




SECURITY AUDIT REPORT



for project: **Raise DAO**

by auditor: **Odd Sequence**

issued: **01 December 2022**

ONE-PAGER

ABOUT

PROJECT

| | |
|-------------|--|
| Name | RaiseDAO |
| Description | RaiseDAO contracts in the Scope aim to launch fundraising campaigns with customized terms. |

AUDITOR

| | |
|-------------|-----------------|
| Team | Odd Sequence |
| Specialists | Two specialists |

TASK

| | |
|-------------------|---|
| Scope links | Github page: - page [link] - commit 4a25e8399848795ca557c69a0e1364557ea7512b [link] |
| Scope Description | Smart-contracts Transaction flow Connections with external contracts |
| Not-in-Scope | Frontend Best-Practice code remarks standardized libraries Gas efficiency (if not dangerous) |
| Networks | EVM networks |
| Languages | Solidity |
| Deployment env. | HardHat framework project |
| Timeline | bughunting: 31.10.2022 - 03.11.2022 check fixes and report: 14.11.2022 - 25.11.2022 |
| To Do | To find some vulnerabilities of the code in the Scope, that will be possible to find given the time and efforts of specialists |
| Done | The auditor has found some vulnerabilities. They are described in the report as "Findings". The report does not cover all vulnerabilities possible. |
| Terms | The service of security analysis has been purchased by the project team. |

FINDINGS

13 findings (7 fixed, 6 non-severe)

Using `[Sale]` naming for both `SaleERC20.sol` and `SaleERC1155.sol`, as contracts are quite similar. Findings in `Sale` imply that they are applicable for both contracts, if not stated otherwise.

| | |
|--|---|
| HIGH acknowledged fixed | Sale allows payTokens be anything, including tokens with dangerous design |
| MEDIUM acknowledged fixed | Sale.setIsUnhealthy() - setting <code>isHealthy_=true</code> is likely not designed |
| MEDIUM acknowledged fixed | Round.isFinal - new rounds can be pushed, even if the latest one has been set <code>isFinal=true</code> |
| MEDIUM acknowledged fixed | Factory.createSale() receives <code>projectTokenDecimals</code> as input when it is easily checked via calls |
| LOW acknowledged not fixed | SaleERC20.sol and SaleERC1155.sol have similar functionality - consider setting it aside in the separate contract and use inheritance |
| LOW downgraded from MEDIUM not acknowledged not fixed | Sale.buy() inconsistent balance calculation methodology |
| LOW acknowledged not fixed | No practical reason in using 1% rounding down in Sale |
| LOW acknowledged not fixed | No functions to manage real balances |
| LOW downgraded from HIGH acknowledged fixed | RaiseStore.sol - public control of token status |
| LOW downgraded from HIGH acknowledged not fixed | RaiseStore.buy() - arbitrary inputs and unpredictable execution |
| INFO acknowledged not fixed | projectToken design is out of control |
| INFO acknowledged fixed | Missing event for parameter changes |
| INFO acknowledged fixed | Typos |

1/1
HIGH

@ SaleERC20.sol
@ SaleERC1155.sol

Acknowledged
Fixed

Sale allows payTokens be anything, including tokens with dangerous design

Description

payToken is assumed to be pure ERC20, but some deviations are possible when the list of allowed payTokens is not limited.

Exploit Scenario One

In buy(), we have lines:

```
@ SaleERC20.sol, lines 241-246 and @ SaleERC1155.sol, lines 222-227
...
uint256 initialPayTokenBalance = payToken.balanceOf(address(this));
payToken.safeTransferFrom(msg.sender, address(this), payTokenAmount);
uint256 finalPayTokenBalance = payToken.balanceOf(address(this));
uint256 payTokenReceived = finalPayTokenBalance - initialPayTokenBalance;
...
```

Lines above can be exploited by hookable tokens. Msg.sender can receive call on payToken transfers and reenter, before token balance change is executed. initialPayTokenBalance will stay constant as before the first enter, but finalPayTokenBalance will rise with every cycle. Thus an attacker can set payTokenReceived to any amount desired.

Lines

[[link](#)], [[link](#)]

Exploit Scenario Two

In all functions assuming internal balances not changing: internal balances can break if inflationary/deflationary/rebase/interestBearing tokens are used as payTokens. As internal balances do not check the real balanceOf(this).

Recommendation

Add the list of allowed payTokens, and introduce global reentrancy protection.

Status

The team acknowledged the issue and fixed it.

