

# Open Dollar



## Scope

The code under review can be found within the [C4 Open Dollar repository](#).

## Summary

In this contest, I discovered 4 Medium-severity issues (including a solo/unique finding) with my QA report being selected for the final public report.

## Findings

ID	Issues	Severity
M-01	User cannot call <code>allowSafe</code> function in <code>ODSafeManager.sol</code> to give safe permissions	Medium
M-02	Approved address can approve other addresses for an owner's safe	Medium
M-03	Function <code>allowHandler()</code> can never be called from a <code>safeHandler</code>	Medium
M-04	Old permissions in <code>handlerCan</code> mapping are still attached to the <code>safeHandler</code> of a transferred safe	Medium
L-01	Image field does not point to an image but the testnet website	Low

ID	Issues	Severity
L-02	Missing check in constructor to see if quotePeriod is not zero	Low
L-03	Missing cardinality check in function read()	Low
L-04	Rounding down in <code>_exitCollateral()</code> function can cause loss of precision leading to loss of funds for users	Low
L-05	Attacker can frontrun initializer functions in Vault721.sol contract to mint safes with any safeld in order to brick the protocol	Low
L-06	ARB supplied through execute() function in user's ODProxy is permanently stuck	Low
L-07	Missing safeAllowed() modifier on addSAFE() function allows non-owners to add an owner's safe to their list of safes	Low
N-01	Public variable not used in external contracts can be marked private/internal	Non-Critical
N-02	Missing event emission for critical state changes	Non-Critical
N-03	Cache variable early to prevent redundant caching and an extra SLOAD	Non-Critical
N-04	Remove redundant condition in <code>_isNotProxy()</code> function	Non-Critical
N-05	Remove <code>if (_dst == address(0)) revert ZeroAddress();</code> redundant check since it is already checked in <code>_afterTokenTransfer()</code> function	Non-Critical
N-06	Function read() does not revert with "OLD!" as mentioned in comments	Non-Critical
N-07	Remove <code>if (extraSurplusReceiver == address(0)) revert AccEng_NullSurplusReceiver();</code> redundant check	Non-Critical
R-01	Consider modifying the build() function which allows anyone to create an ODProxy for any user	Recommendation
R-02	Use <code>user</code> instead of <code>usr</code> in mappings to improve readability	Recommendation
R-03	Use <code>safeAllow</code> and <code>handlerAllow</code> instead of <code>safeCan</code> and <code>handlerCan</code> to better match the intention of the mappings	Recommendation
R-04	Add brackets around <code>10 ** multiplier</code> to improve code readability and provide clarity in which operation takes precedence first	Recommendation

## Findings

## [M-01] User cannot call `allowSafe` function in `ODSafeManager.sol` to give safe permissions

### Impact

The function `allowSafe` in the `ODSafeManager.sol` contract is intended to be called directly from a user's `ODProxy` address. But since the `execute()` function calls the `allowSafe()` function using `delegateCall` (which does not change the `msg.sender` context to `ODProxy` instead of user's address), the user is not able to give safe permissions to someone.

### Proof of Concept

Here is the whole process:

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODProxy.sol#L26](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODProxy.sol#L26)

1. User (EOA) calls `execute()` function in their `ODProxy` with `address _target` parameter as `ODSafeManager` contract and `bytes memory data` parameter as the function signature for `allowSafe()` function (along with values).

```
File: ODProxy.sol
26:  function execute(address _target, bytes memory _data) external
    payable onlyOwner returns (bytes memory _response) {
27:      if (_target == address(0)) revert TargetAddressRequired();
28:
29:
30:      bool _succeeded;
31:      (_succeeded, _response) = _target.delegatecall(_data);
32:
33:
34:      if (!_succeeded) {
35:          revert TargetCallFailed(_response);
36:      }
37:  }
```

2. The `ODProxy` of the user delegate calls the `allowSafe()` function, which keeps the `msg.sender` as the user address only and not the `ODProxy` contract.

```
File: ODProxy.sol
30:      bool _succeeded;
31:      (_succeeded, _response) = _target.delegatecall(_data);
```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L105C1-L109C4](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L105C1-L109C4)

3. The `allowSafe()` function gets called with the respective values. Before the execution begins, we enter the modifier `safeAllowed()` which takes in the `_safe` as parameter for which the permissions are being given to `_usr`.

```
File: ODSafeManager.sol
105:  function allowSAFE(uint256 _safe, address _usr, uint256 _ok)
external safeAllowed(_safe) {
106:    address _owner = _safeData[_safe].owner;
107:    safeCan[_owner][_safe][_usr] = _ok;
108:    emit AllowSAFE(msg.sender, _safe, _usr, _ok);
109: }
```

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L49C1-L53C4>

4. In the modifier, the following happens:

- On Line 50, the owner of the safe is extracted, which is the ODProxy contract
- On Line 51, we check if the `msg.sender` is not the owner. This condition is true since the `msg.sender` is the user's EOA address and not the ODProxy contract of the user due to the `delegateCall` that was made.
- On Line 51, the second condition evaluates to true as well since the user cannot give permissions to anyone (even themselves) in the first place.
- This causes both conditions to evaluate to true and we revert.

```
File: ODSafeManager.sol
49:  modifier safeAllowed(uint256 _safe) {
50:    address _owner = _safeData[_safe].owner;
51:    if (msg.sender != _owner && safeCan[_owner][_safe][msg.sender] ==
0) revert SafeNotAllowed();
52:    _;
53: }
```

## Tools Used

### Manual Review

## Recommended Mitigation Steps


Consider either implementing another function in the ODProxy contract specifically to allow calling `allowSafe()` or use a different modifier that checks if the owner of the ODProxy is the caller.

