BLOCKSTARS TECHNOLOGY

Smart Contract Security Audit

CLIENT:	DLTX
PROJECT:	HORSE LINK
ANALYSIS TYPE:	BASIC
VERSION:	2
START DATE:	16th January 2023

TABLE OF CONTENTS

DECLARATION

This is a Blockstars Technology's smart contract security audit report for our client DLTX and their project, Horse Link.

Results and documentation within this report contain confidential information regarding the clients' contracts.

Analysis results of the smart contracts are supplied containing lists of vulnerabilities and malicious code which could be used to malform the project. Upon the client receiving this report and until the issues are resolved or mitigated, vulnerability results that have been accumulated and listed within this report is kept private between both the Blockstars Technology auditing team and our respected client.

1. Introduction

This audit report contains confidential information of smart contract programs running in the Horse Link project. It analyses security vulnerabilities, smart contract best practices, and possible attacks using standardised automated tests using static, dynamic, and symbolic analysis tools. We outlined our systematic approach to evaluate potential security issues in core smart contracts in the Horse Link project and provide audit summary with remedies for mitigating the vulnerability findings.

2. Project Context

This section explains the client's project in detail with scope.

ITEM	DESCRIPTION
Issuer	DLTX
Website	https://horse.link/
Source	Smart contract programs
Language	Solidity
Blockchain	Ethereum (ETH)
Git Repository	https://github.com/horse-link
Audit type	BASIC
Analysis Methods	Static, Dynamic and Symbolic [Automated Analysis]
Audit Team	Pura, Faizin and Marcus
Approved By	Pura and Nilanga
Timeline	**From:** *16th Jan 2023* **To:** *17th Jan 2023*
Change logs	Version 2

3. Audit Scope

Scope of this project is to identify smart contract vulnerabilities to improve the coding practice in the given smart contract programs from Horse Link project.

- Automated testing using analysis tools which includes static, dynamic, and Symbolic analysis methods.

Blockstars Technology audit method:	**BASIC**

ITEM	DESCRIPTION
Repository	https://github.com/horse-link/contracts.horse.link
Commit Ids	d2a032ea8d6eb2207951a611f3de6e4f52f18895
Branch	Main *(default)*
Technical Documentation	Business logics provided (Readme file) https://github.com/horse-link/contracts.horse.link
Token.sol	https://github.com/horse-link/contracts.horse.link/blob/main/contracts/Token.sol
Vault.sol	https://github.com/horse-link/contracts.horse.link/blob/main/contracts/Vault.sol
Market.sol	https://github.com/horse-link/contracts.horse.link/blob/main/contracts/Market.sol
MarketCurved.sol	https://github.com/horse-link/contracts.horse.link/blob/main/contracts/MarketCurved.sol
MarketOracle.sol	https://github.com/horse-link/contracts.horse.link/blob/main/contracts/MarketOracle.sol
Registry.sol	https://github.com/horse-link/contracts.horse.link/blob/main/contracts/Registry.sol
VaultTimeLock.sol	https://github.com/horse-link/contracts.horse.link/blob/main/contracts/VaultTimeLock.sol
MarketWithRisk.sol	https://github.com/horse-link/contracts.horse.link/blob/main/contracts/MarketWithRisk.sol

4. Definitions

4.1. Security Severity

SEVERITY	DESCRIPTION
High	High level vulnerabilities are difficult to exploit, however they also have significant impact on smart contract execution due to lack of secured access control. (Example: Public access to crucial functions and data)
Medium	Medium vulnerabilities do not lead to loss of assets or data, but it is important to fix those issues.
Low	Low level vulnerabilities are related to out-dated, unused code snippets, and they don't have a significant impact on contract execution.
Informational	Does not contain vulnerabilities, but requires best practices, code standards and documentary code.

5. Audit Findings in Detail

5.1. Market.sol

5.1.1. MEDIUM

1. **Issue:** Unchecked transfer

   Function name: _back

```
L: 281          IERC20(underlying).transferFrom(_msgSender(),
                _self, wager);
```

   Description:
   ignores return value by
   IERC20(underlying).transferFrom(_msgSender(),_self,wager).
   The return value of an external transfer/transferFrom call is not checked

   Resolution:
   Use SafeERC20 or ensure that the transfer/transferFrom return value is checked.

