ADAM

smart contracts
final audit report

November 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer

2. Overview

HashEx was commissioned by the Adam team to perform an audit of their smart contract. The audit was conducted between 15/08/2022 and 29/08/2022.

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 audited contracts are designed to be deployed with proxies. Also, the owner of the contracts is EOA and he can change the implementation of the contracts at any moment. Users have no choice but to trust the owners, who can update the contracts at their will at any time.

The code is available at the GitHub repository @adam-vault/adam-contract-core after the commit 26f17ee.

Update: the Adam team has responded to this report. The updated code is located in the same GitHub repository after the c9fe245 commit.

2.1 Summary

Project name	ADAM
URL	https://adamvault.com
Platform	Ethereum

Language	Solidity

2.2 Contracts

Name	Address
Multiple contracts	
BudgetApprovalExecutee	
CommonBudgetApproval	
PriceResolver	
UniswapSwapper	
Base64	
BytesLib	
Concat	
Constant	
DSMath	
InterfaceChecker	
RevertMsg	
ToString	
TransferERC20BudgetApproval	`0x9e8630aFb3a5c7E85FcA23F135CC55F537f5f2f1`
TransferERC721BudgetApproval	`0xDAFE249B07a4e6342cDc82A225F023C18978429D`

TransferLiquidERC20BudgetApproval	`0x1a08510414881409b9Fd6D0073AA9886087f2D65`
UniswapBudgetApproval	`0xF0876A5e2A860Ce1D205FA7c2e8a5C9a6795f657`
Team	`0x4d562518a3e9b2eAFbd2a46376F09e032dacFeBd`
Membership	`0xfbc0d2Df1300afA0C2c4b7a44540a538fa322Ed7`
MemberToken	`0x825da39C630Fe4d2F4A603E5Eb12905049fE7353`
LiquidPool	`0x72d251D6cb0d3410B7783343ECc1D70787578a30`
GovernFactory	`0xa3Bc6600001cFd92b033f9EdE9C2821f1BA6462a`
Govern	`0xeEd63D279cc9AaA2217F2c0e0573563b6DE169bD`
Dao	`0x48286cbd1aA824c80aD6cBBA3b17d0367709F98E`
Adam	`0x8063Bb9687B22789aebf041c211Df66240e11f04`

3. Found issues

84
Total issues

●	High	3 (4%)
●	Medium	12 (14%)
●	Low	62 (74%)
●	Info	7 (8%)

C1. Multiple contracts

ID	Severity	Title	Status
C1-01	● Low	Gas optimisation	✓ Acknowledged
C1-02	● Low	Storage gaps	✓ Resolved
C1-03	● Low	_disableInitializers() function	✓ Resolved
C1-04	● Info	Lack of documentation (NatSpec)	✓ Acknowledged
C1-05	● Info	Floating Pragma	✓ Resolved
C1-06	● Info	The owner can upgrade the contract	✓ Acknowledged
C1-07	● Info	Redundant import	✓ Resolved

C2. BudgetApprovalExecutee

ID	Severity	Title	Status
C2-01	● Low	Few events	✓ Resolved
C2-02	● Low	Missing function	✓ Resolved
C2-03	● Low	Gas optimization	✓ Resolved
C2-04	● Info	Misuse risk	✓ Partially fixed

C3. CommonBudgetApproval

ID	Severity	Title	Status
C3-01	● Medium	Upgradability	✓ Resolved
C3-02	● Low	Unused imports	✓ Resolved
C3-03	● Low	Gas optimization	✓ Resolved
C3-04	● Low	Input validation	✓ Resolved

C4. PriceResolver

ID	Severity	Title	Status
C4-01	● High	Unification of the return value	✓ Resolved
C4-02	● Medium	Validating return value	✓ Resolved
C4-03	● Low	Unused imports	✓ Resolved
C4-04	● Low	Gas optimization	✓ Resolved

C5. UniswapSwapper

ID	Severity	Title	Status
C5-01	● High	Wrong outputs	✓ Resolved
C5-02	● Low	Gas optimization	✓ Resolved

C9. Constant

ID	Severity	Title	Status
C9-01	● Info	Testnet addresses	✓ Acknowledged

C11. InterfaceChecker

ID	Severity	Title	Status
C11-01	● Low	ERC20 checking	✓ Resolved

C14. TransferERC20BudgetApproval

ID	Severity	Title	Status
C14-01	● Medium	Unworking limitations	✓ Resolved
C14-02	● Low	Unused imports	✓ Resolved
C14-03	● Low	Gas optimization	✓ Resolved
C14-04	● Low	Few events	✓ Resolved