## [M-02] Approved address can approve other addresses for an owner's safe

### Impact

An owner of a safe can give permissions/approval of their safe to another address (let's say address B) through the `allowSafe()` function in the `ODSafeManager.sol` contract. But this other address (address B) also gets the power to approve other addresses for the owner's safe. This is a permissioning problem in the `allowSafe()` function (specifically the `safeAllowed()` modifier) which creates a security risk for the owner's safe.


**This might look like a design choice initially but it has been confirmed as an issue with the sponsor:**




**MrPotatoMagic** Today at 1:40 AM

Is it intended behaviour to allow permissioned safe users (stored in `safeCan`) to give access to call the `allowSafe` function for that specific safe? I think it is a security risk to give them access to the `allowSafe` function

[https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L105C1-L109C4)



**@MrPotatoMagic** Is it intended behaviour to allow permissioned safe users (stored in `safeCan`) to give access to call the all



**hunter** Today at 2:15 AM

i agree. an approved address could approve other addresses. def a vulnerability 👍

## Proof of Concept

Here is the whole process:

[https://github.com/open-dollar/od-](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L105C1-L109C4)

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L105C1-L109C4](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L105C1-L109C4)

1. Owner of the safe gives permission/approval to `address _usr` (address `0x01` for example purposes) for `uint256 _safe` through the `allowSafe()` function.
- On Line 107, `safeCan[_owner][_safe][_usr] = _ok;` is set to any value other than 0 to represent approval.

```
File: ODSafeManager.sol
105:  function allowSAFE(uint256 _safe, address _usr, uint256 _ok)
external safeAllowed(_safe) {
106:    address _owner = _safeData[_safe].owner;
107:    safeCan[_owner][_safe][_usr] = _ok;
108:    emit AllowSAFE(msg.sender, _safe, _usr, _ok);
109: }
```

2. The previously set `address _usr` (address `0x01`) can now call the `allowSafe()` function with parameters `uint256 _safe` which will be the owner's safe and another `address _usr` (address `0x02`), which will give address `0x02` permissions/approval for the owner's safe.
3. This issue arises because of how the checks are evaluated in the `safeAllowed()` modifier. Here is what happens:
  - On Line 50, the owner of the safe is extracted.
  - On Line 51, there are two conditions present that are separated by the `&&` operator.

- On Line 51, the first check evaluates to true, since the msg.sender (address 0x01) is not the owner of the safe
- On Line 51, the second check evaluates to false, since the msg.sender (address 0x01) was previously approved by the owner in step 1 above.
- Since true && false = false, we do not revert and this gives address 0x02 permissions to the owner's safe in the allowSafe() function.

```
File: ODSafeManager.sol
49:   modifier safeAllowed(uint256 _safe) {
50:       address _owner = _safeData[_safe].owner;
51:       if (msg.sender != _owner && safeCan[_owner][_safe][msg.sender] ==
0) revert SafeNotAllowed();
52:   _;
53: }
```

## Tools Used

### Manual Review

## Recommended Mitigation Steps

Consider implementing a separate modifier for the allowSafe() function that only checks if the msg.sender is the owner. If true, then allow execution but if not then revert.

Solution:

```
File: ODSafeManager.sol
modifier onlySafeOwner(uint256 _safe) {
    address _owner = _safeData[_safe].owner;
    if (msg.sender != _owner) revert SafeNotAllowed();
    _;
}
```

## [M-03] Function `allowHandler()` can never be called from a safeHandler

### Impact

The `allowHandler()` function is used to allow/disallow a handler address to manage a safe (as mentioned in the [documentation here](#)). The function is designed in such a way that a safeHandler (msg.sender) needs to make a call to the function allowHandler() with the address being allowed/disallowed as the parameter. But the safeHandler is not able call this function since it [does not contain code in it's contract](#) to make this call. This prevents a safeHandler from approving other addresses. Additionally, functions (`quitSystem()` and `enterSystem()`) in the ODSafeManager.sol contract that use the `handlerAllowed()` modifier revert due to this issue.

### Proof of Concept

Here is the whole process:

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol](https://github.com/open-dollar/od-)  
#L41

1. Let us understand the layout of the handlerCan mapping first.

- [Description of mapping from documentation](#) - Mapping of handler to a caller permissions
- The key field is the `_safeHandler` of a safe for which permissions are being given to the value field which is the `caller` address with the approval/disapproval represented by `_ok`

```
File: ODSafeManager.sol
41:  mapping(address _safeHandler => mapping(address _caller => uint256
_ok)) public handlerCan;
```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol](https://github.com/open-dollar/od-)  
#L112

2. In the function `allowHandler()` below, the following happens:

- On Line 113, `msg.sender` is expected to be the `safeHandler` address calling the function so that the permissions are set correctly. This is where the issue arises since for `safeHandler` to be the `msg.sender`, the call needs to originate from the [safeHandler contract](#), which does not have any function within it to make this call. Due to this, the `allowHandler()` function cannot be called and thus any functions in the `ODSafeManager.sol` that rely on the `handlerAllowed` modifier (or the `handlerCan` mapping) revert.

```
File: ODSafeManager.sol
112:  function allowHandler(address _usr, uint256 _ok) external {
113:    handlerCan[msg.sender][_usr] = _ok;
114:    emit AllowHandler(msg.sender, _usr, _ok);
115:  }
```

## Tools Used

## Manual Review

## Recommended Mitigation Steps

The solution to this problem would be to implement an access controlled function within the [SAFEHandler contract](#) that allows the owner of the safe (since `safeHandler` is unique to a safe) to make an external call to the `allowHandler()` function.