2. **Issue:** Unchecked transfer

   Function name: _back

```
L: 282          IERC20(underlying).transferFrom(address(_vault),
                _self, (payout - wager));
```

   Description:
   ignores return value by
   IERC20(underlying).transferFrom(address(_vault),_self,(payout - wager)).
   The return value of an external transfer/transferFrom call is not checked

   Resolution:
   Use SafeERC20 or ensure that the transfer/transferFrom return value is checked.

3. **Issue:** Unchecked transfer

Function name: _payout

```
L: 350          IERC20(underlying).transfer(recipient,
                _bets[index].payout);
```

Description:
ignores return value by
IERC20(underlying).transfer(recipient,_bets[index].payout).
The return value of an external transfer/transferFrom call is not checked

Resolution:
Use SafeERC20 or ensure that the transfer/transferFrom return value is checked.

5.1.2. LOW

1. **Issue:** Reentrancy benign

Function name: _payout

Description:
Double call.
External calls:
- IERC20(underlying).transfer(recipient,_bets[index].payout) (Market.sol L: 350)

State variables written after the call(s):
- _burn(index) (Market.sol L: 351)
  - _balances[from] -= batchSize (ERC721.sol L: 475)
  - _balances[owner] -= 1 (ERC721.sol L: 335)
  - _balances[to] += batchSize (ERC721.sol L: 478)
- _burn(index) (Market.sol#351)
  - delete _owners[tokenId] (ERC721.sol L: 337)
- _burn(index) (Market.sol#351)
  - delete _tokenApprovals[tokenId] (ERC721.sol L: 330)

Resolution:
Since it calls the ERC20 contract, that is safe and well-audited one, it is safe to use external calls before check-effect.
But it is recommended to follow the check-effects-interaction pattern as a best practice while developing the contracts.

2. **Issue:** Reentrancy events

   Function name: _back

   Description:
   Out-of-order events.
   External calls:
   - IERC20(underlying).transferFrom(_msgSender(),_self,wager) (Market.sol L: 281)
   - IERC20(underlying).transferFrom(address(_vault),_self,(payout - wager)) (Market.sol L: 282)

   Event emitted after the call(s):
   - Placed(index,propositionId,marketId,wager,payout,_msgSender()) (Market.sol L: 303)
   - Transfer(address(0),to,tokenId) (ERC721.sol L: 305)
     - _mint(_msgSender(),index) (Market.sol L: 297)

   Resolution:
   Since it calls the ERC20 contract, that is safe and well-audited one, it is safe to use external calls before check-effect.
   But it is recommended to follow the check-effects-interaction pattern as a best practice while developing the contracts.
3. **Issue:** floating pragma is set

   Contract name: ERC4626Metadata

```
L: 8          pragma solidity ^0.8.0;
```

   Description:
   Use of `^` allows for the compiler to select a solidity version over 0.8.0. This is a risk, as newer versions may be released with fixes to bugs or known attacks. If the contract is not changed before this period, may result in exploitation.
   Resolution:
   Make pragma constant with a stable version.

5.1.3. INFORMATIONAL

1. **Issue:** Boolean equality

   Function name: back

```
L: 243        require(isValidSignature(messageHash, signature)
              == true, "back: Invalid signature");
```

   Description:
   compares to a Boolean constant:
   - require(bool,string)(isValidSignature(messageHash,sign ature) == true,back: Invalid signature)

   Resolution:
   Remove the equality to the Boolean constant.
   Example:
   require(isValidSignature(messageHash, signature), “back: Invalid signature");

2. **Issue:** Boolean equality

Function name: _back

```
L: 273        require(
L: 274            IOracle(_oracle).checkResult(marketId,
                  propositionId) == false,
L: 275            "back: Oracle result already set for this
                  market"
L: 276        );
```

Description:
compares to a Boolean constant:

- require(bool,string)(IOracle(_oracle).checkResult(mark etId,propositionId) == false,back: Oracle result already set for this market)

Resolution:
Remove the equality to the Boolean constant.
Example:
require(!IOracle(_oracle).checkResult(marketId, propositionId), "back: Oracle result already set for this market");

3. **Issue:** Boolean equality

Function name: settle

```
L: 310          require(bet.settled == false, "settle: Bet has
                already settled");
```

Description:
compares to a Boolean constant:

- require(bool,string)(bet.settled == false,settle: Bet has already settled)

Resolution:
Remove the equality to the Boolean constant.
require(!bet.settled, "settle: Bet has settled");

4. **Issue:** Boolean equality

Function name: _payout

```
L: 345          if (result == false) {
```

Description:
compares to a Boolean constant:

- result == false

Resolution:
Remove the equality to the Boolean constant.
Example:
if (!result) {

5. **Issue:** Boolean equality

Function name: settle

```
L: 364            if (bet.settled == false) {
```

Description:
compares to a Boolean constant:

- bet.settled == false

Resolution:
Remove the equality to the Boolean constant.
Example:
if (!bet.settled) {

Notes:

- "_margin" is never used in any of the calculations in the Market.
- "min" is never used in any of the calculations in the Market
- _back () – nonce, odds, signature are never used in the _back function
- No sanity checks for zero oracle address in the constructor.

5.2. MarketOracle.sol

No vulnerabilities detected

5.3. Vault.sol

5.3.1. MEDIUM

1. **Issue:** State variable shadowing

Function name: _decimal

```
L: 16    uint8 private immutable _decimals;
```

Description:
Detection of state variables shadowed.
shadows:

- ERC4626._decimals L: 34

Resolution:
Remove the state variable shadowing. Use different names.

2. **Issue:** Unused return

Function name: maxRedeem

```
L: 43        IERC20(asset()).approve(_market, max);
```

Description:
The return value of an external call is not stored in a local or state variable.
ignores return value by IERC20(asset()).approve(_market,max)

Resolution:
Ensure that all the return values of the function calls are used or handled.

5.3.2. LOW

1. **Issue:** Local variable shadowing

   Function name: getMarketAllowance

```
L: 70        uint256 allowance =
             ERC20(asset()).allowance(_self, _market);
```

   Description:
   Detection of shadowing using local variables.
   shadows:
   - ERC20.allowance(address,address) (ERC20.sol L: 122-124) (function)
   - IERC20.allowance(address,address) (IERC20.sol L: 50) (function)

   Resolution:
   Rename the local variables that shadow with another variable/function.

2. **Issue:** Local variable shadowing

   Function name: maxRedeem

```
L: 88    function maxRedeem(address owner) public view override
         returns (uint256) {
```

   Description:
   Detection of shadowing using local variables.
   shadows:
   - Ownable.owner() (Ownable.sol L: 43-45) (function)

   Resolution:
   Rename the local variables that shadow with another variable/function.

3. **Issue:** Missing zero address validation

Function name: setMarket

```
L: 40    function setMarket(address market, uint256 max) public
         onlyOwner {
L: 41        require(_market == address(0), "setMarket: Market
             already set");
L: 42        _market = market;
L: 43        IERC20(asset()).approve(_market, max);
L: 44    }
```

Description:
Detect missing zero address validation for function's address parameter market

Resolution:
Check that the `market` address is not zero.

5.3.3. INFORMATIONAL

1. **Issue:** Missing Inheritance

Contract name: Vault

Description:
Detect missing inheritance.

Resolution:
Vault can inherit from IVault since the interface is defined and not used.

5.4. MarketCurved.sol

No vulnerabilities detected

5.5. Registry.sol

No vulnerabilities detected

5.6. Token.sol

No vulnerabilities detected

5.7. VaultTimeLock.sol

5.7.1. LOW

1. **Issue:** Local variable shadowing

   Function name: _withdraw

   Description:
   Detection of shadowing in local variables.
   shadows:

   - Ownable.owner() (Ownable.sol L: 43-45) (function)

```
L: 45         address owner,
```

   Resolution:
   Rename the local variables that shadow with another variable/function.