C15. TransferERC721BudgetApproval

ID	Severity	Title	Status
C15-01	● Low	Gas optimization	✓ Resolved
C15-02	● Low	Unused imports	✓ Resolved
C15-03	● Low	Few events	✓ Resolved
C15-04	● Low	Function that returns length	✓ Resolved

C16. TransferLiquidERC20BudgetApproval

ID	Severity	Title	Status
C16-01	● Medium	Limitations aren't working	✓ Resolved
C16-02	● Low	Unused imports	✓ Resolved
C16-03	● Low	Few events	✓ Resolved
C16-04	● Low	Function that returns length	✓ Resolved
C16-05	● Low	Gas optimization	✓ Resolved

C17. UniswapBudgetApproval

ID	Severity	Title	Status
C17-01	● Medium	Broken returns values from UniswapSwapper are used	✓ Resolved
C17-02	● Low	Unused imports	✓ Resolved

C17-03	Low	Gas optimization	Resolved

C18. Team

ID	Severity	Title	Status
C18-01	Low	Few events	Resolved
C18-02	Low	Gas optimization	Resolved
C18-03	Low	Input arguments validation	Resolved

C19. Membership

ID	Severity	Title	Status
C19-01	Medium	Adding identical members	Resolved
C19-02	Medium	Initializing of EIP712	Resolved
C19-03	Low	Gas optimization	Resolved
C19-04	Low	Improperly overriding	Resolved
C19-05	Low	Unused modifier	Resolved

C20. MemberToken

ID	Severity	Title	Status
C20-01	High	Changing delegation of users	Resolved
C20-02	Medium	Initializing of ERC20Permit	Resolved

C20-03	● Low	initialize() lacks validation of input parameters	✓ Resolved
C20-04	● Low	Gas optimization	✓ Resolved
C20-05	● Low	Unused modifier	✓ Resolved
C20-06	● Info	Incorrect error message in require statement	✓ Resolved

C21. LiquidPool

ID	Severity	Title	Status
C21-01	● Low	Transfers of ERC20 tokens	✓ Resolved
C21-02	● Low	Unused imports	✓ Resolved
C21-03	● Low	Redundant computations	✓ Resolved
C21-04	● Low	Length of an array	✓ Resolved
C21-05	● Low	Variable shadowing	✓ Resolved
C21-06	● Low	Native token transfer	✓ Resolved
C21-07	● Low	Gas optimization	✓ Resolved

C22. GovernFactory

ID	Severity	Title	Status
C22-01	● Medium	Function {addVoteToken}	✓ Resolved
C22-02	● Low	Validation of the arguments	✓ Resolved

C23. Govern

ID	Severity	Title	Status
C23-01	● Medium	Votes counting for the quorum	✓ Resolved
C23-02	● Medium	Use of weights	✓ Resolved
C23-03	● Medium	Problem of {addVoteToken} function	✓ Resolved
C23-04	● Low	Wrong return value	✓ Resolved
C23-05	● Low	Unused library import	✓ Resolved
C23-06	● Low	Validation of the arguments	✓ Resolved
C23-07	● Low	Length of arrays	✓ Resolved
C23-08	● Low	Gas optimization	✓ Resolved
C23-09	● Low	Unused imports	✓ Resolved

C24. Dao

ID	Severity	Title	Status
C24-01	● Low	Gas optimization	✓ Resolved
C24-02	● Low	Validation of the arguments	✓ Resolved
C24-03	● Low	Length of an array	✓ Resolved
C24-04	● Low	Wrong error message	✓ Resolved
C24-05	● Low	Unused variable	✓ Resolved
C24-06	● Low	Unused imports	✓ Resolved

C24-07	● Low	Unreachable function	✓ Resolved
C24-08	● Low	Few events	✓ Resolved

C25. Adam

ID	Severity	Title	Status
C25-01	● Low	Few events	✓ Resolved
C25-02	● Low	Functions lacks validation of input parameters	✓ Resolved
C25-03	● Low	Unused variable	✓ Resolved
C25-04	● Low	Removing from the whitelist	✓ Resolved
C25-05	● Low	Gas optimization	✓ Resolved

4. Contracts

C1. Multiple contracts

Overview

The following issues are related to multiple contracts.

Issues

C1-01 Gas optimisation ● Low ✓ Acknowledged

Instead of `require()` statements custom errors can be used, as they are more gas-efficient. Information about them can be found here.

Team response

This does not result in any additional security risks. We will update the custom error handling in the upcoming release as a backlog item.

C1-02 Storage gaps ● Low ✓ Resolved

All contracts that inherit the UUPSUpgradeable contract should implement storage gaps.

C1-03 _disableInitializers() function ● Low ✓ Resolved

All contracts that inherit the Initializable contract should implement a constructor that calls the `_disableInitializers()` function.