Reentrancy protection has been introduced at [783ac0b99a1a596ee5accfaa8c43c6a861713d21](#) and [ac18dbcc1aa8f21ead327fc464fa0f6987d78a51](#) commit.

@ SaleERC20.sol

```
+ import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

- contract SaleERC20 is Pausable, Initializable {
+ contract SaleERC20 is Pausable, Initializable, ReentrancyGuard {

- function buy(uint256 payTokenAmount, uint8 allocationBonusPercent, bytes32[] memory proof)
  public whenNotPaused onlyHealthy {
+ function buy(uint256 payTokenAmount, uint8 allocationBonusPercent, bytes32[] memory proof)
  public whenNotPaused nonReentrant onlyHealthy {
```

@ SaleERC1155.sol

```
+ import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

- contract SaleERC1155 is Pausable, Initializable, ERC1155Holder {
+ contract SaleERC1155 is Pausable, Initializable, ERC1155Holder, ReentrancyGuard {

- function buy(uint256 nftId, uint256 payTokenAmount, uint8 allocationBonusPercent, bytes32[]
  memory proof) public whenNotPaused onlyHealthy {
+ function buy(uint256 nftId, uint256 payTokenAmount, uint8 allocationBonusPercent, bytes32[]
  memory proof) public whenNotPaused nonReentrant onlyHealthy {
```

| | |
|--|--|
| <p>1/3 MEDIUM</p> <p>@ SaleERC20.sol @ SaleERC1155.sol</p> <p>Acknowledged Fixed</p> | <p>Sale.setIsUnhealthy() - setting isHealthy_=true is likely not designed</p> <p>Description</p> <p>Contract is likely designed to not expect setting Unhealthy sales back to Healthy.</p> <pre>@ SaleERC20.sol, lines 394-401 and @ SaleERC1155.sol, lines 359-366 function setIsUnhealthy(bool isUnhealthy_) public onlyRaiseAdmin { isUnhealthy = isUnhealthy_; if (isUnhealthy) sumToRefundIfUnhealthy = totalPayTokenCollected - totalPayTokenWithdrawn; emit HealthStatusSet(isUnhealthy_); }</pre> <p>Lines</p> <p>[link], [link]</p> <p>Exploit Scenario</p> <p>For instance, imagine isHealthy set false, users can refund() some portion of payToken, their donatedPayTokenByUser will set zero. But after isHealthy is set back to true, mapping boughtProjectTokenByUser is not empty, and user can still receive projectTokens calling claim(). Other bad untested scenarios are possible.</p> <p>Recommendation</p> <p>Disable setting isUnhealthy variable from true back to false.</p> <p>Status</p> <p>The team acknowledged the issue and fixed it. Commit 18bcceb06cbd6f115a370044416c694c37dfec70 disabled setting isUnhealthy back to false.</p> <pre>For @ SaleERC20.sol - /// @param isUnhealthy_ Unhealthy status - function setIsUnhealthy(bool isUnhealthy_) public onlyRaiseAdmin { - isUnhealthy = isUnhealthy_; + function setIsUnhealthy() public onlyRaiseAdmin { + isUnhealthy = true; + sumToRefundIfUnhealthy = totalPayTokenCollected - totalPayTokenWithdrawn; + emit UnhealthStatusSet(true); For @ SaleERC1155.sol - emit HealthStatusSet(true); + emit UnhealthStatusSet(true); @ SaleERC20.sol and @ SaleERC1155.sol - event HealthStatusSet(bool isUnhealthy); + event UnhealthStatusSet(bool isUnhealthy);</pre> |
| <p>2/3 MEDIUM</p> <p>@ SaleERC20.sol</p> <p>Acknowledged Fixed</p> | <p>Round.isFinal - new rounds can be pushed, even if the latest one has been set isFinal=true</p> <p>Description</p> <p>Parameter isFinal is checked only in withdraw(). It is quite unlikely being designed, that new rounds can be pushed when the latest one is set final.</p> <pre>@ SaleERC20.sol, line 191 require(roundsLength == 0 rounds[roundsLength-1].wasStopped rounds[roundsLength-1].deadline < block.timestamp, "First stop ongoing round");</pre> |