**[M-04] Old permissions in handlerCan mapping are still attached to the safeHandler of a transferred safe**

## Impact

A safe is associated with a unique safeHandler. This safeHandler can give permissions to addresses through the [allowHandler\(\)](#) function which stores these approvals/disapprovals in the [handlerCan](#) mapping. Now let's say there is a safeHandler which has permissioned some addresses in the handlerCan mapping. When this safe is transferred to a new owner, the previous permissions that were added to the safeHandler are still attached to it without the new owner realizing it.

## Proof of Concept

Here is the whole process:

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L112C1-L115C4](#)

1. Safe handler of a safe (owned by owner A) approves addresses [X,Y,Z] to the [handlerCan](#) mapping through the [allowHandler\(\)](#) function.

```
File: ODSafeManager.sol
112: function allowHandler(address _usr, uint256 _ok) external {
113:     handlerCan[msg.sender][_usr] = _ok;
114:     emit AllowHandler(msg.sender, _usr, _ok);
115: }
```

2. Owner A now transfers the safe (associated with the safe handler) to an Owner B through the [safeTransferFrom\(\)](#) function in the Vault721.sol contract which [inherits the ERC721Enumerable contract](#) (that inherits the ERC721 contract). During the call, the [\\_afterTokenTransfer\(\)](#) hook is called ([overridden in the Vault721.sol contract](#)), which further calls the [transferSAFEOwnership\(\)](#) function. In the function, we can see that the safe is transferred but the [handlerCan](#) mapping is not updated for the safeHandler, which means the old permissions (addresses [X,Y,Z]) for the safeHandler are still attached without the new owner B realizing it.

```
File: ODSafeManager.sol
136: function transferSAFEOwnership(uint256 _safe, address _dst)
external {
137:     require(msg.sender == address(vault721), 'SafeMngr: Only
Vault721');
138:
139:
140:     if (_dst == address(0)) revert ZeroAddress();
141:     SAFEData memory _sData = _safeData[_safe];
142:     if (_dst == _sData.owner) revert AlreadySafeOwner();
143:
144:
145:     _usrSafes[_sData.owner].remove(_safe);
146:     _usrSafesPerCollat[_sData.owner]
[_sData.collateralType].remove(_safe);
```



```
147:
148:
149:     _usrSafes[_dst].add(_safe);
150:     _usrSafesPerCollat[_dst][_sData.collateralType].add(_safe);
151:
152:
153:     _safeData[_safe].owner = _dst;
154:
155:
156:     emit TransferSAFEOwnership(msg.sender, _safe, _dst);
157: }
```

## Tools Used

Manual Review

## Recommended Mitigation Steps

It is not possible to remove the [handlerCan](#) permissions in the [transferSAFEOwnership\(\)](#) function since it is stored in a mapping. The only solution to this would be to add another key field (named `_owner`) to update the `safeHandler` permissions correctly whenever a safe is transferred. We can see this pattern implemented for the [safeCan mapping](#), which correctly updates the safe permissions on transfer.

Solution (use this mapping instead):

```
File: ODSafeManager.sol
41:     mapping(address _owner => (address _safeHandler => mapping(address
    _caller => uint256 _ok))) public handlerCan;
```

## [L-01] Image field does not point to an image but the testnet website

There is 1 instance of this issue:

In the below string variable `contractMetaData`, we can observe the `image` field points as such `"image": "https://app.opendollar.com/collectionImage.png"`. If we follow the link it leads us to the testnet website and not an image.

```
File: Vault721.sol
22:     string public contractMetaData =
23:         '{"name": "Open Dollar Vaults","description": "Tradable Vaults for
the Open Dollar stablecoin protocol. Caution! Trading this NFT means
trading the ownership of your Vault in the Open Dollar protocol and all of
the assets/collateral inside each Vault.", "image":
"https://app.opendollar.com/collectionImage.png", "external_link":
"https://opendollar.com"}';
```

## [L-02] Missing check in constructor to see if `quotePeriod` is not zero

There is 1 instance of this:

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/oracles/CamelotRelayer.sol#L59>

```
File: CamelotRelayer.sol
59:     quotePeriod = _quotePeriod;
```

## [L-03] Missing cardinality check in function read()

There is 1 instance of this issue:

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/oracles/CamelotRelayer.sol#L91C1-L101C4>

On Line 93, the comment mentions that the read() function should revert with "OLD!" if the pool does not have enough cardinality or initialized history. But there is no check done for the cardinality, which can return an incorrect quote.

```
File: CamelotRelayer.sol
092: function read() external view returns (uint256 _result) {
093:     // This call may revert with 'OLD!' if the pool doesn't have
    enough cardinality or initialized history
094:     (int24 _arithmeticMeanTick,) = OracleLibrary.consult(camelotPair,
    quotePeriod);
095:     uint256 _quoteAmount = OracleLibrary.getQuoteAtTick({
096:         tick: _arithmeticMeanTick,
097:         baseAmount: baseAmount,
098:         baseToken: baseToken,
099:         quoteToken: quoteToken
100:     });
101:     _result = _parseResult(_quoteAmount);
102: }
```

Solution:

```
File: CamelotRelayer.sol
69: function read() external view returns (uint256 _result) {
70:     // If the pool doesn't have enough history return false
71:     if (OracleLibrary.getOldestObservationSecondsAgo(camelotPair) <
    quotePeriod) {
72:         return (0, false);
73:     }
74:     // Consult the query with a TWAP period of quotePeriod
75:     (int24 _arithmeticMeanTick,) = OracleLibrary.consult(camelotPair,
```

```

quotePeriod);
76:      // Calculate the quote amount
77:      uint256 _quoteAmount = OracleLibrary.getQuoteAtTick({
78:          tick: _arithmeticMeanTick,
79:          baseAmount: baseAmount,
80:          baseToken: baseToken,
81:          quoteToken: quoteToken
82:      });
83:      // Process the quote result to 18 decimal quote
84:      _result = _parseResult(_quoteAmount);
85:  }

```

## [L-04] Rounding down in `_exitCollateral()` function can cause loss of precision leading to loss of funds for users

There is 1 instance of this issue:

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/actions/CommonActions.sol#L118](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/actions/CommonActions.sol#L118)

In the function `_exitCollateral()`, users can experience loss of funds if the wad amount is smaller than the decimals representation in the denominator.