C1-04 Lack of documentation (NatSpec) ● Info ✓ Acknowledged

We recommend writing documentation using NatSpec Format. This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

Team response

This does not result in any additional security risks. We will add NatSpec documentation in the upcoming release as a backlog item.

C1-05 Floating Pragma ● Info ✓ Resolved

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

C1-06 The owner can upgrade the contract ● Info ✓ Acknowledged

`Adam`, `Govern`, `GovernFactory` and `Team` contracts have the `_authorizeUpgrade()` function with the `onlyOwner` modifier, which means that the owner can upgrade the contract at any time. It is recommended to make a timelock owner of these contracts. This will give users time to think and make decisions.

Team response

We intend to keep it as is until the application becomes stable.

C1-07 Redundant import ● Info ✓ Resolved

Importing `hardhat/console.sol` is redundant. It is needed only for testing and not for production.

C2. BudgetApprovalExecutee

Overview

A contract serving to perform `call()` from BudgetApproval after the budget approval is approved.

Issues

C2-01 Few events ● Low ✓ Resolved

Several functions from the contract lack events:

1. `executeByBudgetApproval()`
2. `createBudgetApprovals()`

C2-02 Missing function ● Low ✓ Resolved

There should be a `__BudgetApprovalExecutee_init()` function that sets a global variable `team`.

C2-03 Gas optimization ● Low ✓ Resolved

The `createBudgetApprovals()` function can be declared as external to save gas.

C2-04 Misuse risk ● Info ✓ Partially fixed

The contract is not declared as abstract, please notice that bare contract implementation without proper checks may lead to drain of funds. The functionality of BudgetApprovalExecutee should be documented and the developers should be notified of misuse risks in order to implement proper checks.

Team response

This does not result in any security risks currently. We will update the documentation in the upcoming release as backlog item.

C3. CommonBudgetApproval

Overview

An abstract contract. When inherited, it carries the skeleton functionality of creating budget approval transactions and their execution, while the `_execute()` method implementation is done by the descendant.

Issues

C3-01 Upgradability ● Medium ✓ Resolved

The function `_authorizeUpgrade()` has the modifier `initializer()`. Because of that, an upgrade to the contract can't be made.

C3-02 Unused imports ● Low ✓ Resolved

The functionality of the `IERC20`, `RevertMsg`, `IMembership` imports is not used in this contract.

C3-03 Gas optimization ● Low ✓ Resolved

The function `executeParams()` can be declared as external to save gas.

Functions `statusOf()`, `approvedCountOf()`, and `deadlineOf()` are redundant.

Struct `Transaction` can be reordered to be more gas efficient. It can be executed like this:

```
struct Transaction {
    uint256 id;
    bytes[] data;
    Status status;
    uint32 deadline;
    bool isExist;
    uint208 approvedCount;
    mapping(address => bool) approved;
}
```

Global variable `dao` is unused.

In the function `executeTransaction()` global variables `transactions[id].data` (its length and elements inside this array), and `allowUnlimitedUsageCount` are read multiple times. Also, the global variable `usageCount` is read after writing.

C3-04 Input validation ● Low ✓ Resolved

Functions `approveTransaction()` and `revokeTransaction()` can be called on a nonvalid id.

C4. PriceResolver

Overview

The contract has the functionality of querying an oracle in order to resolve assets' prices. It may also compute the derived price from base and quote prices, scaling decimals between them.

Issues

C4-01 Unification of the return value ● High ✓ Resolved

In the function `assetBaseCurrencyPrice()` the decimals of the return value may differ for different assets (in case `baseCurrency==ETH` or `asset==ETH`).

This implementation of `ethAssetPrice()` and `assetEthPrice()` will improve it:

```
function ethAssetPrice(address asset, uint256 ethAmount) public view returns (uint256) {
    if (asset == Denominations.ETH || asset == Constant.WETH_ADDRESS)
        return ethAmount;

    (, int price,,,) =
FeedRegistryInterface(Constant.FEED_REGISTRY).latestRoundData(asset, Denominations.ETH);

    uint256 priceDecimals = FeedRegistryInterface(Constant.FEED_REGISTRY).decimals(asset,
Denominations.ETH);
    price = scalePrice(price, priceDecimals, 18 /* ETH decimals */);

    if (price > 0) {
        return ethAmount * (10 ** IERC20Metadata(asset).decimals()) / uint256(price);
    }

    return 0;
}

function assetEthPrice(address asset, uint256 amount) public view returns (uint256) {
    if (asset == Denominations.ETH || asset == Constant.WETH_ADDRESS)
        return amount;

    (, int price,,,) =
FeedRegistryInterface(Constant.FEED_REGISTRY).latestRoundData(asset, Denominations.ETH);

    uint256 baseDecimals = baseCurrencyDecimals();
    uint256 priceDecimals = FeedRegistryInterface(Constant.FEED_REGISTRY).decimals(asset,
Denominations.ETH);
    price = scalePrice(price, priceDecimals, baseDecimals);

    if (price > 0) {
        return uint256(price) * amount / 10 ** IERC20Metadata(asset).decimals();
    }

    return 0;
}
```

C4-02 Validating return value ● Medium ✓ Resolved

The timestamp of oracle's return values doesn’t check. It is a better practice to check it to make sure that the contract doesn't use expired values. This value is in the return values of `latestRoundData()` function.