| | |
|--|---|
| | <p>Lines [link]</p> <p>Recommendation Add additional checks for round finality.</p> <p>Status The team acknowledged the issue and fixed it. The fix at 783ac0b99a1a596ee5accfaa8c43c6a861713d21 commit checks round finality.</p> <div style="border: 1px solid black; padding: 5px;"> <pre>@ SaleERC20.sol + require(rounds.length == 0 !rounds[rounds.length-1].isFinal, "Can't add round after final one"); // Either there're no rounds or last round is not final</pre> </div> |
| <p>3/3 MEDIUM</p> <p>@ SaleFactory.sol</p> <p>Acknowledged Fixed</p> | <p>Factory.createSale() receives projectTokenDecimals as input when it is easily checked via calls</p> <p>Description There no practical reason for manual decimals inputed. Moreover, it can introduce the risk of manual accidental mistakes.</p> <div style="background-color: #e6f2ff; padding: 10px; margin: 10px 0;"> <pre>@ SaleFactory.sol, lines 34-51 function createSale(address saleOwner, SaleType saleType, address payTokenAddr, address projectTokenAddr, uint256 projectTokenDecimals, uint256 minimumAmountToFund, bool isWithdrawVestingEnabled, uint8 serviceFeePercent) public onlyOwner { address newSale = address(new ERC1967Proxy(saleContractAddresses[saleType], abi.encodeWithSignature("initialize(address,address,address,address,uint256,uint256,bool,uint8)", msg.sender, saleOwner, payTokenAddr, projectTokenAddr, 10**projectTokenDecimals, minimumAmountToFund, isWithdrawVestingEnabled, serviceFeePercent)))</pre> </div> <p>Lines [link]</p> <p>Recommendation Consider using projectTokenAddr.decimals() call</p> <p>Status The team acknowledged the issue and fixed it. Decimals is not the input since 783ac0b99a1a596ee5accfaa8c43c6a861713d21 commit.</p> <div style="border: 1px solid black; padding: 5px;"> <pre>@ SaleFactory.sol - /// @param projectTokenDecimals Decimals of project token - uint256 projectTokenDecimals, + import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol"; + uint256 projectTokenDecimals = IERC20Metadata(projectTokenAddr).decimals();</pre> </div> |

| | |
|---|---|
| <p>1/6 LOW</p> <p>@ SaleERC20.sol @ SaleERC1155.sol</p> <p>Acknowledged Not Fixed</p> | <p>SaleERC20.sol and SaleERC1155.sol have similar functionality - consider setting it aside in the separate contract and use inheritance</p> <p>Description</p> <p>Both contracts have similar functions, modifiers, variables.</p> <p>Current design implies that code corrections must be done in two contracts simultaneously. This leaves the risk of manual mistakes and risk of leaving some vulnerabilities on contracts after fixes.</p> <p>Status</p> <p>The team acknowledged the issue but did not change the contract structure, with the reason that similarities are not significant at the current moment.</p> |
| <p>2/6 LOW</p> <p>@ SaleERC20.sol</p> <p>Not Acknowledged Not Fixed</p> | <p>Sale.buy() inconsistent balance calculation methodology</p> <p>Description</p> <p>Calculation for projectTokenAmount uses payTokenAmount (minus) input within buy() call.</p> <p>Calculation for totalPayTokenCollected and donatedPayTokenByUser uses the construction balance->TransferFrom->Balance to calculated payTokens received in fact. This construction usually is used to get rid off the effects of some special tokens (like deflationary ones) but here it is implemented inconsistent and may introduce risks (one calculation for formulas, another one to be written in variables).</p> <pre> @ SaleERC20.sol, lines 241-246 uint256 initialPayTokenBalance = payToken.balanceOf(address(this)); payToken.safeTransferFrom(msg.sender, address(this), payTokenAmount); uint256 finalPayTokenBalance = payToken.balanceOf(address(this)); uint256 payTokenReceived = finalPayTokenBalance - initialPayTokenBalance; @ SaleERC20.sol, lines 220-223 function buy(uint256 payTokenAmount, uint8 allocationBonusPercent, bytes32[] memory proof) public whenNotPaused onlyHealthy { Round memory ongoingRound = getOngoingRound(); uint256 projectTokenAmount = payTokenAmount * oneProjectToken / ongoingRound.tokenPrice; </pre> <p>Lines</p> <p>[link], [link]</p> <p>Recommendation</p> <p>Add the list of allowed payTokens, with predictable behavior and that will not require checks.</p> <p>Status</p> <p>The team has not acknowledged the issue with the reason that deflationary token do not introduce exploit scenarios and losses of deflationary tokens are negligible. No code changes.</p> |
| <p>3/6 LOW</p> <p>@ SaleERC20.sol</p> <p>Acknowledged Not Fixed</p> | <p>No practical reason in using 1% rounding down in Sale</p> <p>Description</p> <p>For percentage calculation 100 is used, thus all shares are rounded to 1%. That is the risk for more precise calculations. Like in claim() function.</p> <pre> @ SaleERC20.sol, line 261 uint256 percentsWithdrawn = withdrawnProjectToken * 100 / boughtProjectToken; </pre> <p>Lines</p> <p>[link]</p> <p>Recommendation</p> <p>Consider using at least 10000 for percentage calculations.</p> |

