistent comment ● Info ✓ Resolved

The `balanceOf()` function contains commented out modifier `onlyRole(DEFAULT_ADMIN_ROLE)`.

C89Ib9 The number of rounds is not limited ● Info ✓ Resolved

Using the `setRound()` function the admin of the contract can add any amount of new rounds. But because of this, the function `deleteRound()` can be blocked because of it. If there are too many rounds in the contract, the call to this function may cost more than there is available gas in a block and a transaction will fail.

Recommendation

Limit the number of rounds that can be added to the contract.

C8a. PoocashSale.sol

Overview

A sale contract to be coupled with the Rounds contract, allowing users to participate in token sale with different prices for different rounds of sale. All purchases must either contain an external signature from the backend or be performed by the owner himself. Payment tokens are meant to be stable tokens only.

Issues

C8aIad The owner can spend user allowance ● High ✓ Resolved

The contract owner has an exclusive access to the `buyFor()` function that can be used to spend user's allowance in any ERC20 token by force the user to participate in sale. The referral address and the price are controlled by the contract owner, meaning that he can abuse the `buyFor()` function to steal user's approved funds.

Recommendation

The buyFor() function must receive payment from the caller.

C8alb1 All payment tokens assumed to have same price

● Medium ✓ Acknowledged

The payment can be made in any ERC20 tokens from the whitelist. The price of these tokens is returned by IRounds.getPrice() function, which doesn't receive address of payment token in parameters. Only stable coins are meant to be whitelisted as payment tokens. Any significant discrepancy in stable coins prices may result in unexpected sale results.

```
    function _getSold(
        address token_,
        uint256 amount_
    ) internal view returns (uint256) {
        uint8 decimals = IERC20Metadata(token_).decimals();
        return
            ((amount_ * 10 ** DECIMALS * UNITS) / 10 ** decimals) /
            _rounds.getPrice();
    }
```

Recommendation

Use separate price feeds for supported payment tokens.

C8alaf Signature reuse

● Medium ✓ Partially fixed

A signature from backend is required for calling the buyS() function. Signed data includes payment data, and deadline, but doesn't include verification contract address and chain-specific parameters, such as chainId. This allows possible re-use of signature in different contract or chain.

```
    function getHash(
        address sender,
        uint256 time
    ) public pure returns (bytes32) {
        return keccak256(abi.encodePacked(address(sender), uint256(time)));
    }
```

Recommendation

Consider securing the signature by including chain parameter and address of the sale contract. Should use the EIP-712 standard.

Update

Address of the sale contract was included into the signature, but chain specific parameters were not. The contract should either be deployed to other chains to different addresses or different signer should be used.

C8alb0 Irreversible action

● Medium ✓ Resolved

The `updateTokens()` governance function is irreversible, i.e., mistakenly added payment tokens can't be removed.

```
    function updateTokens(address[] memory newTokens) external onlyOwner {
        require(
            newTokens.length > 0 && newTokens.length <= 2,
            "Invalid token array size"
        );

        for (uint256 i = 0; i < newTokens.length; i++) {
            require(newTokens[i] != address(0), "Token address is zero");
            _tokens[newTokens[i]] = Token({defined: true, total: 0});
        }
    }
```

Recommendation

Consider moving initialization to the constructor section.

C8aIbe Possibility of data loss ● Medium ✓ Resolved

In the updateTokens() function there is no check that a new token doesn't exist. In case, the new token exists, the field total will be overwritten, and previous data will be lost.

```
function updateTokens(address[] memory newTokens) external onlyOwner {
    require(
        newTokens.length > 0 && newTokens.length <= 2,
        "Invalid token array size"
    );

    for (uint256 i = 0; i < newTokens.length; i++) {
        require(newTokens[i] != address(0), "Token address is zero");
        _tokens[newTokens[i]] = Token({defined: true, total: 0});
    }
}
```

Recommendation

Consider to update only the defined filed of the Token structure.

C8aIbd Not enough events ● Low ✓ Resolved

Functions setWhitelistSigner(), updateTokens(), updateRounds() , and the constructor of the contract don't have events.

C8aIbf Lack of error message ● Low ✓ Acknowledged

In the _getSold() function there is no custom error for the case when the getPrice() function of the Rounds contract returns zero. In this case, the contract will fail without a message.

```
    function _getSold(
        address token_,
        uint256 amount_
    ) internal view returns (uint256) {
        uint8 decimals = IERC20Metadata(token_).decimals();
        return
            ((amount_ * 10 ** DECIMALS * UNITS) / 10 ** decimals) /
            _rounds.getPrice();
    }
```

C8alae Gas optimizations ● Low ✓ Partially fixed

1. Multiple external calls for `token_.decimals()` in the `_buy()` internal function: first call is direct, second call is in the `_getSold()` function.

2. Unnecessary duplicated calculations in the `_getSold()` function: the `amount_ * UNITS / 10 ** decimals` is already calculated as funds in the `_buy()` function.

3. Multiple hash calculation in the `buyS()` function: the `getHash()` function is called directly as well as in the `isSenderWhitelisted` modifier.

4. The ECDSA library has been optimized after the 4.9 release, imported version is 4.2.

C8b. IRounds.sol

Overview

An interface for the Rounds contract.

C8c. All contracts in scope

Issues

C8cIa4 Lack of tests and documentation ● High ✓ Acknowledged

The project was provided without any tests and documentation. We urgently recommend increasing test coverage. We also suggest providing the documentation section and enrich in-code descriptions using the NatSpec Format.

6. Conclusion

2 critical, 4 high, 7 medium, 9 low severity issues were found during the audit. 1 critical, 2 high, 3 medium, 3 low issues were resolved in the update. The reviewed contracts are highly dependent on the owner's account. See the centralization risks chapter.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. Issue status description

✓ **Resolved.** The issue has been completely fixed.

✓ **Partially fixed.** Parts of the issue have been fixed but the issue is not completely resolved.

✓ **Acknowledged.** The team has been notified of the issue, no action has been taken.

? **Open.** The issue remains unresolved.

Appendix C. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

Appendix D. Centralization risks classification

Centralization level

- 🔴 **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- 🟠 **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- 🟢 **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

Centralization risk

- 🔴 **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- 🟠 **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- 🟢 **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

✉ contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx
BLOCKCHAIN SECURITY