The below code snippet is from the `CommonActions.sol` contract is inherited in the `BasicActions.sol` contract. This `_exitCollateral()` function is called when the `freeTokenCollateral()` function in the `BasicActions.sol` contract is called. This function does the operation below based on the decimals of the ERC20 token being used. In case the numerator i.e. the `_wad` amount is smaller than the denominator, the final `_weiAmount` rounds down to zero. This can lead to loss of funds for the user who tries to exit with his collateral.

```

File: CommonActions.sol
118:      uint256 _weiAmount = _wad / 10 ** (18 - _decimals);
119:      __collateralJoin.exit(msg.sender, _weiAmount);

```

## [L-05] Attacker can frontrun initializer functions in `Vault721.sol` contract to mint safes with any safeld in order to brick the protocol

### Impact

**Root cause:** The `initializeManager` and `initializeRenderer` functions can be frontrun by an attacker to (Note: Below mentioned are the 3 impacts of the root cause):

1. Set malicious `safeManager`, build an `ODProxy` (for themselves or any other address) through `build()` function, mint safes with any `safeld` (for themselves or any other address) by calling `mint()` with the help of the malicious `safeManager`
2. Set malicious or invalid `safeManager` and `nftRenderer` addresses only
3. Brick protocol from opening more safes

**Note: Minting safes from the malicious safeManager and not the original safeManager breaks the core invariant of the Vault721.sol contract, which mentions that the safeManager should enforce that only ODProxies call the `openSAFE()` function (obtained from the comments - See [here](#)):**

```
File: src/contracts/proxies/Vault721.sol
90:  /**
91:   * @dev mint can only be called by the SafeManager
92:   * enforces that only ODProxies call `openSafe` function by checking
   _proxyRegistry
93:   */
```

## Proof of Concept

Here is the whole process:

### A) Let us understand how the first impact will be executed:

1. Below we can observe the `initializeManager` function that does not have access control on it and is open to call due to external visibility.

```
File: src/contracts/proxies/Vault721.sol
56:  function initializeManager() external {
57:      if (address(safeManager) == address(0))
   _setSafeManager(msg.sender);
58:  }
```

2. Attacker calls the above function **(through his malicious safeManager address since `msg.sender` is passed to `_setSafeManager()`)** with a higher gas price in order to frontrun team's transaction and sets malicious address for `safeManager` **(Note: Here the team's tx does not revert and goes through (acting like the team's tx was successful) without making any changes to the `safeManager` state variable)**
3. Attacker calls the `build()` function (for himself as `msg.sender` or for any other address) to build a proxy:

```
File: src/contracts/proxies/Vault721.sol
77:  function build() external returns (address payable _proxy) {
78:      if (!_isNotProxy(msg.sender)) revert ProxyAlreadyExist();
79:      _proxy = _build(msg.sender);
80:  }
81:
82:
83:  /**
84:   * @dev allows user without an ODProxy to deploy a new ODProxy
85:   */
86:  function build(address _user) external returns (address payable
   _proxy) {
```

```

87:     if (!_isNotProxy(_user)) revert ProxyAlreadyExist();
88:     _proxy = _build(_user);
89: }

```

4. Attacker calls `mint()` through the malicious `safeManager` to pass the check on Line 95 below (since `safeManager` was initialized to malicious `safeManager` previously). The attacker also passes the check on Line 96 since he built a proxy (for himself or any other address) through the `build()` function in the previous step. Line 98 mints the safe with any `safeId` for the attacker or the other address.

```

File: src/contracts/proxies/Vault721.sol
94:  function mint(address _proxy, uint256 _safeId) external {
95:    require(msg.sender == address(safeManager), 'V721: only
safeManager');
96:    require(_proxyRegistry[_proxy] != address(0), 'V721: non-native
proxy');
97:    address _user = _proxyRegistry[_proxy];
98:    _safeMint(_user, _safeId);
99: }

```

## B] Let us understand how the second impact will be executed (more simplistic):

1. Below we can observe the `initializeManager` and `initializeRenderer` functions that do not have access control on them and are open to call due to external visibility.

```

File: src/contracts/proxies/Vault721.sol
56:  function initializeManager() external {
57:    if (address(safeManager) == address(0))
_setSafeManager(msg.sender);
58:  }
59:
60:
61:  /**
62:   * @dev initializes NFTRenderer contract
63:   */
64:  function initializeRenderer() external {
65:    if (address(nftRenderer) == address(0))
_setNftRenderer(msg.sender);
66:  }

```

2. Attacker calls the above two functions **(through his malicious `safeManager` address since `msg.sender` is passed to `_setSafeManager()`)** with a higher gas price in order to frontrun team's transaction and sets malicious or invalid values for `safeManager` and `nftRenderer` addresses **(Note: Here the team's tx does not revert and goes through (acting like the team's tx was successful) without making any changes to the `safeManager` state variable).**

## C] Let us understand the 3rd impact (bricking the protocol) now:

Let's say the team realizes this attack and resets the `safeManager` and `nftRenderer` to the correct addresses using `setSafeManager()` and `setNftRenderer()`. If during the attack, a safe is opened by the attacker for a `safeld` (let's say 10), then when the `safeld` state variable in the `ODSafeManager` contract reaches 10, no more safes can be opened by anyone (since `_safeMint()` would revert) thus bricking the protocol.