5.8. ERC4626Metadata.sol

No vulnerabilities detected

5.9. HorseLink.sol

No vulnerabilities detected

5.10. IMarket.sol

No vulnerabilities detected

5.11. IMintable.sol

No vulnerabilities detected

5.12. IOracle.sol

No vulnerabilities detected

5.13. IVault.sol

No vulnerabilities detected

5.14. OddsLib.sol

5.14.1. INFORMATIONAL

1. **Issue:** Divide before multiplying

Function name: getCurvedAdjustedOdds

```
L: 37     function getCurvedAdjustedOdds(
L: 38       uint256 wager,
L: 39       uint256 odds,
L: 40       uint256 liquidity
L: 41   ) external pure returns (uint256) {
L: 42       assert(odds >= 1 * PRECISION);
L: 43       uint256 SQRT_PRECISION = 1e3;
L: 44       uint256 potentialPayout = (wager * odds /
            PRECISION);
L: 45       uint256 adjustedPayout = (liquidity + wager) -
L: 46           (liquidity * SQRT_PRECISION) /
L: 47           Math.sqrt(
L: 48               2 * (potentialPayout * PRECISION) /
                    liquidity + (1 * PRECISION),
L: 49                Math.Rounding.Up
L: 50            );
L: 51        // Return the odds need to generate this adjusted
             payout with the given wager
L: 52        return Math.max(1 * PRECISION, (adjustedPayout *
             PRECISION) / wager);
L: 53    }
```

Description:
Solidity's integer division truncates. Thus, performing division before multiplication can lead to precision loss.
performs a multiplication on the result of a division:

- potentialPayout = (wager * odds / PRECISION) (L: 44)
- adjustedPayout = (liquidity + wager) - (liquidity * SQRT_PRECISION) / Math.sqrt(2 * (potentialPayout * PRECISION) / liquidity + (1 * PRECISION),Math.Rounding.Up) (L: 45-50)

Resolution:
Consider ordering multiplication before division.
Logic that suits: potentialPayout = wager * (odds / PRECISION);

5.15. MarketWithRisk.sol

8.15.1. LOW

1. **Issue:** Boolean equality

   Function name: getCurvedAdjustedOdds

```
L: 65          require(isValidSignature(messageHash, signature)
               == true, "back: Invalid signature");
```

   Description:
   Detects the comparison to Boolean constants.
   compares to a Boolean constant:

   - require(bool,string)(isValidSignature(messageHash,sign ature) == true,back: Invalid signature)

   Resolution:
   Remove the equality to the Boolean constant.
   Example:
   require(isValidSignature(messageHash, signature), "back: Invalid signature");

5.16. Migrations.sol

No vulnerabilities detected

5.17. SignatureLib.sol

No vulnerabilities detected

5.18. TabOracle.sol

No vulnerabilities detected

6. Conclusion

The auditing team at Blockstars Technology was tasked with 18 smart contracts in total from DLTX for a **BASIC** audit evaluation.
The team has done automation tests for all the Horse Link contracts. The audit used static, dynamic, and symbolic analysis tools for reviewing each function within all the contracts.

According to this analysis, there are two high priority Horse Link contracts that require the most attention with the following vulnerabilities:

!!! **Market.sol**	3 medium	3 low	5 informational
!!! **Vault.sol**	3 medium	3 low	1 informational

There are also 3 other Horse Link contracts that require attention with the following vulnerabilities**:**

!! **OddsLib.sol**	1 medium
! **VaultTimeLock.sol**	3 low
! **MarketWithRisk.sol**	1 low

All other Horse Link contracts were analysed and contained no detected vulnerabilities according to the automation tools.

Symbolic analysis was conducted on all contracts. All contracts passed and were not found to contain Suicidal, Prodigal or Greedy instructions.