Recommendation

Check the timestamp of the oracle's return values.

C4-03 Unused imports ● Low ✓ Resolved

The functionality of the `ERC1155Upgradeable`, `UUPSUpgradeable`, `Counters`, `Strings` imports is not used in this contract.

C4-04 Gas optimization ● Low ✓ Resolved

In the function `assetBaseCurrencyPrice()` global variable `baseCurrency` is read multiple times.

C5. UniswapSwapper

Overview

A contract containing encoding and decoding functions to interact with UniSwap V2 and V3 Routers.

Issues

C5-01 Wrong outputs ● High ✓ Resolved

In the functions `_decodeUniswapRouter()` (both) return values `tokenIn`, `tokenOut`, `amountIn`, and `amountOut` will be calculated incorrectly in some cases. For example, if multicall there are two calls to swap

1. `token1` to `token2` with `amountIn1` and `amountOut1`
2. `token3` to `token4` with `amountIn2` and `amountOut2`

In this case `tokenIn` will be `token3`, `tokenOut` will be `token4`, `amountIn` will be `amountIn1+amountIn2`, `amountOut` will be `amountOut1+amountOut2`. Obvious that these values are incorrect.

Recommendation

Change the return types from

```
returns(address tokenIn, address tokenOut, uint256 amountIn, uint256 amountOut, bool
estimatedIn, bool estimatedOut)
```

to this

```
returns(address[] memory tokensIn, address[] memory tokensOut, uint256[] memory amountsIn,
uint256[] memory amountsOut, bool[] memory estimatedIns, bool[] memory estimatedOuts)
```

C5-02 Gas optimization ● Low ✓ Resolved

The `decodeUniswapDataBeforeSwap()`, `decodeUniswapDataAfterSwap()`, `exactOutputSingle()`, `exactInputSingle()`, `exactOutput()`, `exactInput()`, `swapTokensForExactTokens()`, `swapExactTokensForTokens()` functions can be declared as external to save gas.

C6. Base64

Overview

A library, containing `base64()` bytes to string encoding function.

C7. BytesLib

Overview

A library containing bytes helper functions.

C8. Concat

Overview

A library containing a helper function `concat()`. Given an input of two strings, it returns one concatenated.

C9. Constant

Overview

A collection of predefined values: `WETH_ADDRESS`, `UNISWAP_ROUTER` address, chainlink aggregator `FEED_REGISTRY` address and `BLOCK_NUMBER_IN_SECOND` number.

Issues

C9-01 Testnet addresses ● Info ✓ Acknowledged

Addresses that are provided in this file are in the rinkeby testnet. Before deploying contracts into the mainntet these addresses should be replaced with mainnet values.

Team response

These addresses will be replaced with mainnet values when deploying mainnet.

C10. DSMath

Overview

A library conaining miscellaneous math functions.

C11. InterfaceChecker

Overview

A library containing helper functions, checking whether the contracts support the interface of ERC20, ERC721, and ERC1155.

Issues

C11-01 ERC20 checking ● Low ✓ Resolved

For ERC20 there is an inaccurate checking because ERC721 has a `balanceOf()` function too.

Check for ERC1155

ERC1155 also implements balanceOf method. Add the check `if (isERC1155(check)) { return false; }` too.

C12. RevertMsg

Overview

A library containing the helper ToString function for byte values returned by `.call()`

C13. ToString

Overview

A library converting bytes, addresses and uint into strings.

C14. TransferERC20BudgetApproval

Overview

Derived from CommonBudgetApproval, TransferERC20BudgetApproval is one of DAO's budget approvals with a single token specified.

Budget approvals are created and approved. After that, they may be executed. Execution of such approval is a transfer of the ERC20 token.

Issues

C14-01 Unworking limitations ● Medium ✓ Resolved

Limitation of the `checkAmountPercentageValid()` function doesn't work as it should because this is the limitation only for one transfer, but in one transaction in the `CommonBudgetApproval` contract, there can be multiple transfers.