### There are 4 important points to note here:

1. Core invariant of the `Vault721.sol` contract is broken (i.e. minting safes can only occur through the `safeManager`, which should enforce that only `ODProxies` call the `openSAFE()` function).
2. After the frontrun transaction of the attacker gets executed and sets the values, the following second transaction from the team still gets executed and does not revert, thus acting like the team's tx was successful in setting the values. Additionally, there is no event emission occurring when the values are set, which does not allow offchain monitoring to track this issue as well.
3. The `initializeManager` function is called during deployment of the `safeManager` contract itself (see [here](#)). This can give the attacker enough time to just do a normal transaction instead of even frontrunning the team's tx since the `Vault721.sol` contract (containing `initializeManager`) would need to be deployed first before the `ODSafeManager.sol` contract (containing call to `initializeManager()` in constructor) is deployed.

### Tools Used

### Manual Review

### Recommended Mitigation Steps

The simplest solution to this issue would be to:

1. Remove [Line 67](#) from the constructor of the `safeManager` contract:

```
File: src/contracts/proxies/ODSafeManager.sol
64:   constructor(address _safeEngine, address _vault721) {
65:       safeEngine = _safeEngine.assertNonNull();
66:       vault721 = IVault721(_vault721);
67:       vault721.initializeManager(); //@audit Remove this line
68:   }
```

2. Add the `onlyGovernor` modifier over both `initializeManager` and `initializeRenderer` functions to ensure only governance can call them manually (Solution below):

```
File: src/contracts/proxies/Vault721.sol
```solidity
File: src/contracts/proxies/Vault721.sol
56:   function initializeManager() external onlyGovernor {
57:       if (address(safeManager) == address(0))
58:           _setSafeManager(msg.sender);
59:   }
```

```

60:
61:  /**
62:   * @dev initializes NFTRenderer contract
63:   */
64:  function initializeRenderer() external onlyGovernor {
65:      if (address(nftRenderer) == address(0))
        _setNftRenderer(msg.sender);
66:  }

```

## [L-06] ARB supplied through execute() function in user's ODProxy is permanently stuck

### Impact

ARB (i.e. msg.value) is supplied through the payable execute() function in the user's ODProxy but it is not forwarded further by the delegatecall, which can cause the native ARB to be permanently locked in the user's ODProxy contract.

### Proof of Concept

Here is the whole process:

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODProxy.sol#L26](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODProxy.sol#L26)

1. User calls the payable `execute()` function with the respective values and supplies msg.value amount to provide ARB as collateral.

```

File: ODProxy.sol
26:  function execute(address _target, bytes memory _data) external
    payable onlyOwner returns (bytes memory _response) {
27:      if (_target == address(0)) revert TargetAddressRequired();
28:
29:      bool _succeeded;
30:      (_succeeded, _response) = _target.delegatecall(_data);
31:
32:      if (!_succeeded) {
33:          revert TargetCallFailed(_response);
34:      }
35:  }

```

2. On Line 30, we see the user's ODProxy making a delegate call on the `_target` address with `_data` but it does not forward the msg.value that was supplied by the user. This causes the funds to be stuck in the user's ODProxy contract permanently.

```

File: ODProxy.sol
29:      bool _succeeded;
30:      (_succeeded, _response) = _target.delegatecall(_data);

```

## Tools Used

Manual Review

## Recommended Mitigation Steps


Implement a withdrawal mechanism to withdraw this ARB.



### [L-07] Missing `safeAllowed()` modifier on `addSAFE()` function allows non-owners to add an owner's safe to their list of safes

#### Impact

The `addSAFE()` function in the `ODSafeManager.sol` contract allows non-owners to add an owner's safe to their list of safes in the `_usrSafes` and `_usrSafesPerCollat` mappings. This would be a big unfixable UI and storage problem since when the safe data is pulled from the mappings, it would show multiple users having the same safe. The mappings would incorrectly hold these values since each owner has their own `EnumerableSet.UintSet`. Additionally, once a safeld is added, the non-owners will not be able to remove them since `removeSAFE()` is access controlled with `safeAllowed()`.

#### Conversation between sponsor and I for more context:

**MrPotatoMagic** Today at 4:52 PM  
Hi, had another question. Currently, the `addSafe` function allows users to add a safeld to their list of safes. This means anyone can add anyone's safe to their list since it does not check if the owner is the caller. In this case `getSafesData` will return incorrect data. Wondering if this is an issue since idk how `getSafesData` will be used?  
<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L235>

**hunrr** Today at 7:31 PM  
@MrPotatoMagic Hi, had another question. Currently, the `addSafe` function allows users to add a safeld to their list of safes  
this is potentially an issue. especially since `removeSafe` is access controlled, so if a non-owner adds safe, they will not be able to remove it, which presents a UI problem for sure  
to be clear, `addSafe` and `removeSafe` are intended for the frontend to pull a list of safes into UI  
 1

## Proof of Concept

Here is the whole process:

[https://github.com/open-dollar/od-](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L235)

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L235](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L235)

1. In the `addSAFE()` function below we can see it does not have the `safeAllowed()` modifier, which allows anyone to call it. A non-owner of a safe calls it with the safeld of an owner.

```
File: ODSafeManager.sol
235:   function addSAFE(uint256 _safe) external {
236:       SAFEData memory _sData = _safeData[_safe];
```



```

237:     _usrSafes[msg.sender].add(_safe);
238:     _usrSafesPerCollat[msg.sender][_sData.collateralType].add(_safe);
239: }

```

2. On Lines 237 and 238, the mappings `_usrSafes` and `_usrSafesPerCollat` add the safe to the non-owner's list of safes.