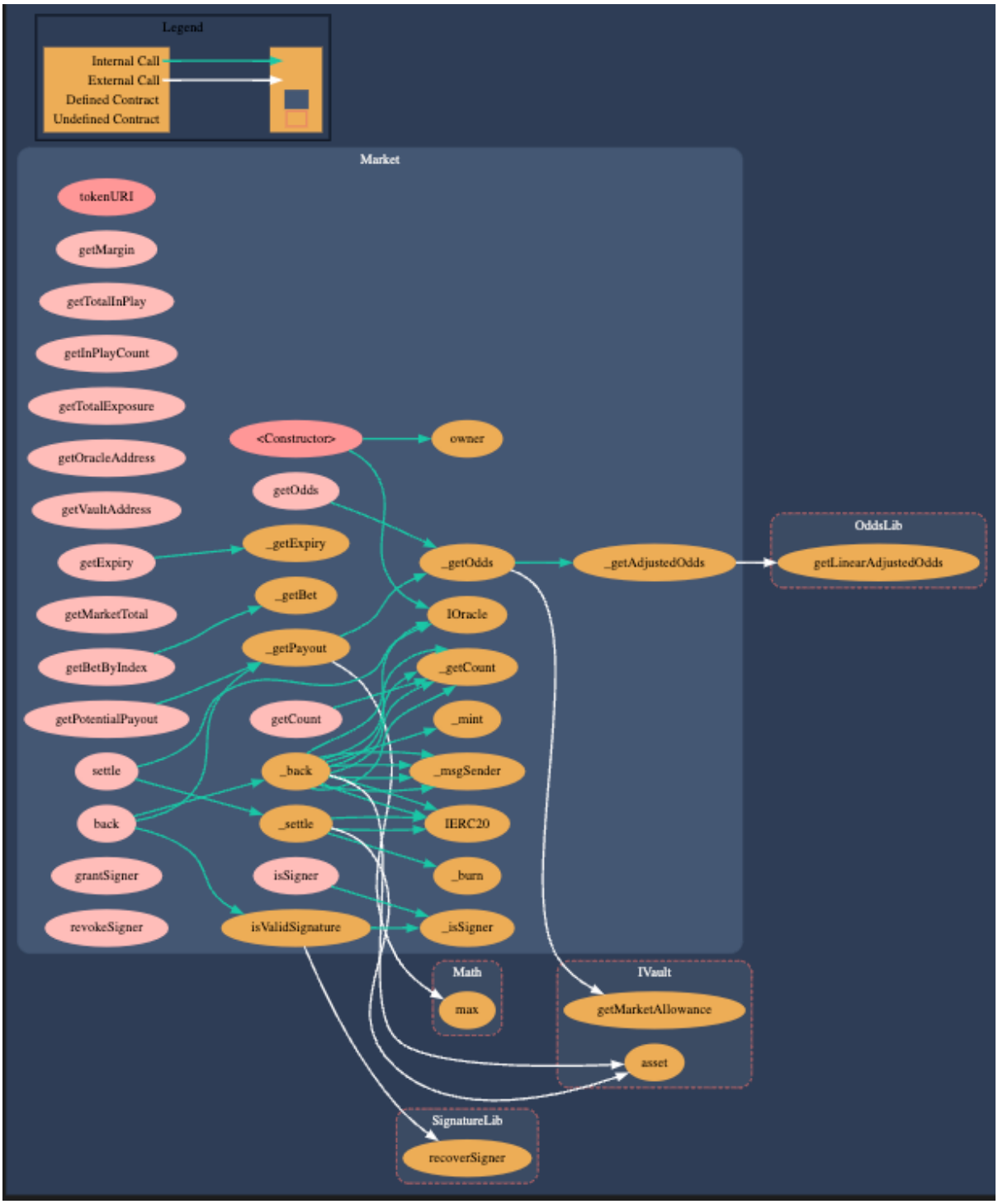
According to these findings from automated tests, the Horse Link contracts are mostly secure, with some contracts containing vulnerabilities that can cause interruption or even harm to the Horse Link project. Please focus on contracts which contain higher priorities and continue with fixing those of lower impact.