Recommendation

The contract can store the transfer amount for the block. It will allow extending these restrictions to the whole transaction, not only for the function call.

C14-02 Unused imports ● Low ✓ Resolved

The functionality of the `PriceResolver`, `IDao`, `IAdam` imports is not used in this contract.

C14-03 Gas optimization ● Low ✓ Resolved

The `initialize()`, `executeParams()` functions can be declared as external to save gas.

In the `_execute()` function global variables `allowAnyAmount` and `totalAmount` can be read twice.

In the `checkAmountPercentageValid()` function global variable `amountPercentage` can be read twice

C14-04 Few events ● Low ✓ Resolved

The function `_execute()` from the contract lacks events.

C15. TransferERC721BudgetApproval

Overview

Derived from CommonBudgetApproval, TransferERC721BudgetApproval is one of DAO's budget approvals of ERC721 tokens.

Budget approvals are created and approved. After that, they may be executed. Execution of such approval is a `safeTransferFrom` call to ERC721 token.

Issues

C15-01 Gas optimization ● Low ✓ Resolved

The `initialize()`, `executeParams()` functions can be declared as external to save gas.

C15-02 Unused imports ● Low ✓ Resolved

The functionality of the `IERC721` import is not used in this contract.

C15-03 Few events ● Low ✓ Resolved

The function `_execute()` from the contract lacks events.

C15-04 Function that returns length ● Low ✓ Resolved

There is no public function that returns the length of the `tokens` array.

C16. TransferLiquidERC20BudgetApproval

Overview

Derived from CommonBudgetApproval, TransferERC20BudgetApproval is one of DAO's budget approvals with a list of tokens and base currency specified.

Budget approvals are created and approved. After that they may be executed. Execution of such approval is a transfer of ERC20 token or ETH.

After transfer, the allowed amount in base currency is decreased.

Issues

C16-01 Limitations aren't working ● Medium ✓ Resolved

Limitation of the `checkAmountPercentageValid()` function doesn't work as it should, because this is the limitation only for one transfer, but in one transaction in `CommonBudgetApproval` contract, there can be multiple transfers.

Recommendation

The contract can store the transfer amount for the block. It will allow extending these restrictions to the whole transaction, not only for the function call.

C16-02 Unused imports ● Low ✓ Resolved

The functionality of the `IDao`, `IAdam` imports is not used in this contract.

C16-03 Few events ● Low ✓ Resolved

The function `_execute()` doesn't have an appropriate event.

C16-04 Function that returns length ● Low ✓ Resolved

There is no public function that returns the length of the `tokens` array.

C16-05 Gas optimization ● Low ✓ Resolved

The `initialize()` and the `executeParams()` functions can be declared as external to save gas.

In the `_execute()` function global variables `allowAnyAmount` and `totalAmount` can be read twice.

In the `checkAmountPercentageValid()` function global variables `amountPercentage`, `tokens.length`, `tokens[i]`, and `executee` can be read multiple times.

C17. UniswapBudgetApproval

Overview

Derived from CommonBudgetApproval, UniswapBudgetApproval is one of DAO's budget approvals with a list of whitelisted "from" and "to" tokens, base currency, and maximum tokens approval to the BudgetExecutee contract.

Budget approvals are created and approved. After that, they may be executed. Execution of such approval is a call (swap with base currency) to UniSwap v3 Router address or the WETH address.

Issues

C17-01 Broken returns values from UniswapSwapper are used ● Medium ✓ Resolved

In the function `_execute()` require statements on the lines 101-104 won't work properly because used broken return values from the `decodeUniswapDataAfterSwap()` function from the `UniswapSwapper` contract.

Also, the limitation of the `checkAmountPercentageValid()` function doesn't work as it should because this is the limitation only for one transfer, but in one transaction in the CommonBudgetApproval contract, there can be multiple transfers.

Recommendation

See the recommendations for the C5-01 issue. Fixed return values can be used to track all swaps that are done through the Uniswap router.

Also, the contract can store the transfer amount for the block. It will allow extending these restrictions to the whole transaction, not only for the function call.

C17-02 Unused imports ● Low ✓ Resolved

The functionality of the `IDao`, `IAdam` imports is not used in this contract.

C17-03 Gas optimization ● Low ✓ Resolved

The `initialize()`and `executeParams()` functions can be declared as external to save gas.

In the `afterInitialized()` function global variables `fromTokens.length`, `fromTokens[i]`, and `executee` is read multiple times.

In the `_execute()` function global variables `allowAnyAmount`, and `totalAmount` can be read twice.

In the `checkAmountPercentageValid()` function global variables `amountPercentage`, `fromTokens.length`, `fromTokens[i]`, and `executee` can be read multiple times.