```

File: ODSafeManager.sol
237:     _usrSafes[msg.sender].add(_safe);
238:     _usrSafesPerCollat[msg.sender][_sData.collateralType].add(_safe);

```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L73C1-L95C4](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L73C1-L95C4)

3. Now whenever the safes of the non-owner are retrieved through the below getter functions or the two mappings mentioned above, it would return the non-owned safes as well.

```

File: ODSafeManager.sol
73:  function getSafes(address _usr) external view returns (uint256[]
memory _safes) {
74:     _safes = _usrSafes[_usr].values();
75: }
76:
77:
78:  /// @inheritdoc IODSafeManager
79:  function getSafes(address _usr, bytes32 _cType) external view
returns (uint256[] memory _safes) {
80:     _safes = _usrSafesPerCollat[_usr][_cType].values();
81: }
82:
83:
84:  /// @inheritdoc IODSafeManager
85:  function getSafesData(address _usr)
86:     external
87:     view
88:     returns (uint256[] memory _safes, address[] memory _safeHandlers,
bytes32[] memory _cTypes)
89:  {
90:     _safes = _usrSafes[_usr].values();
91:     _safeHandlers = new address[](_safes.length);
92:     _cTypes = new bytes32[](_safes.length);
93:     for (uint256 _i; _i < _safes.length; _i++) {
94:         _safeHandlers[_i] = _safeData[_safes[_i]].safeHandler;
95:         _cTypes[_i] = _safeData[_safes[_i]].collateralType;
96:     }
97: }

```

4. In case the safe is added by mistake and the user wants to now remove it, the user is not able to do so since the `removeSAFE()` function is access controlled with the `safeAllowed()` modifier, which only allows the owner of a safe or approved users to remove the safe from their list of safes. Due to this, the UI problem still exists and cannot be fixed.

```
File: ODSafeManager.sol
266:   function removeSAFE(uint256 _safe) external safeAllowed(_safe) {
267:       SAFEData memory _sData = _safeData[_safe];
268:       _usrSafes[_sData.owner].remove(_safe);
269:       _usrSafesPerCollat[_sData.owner]
[_sData.collateralType].remove(_safe);
270:   }
```

## Tools Used

### Manual Review

## Recommended Mitigation Steps

Add the `safeAllowed()` modifier to the `addSAFE()` function to ensure only the safe owner can add a safe or approved users.

## [N-01] Public variable not used in external contracts can be marked private/internal

There are 3 instances of this:

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/Vault721.sol#L18C1-L20C34>

```
File: src/contracts/proxies/Vault721.sol
18:   address public governor;
19:   IODSafeManager public safeManager;
20:   NFTRenderer public nftRenderer;
```

## [N-02] Missing event emission for critical state changes

There are 13 instances of this:

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/Vault721.sol#L33C1-L35C4>

```
File: Vault721.sol
34:   constructor(address _governor) ERC721('OpenDollar Vault', 'ODV') {
```

```
35:     governor = _governor;
36:     //@audit NC - missing event emission
37: }
```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/Vault721.sol#L104C1-L114C4](https://github.com/open-dollar/od-)

```
File: Vault721.sol
110: function updateNftRenderer(
111:     address _nftRenderer,
112:     address _oracleRelayer,
113:     address _taxCollector,
114:     address _collateralJoinFactory
115: ) external onlyGovernor nonZero(_oracleRelayer)
nonZero(_taxCollector) nonZero(_collateralJoinFactory) {
116:     address _safeManager = address(safeManager);
117:     require(_safeManager != address(0));
118:     _setNftRenderer(_nftRenderer);
119:     nftRenderer.setImplementation(_safeManager, _oracleRelayer,
_taxCollector, _collateralJoinFactory);
120:     //@audit NC - missing event emission
121: }
```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/Vault721.sol#L119C1-L121C4](https://github.com/open-dollar/od-)

```
File: Vault721.sol
127: function updateContractURI(string memory _metaData) external
onlyGovernor {
128:     contractMetaData = _metaData;
129:     //@audit NC - missing event emission
130: }
```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/Vault721.sol#L172C1-L175C1](https://github.com/open-dollar/od-)

```
File: Vault721.sol
183: function _setSafeManager(address _safeManager) internal
nonZero(_safeManager) {
184:     safeManager = IODSafeManager(_safeManager);
185:     //@audit NC - missing event emission
186: }
```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/Vault721.sol#L179C1-L181C4](https://github.com/open-dollar/od-)

```
File: Vault721.sol
191:   function _setNftRenderer(address _nftRenderer) internal
nonZero(_nftRenderer) {
192:       nftRenderer = NFTRenderer(_nftRenderer);
193:       //@audit NC - missing event emission
194:   }
```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L64C1-L68C4](https://github.com/open-dollar/od-)

```
File: ODSafeManager.sol
65:   constructor(address _safeEngine, address _vault721) {
66:       safeEngine = _safeEngine.assertNonNull();
67:       vault721 = IVault721(_vault721);
68:       vault721.initializeManager();
69:       //@audit NC - missing event emission
70:   }
```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L235](https://github.com/open-dollar/od-)

```
File: ODSafeManager.sol
265:   function addSAFE(uint256 _safe) external {
266:       SAFEData memory _sData = _safeData[_safe];
267:       _usrSafes[msg.sender].add(_safe);
268:       _usrSafesPerCollat[msg.sender][_sData.collateralType].add(_safe);
269:       //@audit NC - missing event emission
270:   }
```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L242](https://github.com/open-dollar/od-)