Although, it is recommended that these vulnerabilities are fixed before their deployment is committed, it is highly recommended that a DEEP analysis including a manual review of the code is completed before considering deployment.
It is NOT recommended to underestimate the low severity of less impactful vulnerabilities; If deliberately left unmaintained, up-to-date bad actors may find exploits to harm the Horse Link project.

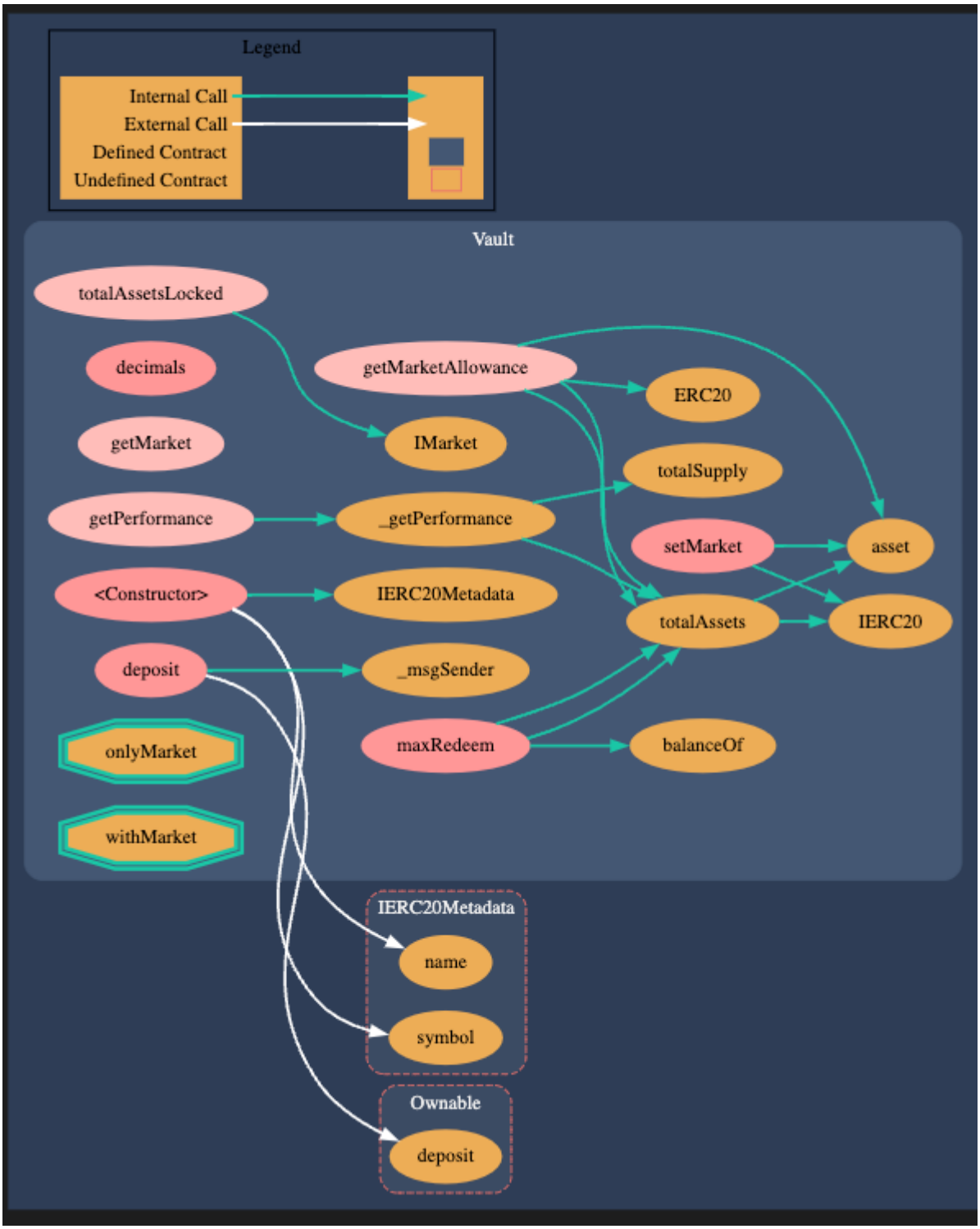