In the function `_executeUniswapCall()` global variables `_tokenInAmount[mData.tokenIn]`, `allowAnyAmount`, `totalAmount`, and `_tokenIn.length` are read multiple times.

In the function `_executeWETH9Call()` global variables `allowAnyAmount` and `totalAmount` are read multiple times.

In the function `_fromTokensPrice()` global variables `fromTokens.length` and `fromTokens[i]` are read multiple times. Also, there are multiple calls of `executee()` function.

C18. Team

Overview

An ERC1155Upgradeable, OwnableUpgradeable Team contract.

Team is assigned to Dao and is used to whitelist the budget approvers and executors.

Dao is assigned a predefined team contract address through Adam.

Governor may create a new team in Dao. When casting a new team, a minter address is specified, and "soulbound" team tokens are minted to the specified members. Later "minter" may remove and add members to the team.

Issues

C18-01 Few events

● Low ✓ Resolved

Many functions from the contract lack events:

1. `addTeam()`
2. `addMembers()`
3. `removeMembers()`

C18-02 Gas optimization

● Low ✓ Resolved

The `addTeam()`, `addMembers()`, `removeMembers()`, `setRepository()` , `setInfo()`, `uri()` functions can be declared as external to save gas.

In the function `addTeam()` global variable `_tokenIds.current()` is read multiple times.

All public functions except the `uri()` function can be set as external functions.

C18-03 Input arguments validation ● Low ✓ Resolved

In `_beforeTokenTransfer()` all checks are done only for the first element of `ids` argument.

C19. Membership

Overview

The ERC721VotesUpgradeable "soulbound" token implementation, issued to the user after the first deposit to the DAO's liquidity pool.

Issues

C19-01 Adding identical members ● Medium ✓ Resolved

There is no check in the `createMember()` function that a member has already been added, so can mint many tokens to one address. Because of that in the contract `Dao` the function `byPassGovern()` can work unproperly.

```
function createMember(address to) public {
    ...
    uint256 newId = _tokenIds.current();
    _safeMint(to, newId, "");
    isMember[to] = true;
    ...
}
```

Recommendation

Add a `require()` statement to check that the value `isMember[to]` is false.

C19-02 Initializing of EIP712 ● Medium ✓ Resolved

In the function `initialize()` should be called the function `__EIP712_init()` from the contract `EIP712Upgradeable`

.

Recommendation

Into the `initialize()` function add call of the `__EIP712_init()` function.

C19-03 Gas optimization ● Low ✓ Resolved

In the function `createMember()` the global variable `totalSupply` is read twice.

Functions `initialize()` and `createMember()` can be set as external functions.

C19-04 Improperly overriding ● Low ✓ Resolved

In the function `_beforeTokenTransfer()` there should be a call of the `super._beforeTokenTransfer()` function.

C19-05 Unused modifier ● Low ✓ Resolved

The `onlyDao()` modifier is not used anywhere. This modifier must be used in the `createMember()` function, but the check via `require()` is used.

```
    function createMember(address to) public {
        require(msg.sender == dao, "access denied");
        ...
    }
```

C20. MemberToken

Overview

An ERC20VotesUpgradeable token, assigned to DAO during its initialization.

It excludes the minter (DAO account) from voting and assures the voting power is attached to the correct account via `_delegate(to, to)` in `afterTokenTransfer`.

Issues

C20-01 Changing delegation of users ● High ✓ Resolved

By a token transfer to some user, a delegation of this user can be changed. For example, there is a contract that delegates its votes to some address in its constructor. After that, some malicious user sends some amount of tokens to this contract and the delegation of this contract will be changed to the address of this contract. This may break the contract's logic. Also, this issue can be used as a front-run attack before some user makes a new proposal. This may add a new vector of attack.

Recommendation

In the function `_afterTokenTransfer()` on line 48 add a condition

```
if (from == address(0)) {
    _delegate(to, to);
}
```

C20-02 Initializing of ERC20Permit ● Medium ✓ Resolved

In the function `initialize()` should be called the `__ERC20Permit_init()` function from the contract `ERC20PermitUpgradeable`.

Recommendation

Add a call to the function `__ERC20Permit_init()` to the `initialize()` function

C20-03 initialize() lacks validation of input parameters ● Low ✓ Resolved

The contract function `initialize()` does not check the address `_minter` against a null address.

C20-04 Gas optimization ● Low ✓ Resolved

The `initialize()`and the `mint()` functions can be declared as external to save gas.

C20-05 Unused modifier ● Low ✓ Resolved

The `onlyDao()` modifier is not used anywhere.

C20-06 Incorrect error message in require statement ● Info ✓ Resolved

Incorrect error message in require on 20L. Instead of `"Not minter"` it should be `"Not dao"`.