```
File: ODSafeManager.sol
273:   function removeSAFE(uint256 _safe) external safeAllowed(_safe) {
274:       SAFEData memory _sData = _safeData[_safe];
275:       _usrSafes[_sData.owner].remove(_safe);
276:       _usrSafesPerCollat[_sData.owner]
[_sData.collateralType].remove(_safe);
```

```
277:    //@audit NC - missing event emission
278: }
```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODProxy.sol#L14C1-L17C1](https://github.com/open-dollar/od-)

```
File: ODProxy.sol
14:  constructor(address _owner) {
15:    OWNER = _owner;
16:    //@audit NC - missing event emission
17:  }
```

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/oracles/CamelotRelayer.sol#L40C1-L62C4](https://github.com/open-dollar/od-)

```
File: CamelotRelayer.sol
40:  constructor(address _baseToken, address _quoteToken, uint32
_quotePeriod) {
41:    // camelotPair =
ICamelotFactory(_CAMELOT_FACTORY).getPair(_baseToken, _quoteToken);
42:    camelotPair =
IAlgebraFactory(_CAMELOT_FACTORY).poolByPair(_baseToken, _quoteToken);
43:    if (camelotPair == address(0)) revert
CamelotRelayer_InvalidPool();
44:
45:    address _token0 = ICamelotPair(camelotPair).token0();
46:    address _token1 = ICamelotPair(camelotPair).token1();
47:
48:    // The factory validates that both token0 and token1 are desired
baseToken and quoteTokens
49:    if (_token0 == _baseToken) {
50:      baseToken = _token0;
51:      quoteToken = _token1;
52:    } else {
53:      baseToken = _token1;
54:      quoteToken = _token0;
55:    }
56:
57:    baseAmount = uint128(10 ** IERC20Metadata(_baseToken).decimals());
58:    multiplier = 18 - IERC20Metadata(_quoteToken).decimals();
59:    quotePeriod = _quotePeriod;
60:
61:    symbol =
string(abi.encodePacked(IERC20Metadata(_baseToken).symbol(), ' / ',
IERC20Metadata(_quoteToken).symbol()));
62:    //@audit NC - missing event emission
63:  }
```

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/AccountingEngine.sol#L86>

```
File: AccountingEngine.sol
086:   constructor(
087:       address _safeEngine,
088:       address _surplusAuctionHouse,
089:       address _debtAuctionHouse,
090:       AccountingEngineParams memory _accEngineParams
091:   ) Authorizable(msg.sender) validParams {
092:       safeEngine = ISAFEEngine(_safeEngine.assertNonNull());
093:       _setSurplusAuctionHouse(_surplusAuctionHouse);
094:       debtAuctionHouse = IDebtAuctionHouse(_debtAuctionHouse);
095:
096:       lastSurplusTime = block.timestamp;
097:
098:       _params = _accEngineParams;
099:       //@audit NC - missing event emission
100:   }
```

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/AccountingEngine.sol#L247>

```
File: AccountingEngine.sol
265:   function _onContractDisable() internal override {
266:       totalQueuedDebt = 0;
267:       totalOnAuctionDebt = 0;
268:       disableTimestamp = block.timestamp;
269:
270:       surplusAuctionHouse.disableContract();
271:       debtAuctionHouse.disableContract();
272:
273:       uint256 _debtToSettle =
Math.min(safeEngine.coinBalance(address(this)),
safeEngine.debtBalance(address(this)));
274:       safeEngine.settleDebt(_debtToSettle);
275:       //@audit NC - missing event emission
276:   }
```

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/AccountingEngine.sol#L305>

```
File: AccountingEngine.sol
326:   function _setSurplusAuctionHouse(address _surplusAuctionHouse)
internal {
327:       if (address(surplusAuctionHouse) != address(0)) {
```

```

328:         safeEngine.denySAFEModification(address(surplusAuctionHouse));
329:     }
330:     surplusAuctionHouse = ISurplusAuctionHouse(_surplusAuctionHouse);
331:     safeEngine.approveSAFEModification(_surplusAuctionHouse);
332:     //@audit NC - Missing event emission
333: }

```

## [N-03] Cache variable early to prevent redundant caching and an extra SLOAD

There is 1 instance of this:

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/Vault721.sol#L94C1-L99C4](https://github.com/open-dollar/od-/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/Vault721.sol#L94C1-L99C4)

On Line 97, we can see `_proxyRegistry[_proxy]` being cached to `_user`. This caching is redundant since it could anyways be inlined on Line 98 directly. But we also observe `_proxyRegistry[_proxy]` on Line 96 in the require check. This is an extra SLOAD that could've been prevented if we perform the caching done on Line 97 before the require check on Line 96.

```

File: src/contracts/proxies/Vault721.sol
94:  function mint(address _proxy, uint256 _safeId) external {
95:      require(msg.sender == address(safeManager), 'V721: only
safeManager');
96:      require(_proxyRegistry[_proxy] != address(0), 'V721: non-native
proxy');
97:      address _user = _proxyRegistry[_proxy];
98:      _safeMint(_user, _safeId);
99:  }

```

### Solution:

```

File: src/contracts/proxies/Vault721.sol
94:  function mint(address _proxy, uint256 _safeId) external {
95:      address _user = _proxyRegistry[_proxy];
96:      require(msg.sender == address(safeManager), 'V721: only
safeManager');
97:      require(_user != address(0), 'V721: non-native proxy');
98:      _safeMint(_user, _safeId);
99:  }

```

## [N-04] Remove redundant condition in `_isNotProxy()` function

There is 1 instance of this:

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/Vault721.sol#L154C1-L156C4](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/Vault721.sol#L154C1-L156C4)

The condition `ODProxy(_userRegistry[_user]).OWNER() != _user` is redundant and the first condition itself is sufficient to determine whether a user has an ODProxy or not. There's never a situation where the second condition could evaluate to true. Look at the table below:

First condition	Second condition
True	Not checked since first is true
False	False