7. Appendix

7.1. Functional Flow Chart

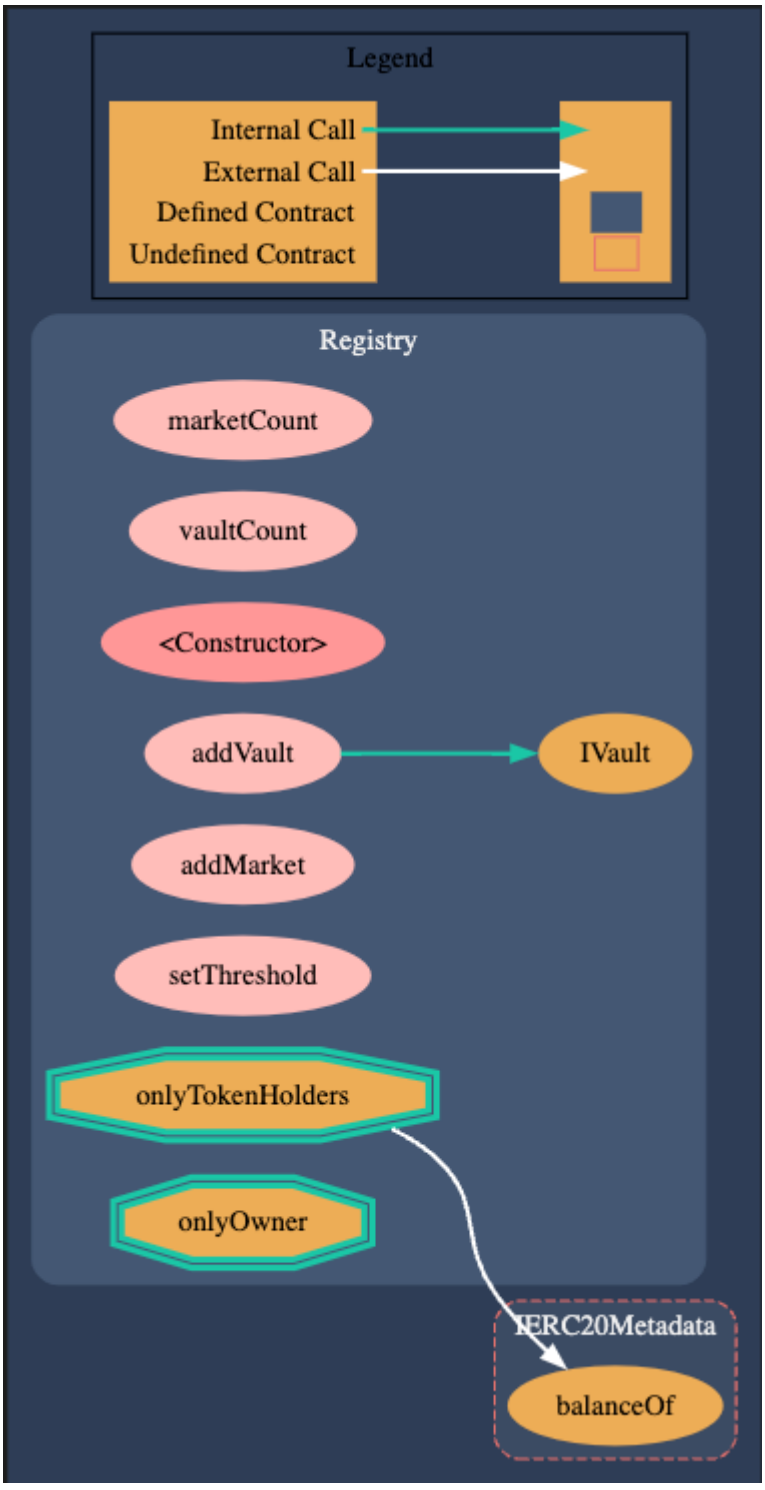
7.1.1. Market.sol



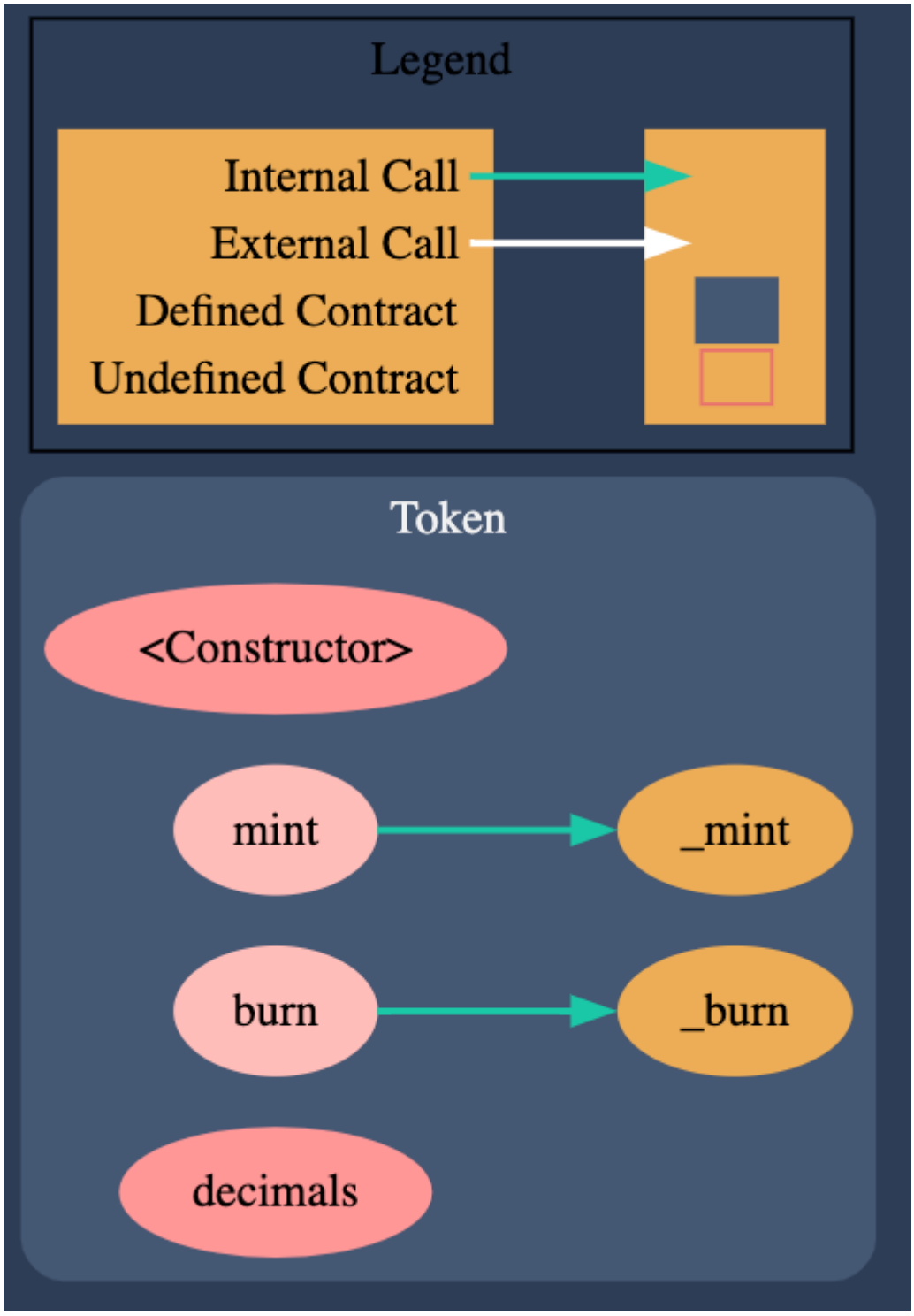
7.1.2. Vault.sol



7.1.3. Registry.sol



7.1.4. Token.sol



7.1.5. MarketOracle.sol