C21. LiquidPool

Overview

A liquidity pool that allows staking several types of coins and receiving the membership tokens (via DAO) as well as LP tokens.

Issues

C21-01 Transfers of ERC20 tokens ● Low ✓ Resolved

It is better to use the `SafeERC20` library for transferring ERC20 tokens.

C21-02 Unused imports ● Low ✓ Resolved

The functionality of `ERC1967Proxy` is not used in this contract.

C21-03 Redundant computations ● Low ✓ Resolved

In the function `quote()` on line 65 there are excessive actions. The statement should be replaced with this:

```
return amount * totalSupply() / totalPrice();
```

Also in the function `deposit()` on line 97 the second argument of the `_mint()` function should be replaced with this:

```
assetBaseCurrencyPrice(Denominations.ETH, msg.value) * totalSupply() / total
```

C21-04 Length of an array ● Low ✓ Resolved

There is no function that returns the length of the `assets` array.

C21-05 Variable shadowing ● Low ✓ Resolved

In the `initialize()` function the argument `baseCurrency` shadows global variable of `PriceResolver` contract.

C21-06 Native token transfer ● Low ✓ Resolved

In the function `_transferAsset()` it is better to use the function `call()` instead of the `transfer()` function for transferring native tokens.

C21-07 Gas optimization ● Low ✓ Resolved

In the function `_beforeCreateBudgetApproval()` call to the `Dao` contract can be replaced with a call to the `Adam` contract (`budgetApprovals()` function) to reduce the number of external calls.

In the `onlyGovern()` modifier the global variable `dao` can be read twice.

In the `initialize()` function the global variable `dao` is read after writing.

In the function `depositToken()` the function `assetBaseCurrencyPrice()` is called twice, but the return value is the same.

In the `assetsShares()` function the `totalSupply()` function is called multiple times.

In the `quote()` function `totalSupply()` is called twice.

In the `totalPrice()` and `totalPriceInEth()` functions the global variables `assets.length` and `assets[i]` are read multiple times.

In the `deposit()` the functions `totalSupply()`, `baseCurrencyDecimals()` and `assetBaseCurrencyPrice()` (with the same arguments) are called multiple times.

In the `redeem()` function the global variables `dao`, `assets.length`, and `assets[i]` are read multiple times.

C22. GovernFactory

Overview

The contract used by DAO to create new Governs and associate them to their types.

Also new vote tokens can be attached to the govern via govern factory.

Issues

C22-01 Function {addVoteToken} ● Medium ✓ Resolved

The function `addVoteToken()` won’t work because the `GovernFactory` contract doesn’t own `Govern` contracts.

C22-02 Validation of the arguments ● Low ✓ Resolved

The contract function `initialize()` does not check the address `_governImplementation` against a null address.

C23. Govern

Overview

A GovernorUpgradeable contract implies being created and owned by the DAO.

Governor created with type "General" in GovernFactory is used to rule the DAO.

The quorum's type is Bravo: only positive votes are meant to be counted to reach the quorum, but the votes of type "abstain" are counted too in the current implementation.

Issues

C23-01 Votes counting for the quorum ● Medium ✓ Resolved

In the `_quorumReached()` function only `forVotes` should be counted (because `quorum=bravo`). In the current implementation, `abstainVotes` counted too.

abstainVotes replaced

abstainVotes replaced with againstVotes, but the problem remains

C23-02 Use of weights ● Medium ✓ Resolved

The array `voteWeights` used in the function `getVotes()` but not in `totalPastSupply()`. In some cases, quorum won't be achieved.

C23-03 Problem of {addVoteToken} function ● Medium ✓ Resolved

In some cases the function `addVoteToken()` can break voting and quorum won't be achieved. For example, some proposal is active and some users have already cast their votes to it and then the admin adds a new vote token. In this case, the quorum will be increased and if a big amount of new tokens belongs to the users that have already cast their votes for this proposal, the quorum may be unreached.

C23-04 Wrong return value ● Low ✓ Resolved

The function `quorum()` should return the minimum number of cast votes required for a proposal to be successful but it returns another value.

C23-05 Unused library import ● Low ✓ Resolved

`TimersUpgradeable` and `SafeCastUpgradable` libraries are imported but never used.

C23-06 Validation of the arguments ● Low ✓ Resolved

In the function `initialize()` there should be a check that the length of `_voteWeights` is equal to `_voteTokens`. Also, there's no check that the addresses `_owner` and `_voteTokens` aren't null addresses.

C23-07 Length of arrays ● Low ✓ Resolved

There are no functions that return the length of `voteWeights` and `voteTokens` arrays.