```
File: Vault721.sol
163:   function _isNotProxy(address _user) internal view returns (bool) {
164:       return _userRegistry[_user] == address(0) ||
ODProxy(_userRegistry[_user]).OWNER() != _user;
165:   }
```

### [N-05] Remove `if (_dst == address(0)) revert ZeroAddress();` redundant check since it is already checked in `_afterTokenTransfer()` function

There is 1 instance of this issue:

<https://github.com/open-dollar/od->

[contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L139](https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L139)

Remove below check:

```
File: src/contracts/proxies/ODSafeManager.sol
139: if (_dst == address(0)) revert ZeroAddress();
```

Check already done before in `_afterTokenTransfer()` function:

```
File: Vault721.sol
206:     if (_isNotProxy(to)) {
207:         proxy = _build(to);
208:     } else {
209:         proxy = payable(_userRegistry[to]);
210:     }
```

### [N-06] Function `read()` does not revert with "OLD!" as mentioned in comments



There is 1 instance of this:

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/oracles/CamelotRelayer.sol#L91C1-L101C4>

Function does not revert with "OLD!" since there is no such revert message in the consult function.

```
File: CamelotRelayer.sol
092:  function read() external view returns (uint256 _result) {
093:      // This call may revert with 'OLD!' if the pool doesn't have
    enough cardinality or initialized history
094:      (int24 _arithmeticMeanTick,) = OracleLibrary.consult(camelotPair,
    quotePeriod);
095:      uint256 _quoteAmount = OracleLibrary.getQuoteAtTick({
096:          tick: _arithmeticMeanTick,
097:          baseAmount: baseAmount,
098:          baseToken: baseToken,
099:          quoteToken: quoteToken
100:      });
101:      _result = _parseResult(_quoteAmount);
102:  }
```

### [N-07] Remove `if (extraSurplusReceiver == address(0)) revert AccEng_NullSurplusReceiver();` redundant check

There is 1 instance of this:

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/AccountingEngine.sol#L225>

Remove check below:

```
225: if (extraSurplusReceiver == address(0)) revert
    AccEng_NullSurplusReceiver();
```

Check already **done before in the same function**:

```
201: if (extraSurplusReceiver == address(0)) revert
    AccEng_NullSurplusReceiver();
```

### [R-01] Consider modifying the build() function which allows anyone to create an ODPProxy for any user

There is 1 instance of this:

The `build()` function on Line 90 below allows anyone to create an ODProxy for any user. Although this is a convenient method, it should still implement some sort of approval mechanism where the caller can deploy an ODProxy only on the approval of the user (for whom the proxy is being created). That way the method is less prone to frontrunning attacks that could DOS the user's call (since the Proxy would already be created but user call fails when trying to build one).

```
File: Vault721.sol
81:  function build() external returns (address payable _proxy) {
82:      if (!_isNotProxy(msg.sender)) revert ProxyAlreadyExist();
83:      _proxy = _build(msg.sender);
84:  }
85:
86:  /**
87:   * @dev allows user without an ODProxy to deploy a new ODProxy
88:   */
89:  //@audit Low - allows anyone to build an ODProxy for any user
90:  function build(address _user) external returns (address payable
_proxy) {
91:      if (!_isNotProxy(_user)) revert ProxyAlreadyExist();
92:      _proxy = _build(_user);
93:  }
```

## [R-02] Use `user` instead of `usr` in mappings to improve readability

There are 2 instances of this:

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L32C2-L34C110>

Instead of `_usrSafes` use `_userSafes` and `_userSafesPerCollat` instead of `_usrSafesPerCollat`

```
File: ODSafeManager.sol
32:  mapping(address _safeOwner => EnumerableSet.UintSet) private
_usrSafes;
33:  /// @notice Mapping of user addresses to their enumerable set of
safes per collateral type
34:  mapping(address _safeOwner => mapping(bytes32 _cType =>
EnumerableSet.UintSet)) private _usrSafesPerCollat;
```

## [R-03] Use `safeAllow` and `handlerAllow` instead of `safeCan` and `handlerCan` to better match the intention of the mappings

There are 2 instances of this:

<https://github.com/open-dollar/od-contracts/blob/f4f0246bb26277249c1d5afe6201d4d9096e52e6/src/contracts/proxies/ODSafeManager.sol#L39C1-L42C1>

These mappings represent the allowances the `_owner` gives to other addresses. To suit this intention of allowance, it would be better to rename the mappings to `safeAllow` and `handlerAllow`.

```
File: ODSafeManager.sol
40:  mapping(address _owner => mapping(uint256 _safeId => mapping(address
    _caller => uint256 _ok))) public safeCan;
41:  /// @inheritdoc IOSafeManager
42:  mapping(address _safeHandler => mapping(address _caller => uint256
    _ok)) public handlerCan;
```

[R-04] Add brackets around `10 ** multiplier` to improve code readability and provide clarity in which operation takes precedence first

There is 1 instance of this:

Instead of this:

```
File: CamelotRelayer.sol
104:  function _parseResult(uint256 _quoteResult) internal view returns
    (uint256 _result) {
105:      return _quoteResult * 10 ** multiplier;
106:  }
```

Use this:

```
File: CamelotRelayer.sol
104:  function _parseResult(uint256 _quoteResult) internal view returns
    (uint256 _result) {
105:      return _quoteResult * (10 ** multiplier);
106:  }
```