C23-08 Gas optimization ● Low ✓ Resolved

The `initialize()`, `getProposalVote()`, `getVotes()`, `quorumReached()`, `voteSucceeded()`, `addVoteToken()` functions can be declared as external to save gas.

In the `getVotes()` and `totalPastSupply()` functions the global variables `voteTokens.length` and `voteTokens[i]` are read multiple times.

In the `addVoteToken()` function the global variable `voteTokens.length` is read multiple times.

In the `_voteSucceeded()` function the global variable `proposalvote.forVotes` is read multiple times.

C23-09 Unused imports ● Low ✓ Resolved

The functionality of the `ERC1967Proxy`, `ERC165Upgradeable` imports is not used in this contract.

C24. Dao

Overview

A DAO issued by a user in the Adam contract. DAO contract is an implementation of ERC721HolderUpgradeable and ERC1155HolderUpgradable. It derives from the BudgetApprovalExecutee so that DAO executes queries from miscellaneous budget approvals whitelisted in Adam and assigned to DAO by the moment of initialization.

A governed contract: most non-view methods are available only to the Governor of type "General".

Teams, members and member tokens can be created via DAO.

Issues

C24-01 Gas optimization ● Low ✓ Resolved

In the `initialize()` function the global variable `creator` is read after writing.

In the `getVoteTypeValues()` function the global variables `membership` and `memberToken` are read multiple times.

In the `_createMemberToken()` function the global variable `memberToken` is read multiple times.

In the `_setAdmissionToken()` function the global variable `memberToken` is read multiple times.

In the `isPassAdmissionToken()` function the global variables `admissionTokens.length` and `admissionTokenSetting[token].minTokenToAdmit` are read multiple times.

C24-02 Validation of the arguments ● Low ✓ Resolved

The contract function `initialize()` does not check the struct input addresses against a null address.

In the `_setAdmissionToken()` function there is no check that the `_admissionTokens` array contains different values.

C24-03 Length of an array ● Low ✓ Resolved

There is no function that returns the length of `admissionTokens` array.

C24-04 Wrong error message ● Low ✓ Resolved

In the functions `_mintMemberToken()` and `_transferMemberToken()` there is no check that `memberToken` is not a zero address. In the current implementation revert message in case of zero address will be inappropriate.

C24-05 Unused variable ● Low ✓ Resolved

The global variable `teamWhitelist` isn't used anywhere.

C24-06 Unused imports ● Low ✓ Resolved

The functionality of the `IERC165`, `IBudgetApprovalExecutee` imports is not used in this contract.

C24-07 Unreachable function ● Low ✓ Resolved

The function `createGovern()` is declared with the modifier `onlyGovern("Govern")` so that it is unreachable when the govern with type "Govern" is absent and the `totalSupply()` of members is higher than 1. Consider changing the type of govern to "General" or removing the method.

C24-08 Few events ● Low ✓ Resolved

Several functions from the contract lack events:

1. `updateDaoSetting()`

C25. Adam

Overview

The contract allows any user to create his own DAO.

The contract holds the implementations of DAO, Membership, MemberToken and LiquidPool and the list of whitelisted BudgetApprovals that can be updated by the owner.

Issues

C25-01 Few events ● Low ✓ Resolved

Several functions from the contract lack events:

1. `setDaoImplementation()`
2. `setMembershipImplementation()`
3. `setLiquidPoolImplementation()`
4. `setMemberTokenImplementation()`

C25-02 Functions lacks validation of input parameters ● Low ✓ Resolved

The contract functions `initialize()`, `setDaoImplementation()`, `setMembershipImplementation()`, `setLiquidPoolImplementation()`, `setMemberTokenImplementation()` do not check the input addresses against a null address.

C25-03 Unused variable

● Low ✓ Resolved

The `governImplementation` variable is not used in the contract.

C25-04 Removing from the whitelist

● Low ✓ Resolved

There is no function that removes from the whitelist some budget approval.

C25-05 Gas optimization

● Low ✓ Resolved

The `initialize()`, `setDaoImplementation()`, `setMembershipImplementation()`, `setLiquidPoolImplementation()` , `setMemberTokenImplementation()`, `createDao()` functions can be declared as external to save gas.

5. Conclusion

3 high, 12 medium, 62 low severity issues were found during the audit. 3 high, 12 medium, 61 low issues were resolved in the update.

The reviewed contracts are highly dependent on the owner's account. Users using the project have to trust the owner and that the owner's account is properly secured.

The audited contracts are designed to be deployed with proxies. Also, the owner of the contracts is EOA and he can change the implementation of the contracts at any moment. Users have no choice but to trust the owners, who can update the contracts at their will at any time.

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. 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

contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

#HashEx
BLOCKCHAIN SECURITY