| | |
|---|---|
| | <p>Status</p> <p>The team acknowledged the issue but did not change the percentage calculation, with the reason that more precision will not be required.</p> |
| <p>4/6 LOW</p> <p>@ SaleERC20.sol @ SaleERC1155.sol</p> <p>Acknowledged Not fixed</p> | <p>No functions to manage real balances</p> <p>Description</p> <p>Sale contracts use internal balance variable to calculate transfers, even emergency functions. It is better to check real balances using <code>balanceOf(this)</code> where possible. In additional, some tokens can stuck by mistakes. For instance, an owner can fund a Sale contract ignoring <code>fund()</code> function, and send tokens to contract directly. It is recommended when there are functions to manage such cases.</p> <p>Status</p> <p>The team acknowledged the issue but did not change the balance accounting, with the reason that managing real balances will introduce more risks and attack vectors. It is expected that users will not send tokens to contract directly (they will use contract functions).</p> |
| <p>5/6 LOW</p> <p>@ RaiseStore.sol</p> <p>Acknowledged Fixed</p> | <p>RaiseStore.sol - public control of token status</p> <p>Description</p> <p>Functions <code>whitelistTokens()</code> and <code>blacklistTokens()</code> are accessible by anyone, thus the mapping <code>whitelistedTokens</code> is out of control. Blacklisted tokens can easily leave the blacklist, the same with whitelisted ones.</p> <pre>@ RaiseStore.sol, lines 38-48 function whitelistTokens(address[] calldata tokens) public { for(uint i = 0; i < tokens.length; i++) { whitelistedTokens[tokens[i]] = true; } } function blacklistTokens(address[] calldata tokens) public { for(uint i = 0; i < tokens.length; i++) { whitelistedTokens[tokens[i]] = false; } }</pre> <p>Lines</p> <p>[link], [link]</p> <p>Status</p> <p>The finding was initially evaluated as HIGH. The team stated that the contract will not participate in the sensitive smart contract logic elsewhere and will be used for frontend calculations only. That is why the severity was decreased to LOW. Nevertheless the team fixed the issue with the <code>onlyOwner</code> modifier at <code>783ac0b99a1a596ee5accfaa8c43c6a861713d21</code> commit.</p> <pre>@ RaiseStore.sol - function whitelistTokens(address[] calldata tokens) public { + function whitelistTokens(address[] calldata tokens) public onlyOwner { - function blacklistTokens(address[] calldata tokens) public { + function blacklistTokens(address[] calldata tokens) public onlyOwner {</pre> |
| <p>6/6 LOW</p> <p>@ RaiseStore.sol</p> <p>Acknowledged Not Fixed</p> | <p>RaiseStore.buy() - arbitrary inputs and unpredictable execution</p> <p>Description</p> <p>Calldata input <code>order</code> is not checked. Thus an attacker can provide any <code>item</code> content desired.</p> <pre>@ RaiseStore.sol, lines 50-65</pre> |

| | |
|--|---|
| | <pre> function buy(UserOrder calldata order) public { for(uint256 i = 0; i < order.items.length; i++) { OrderItem memory item = order.items[i]; require(whitelistedTokens[item.payToken], "Token is not whitelisted"); _mint(msg.sender, item.itemUUID, item.amount, ""); uint256 sum = item.amount * item.itemPrice; uint256 fee = sum * serviceFeePromille / 1000; IERC20(item.payToken).safeTransferFrom(msg.sender, order.sellerAddr, sum - fee); IERC20(item.payToken).safeTransferFrom(msg.sender, address(this), fee); } emit UserBought(order.sellerAddr, order); } </pre> <p>Lines [link]</p> <p>Exploit Scenario</p> <ul style="list-style-type: none"> - mint any <code>itemUUID</code> with any <code>amount</code> to <code>msg.sender</code> - pay with any <code>payToken</code>, or set a malicious contract as <code>payToken</code> to receive calls <p>Status</p> <p>The finding was initially evaluated as HIGH. The team stated that the contract will not participate in the sensitive smart contract logic elsewhere and will be used for frontend calculations only. That is why the severity was decreased to LOW. No code changes added for this issued.</p> |
| <p>1/3 INFO</p> <p>@ SaleERC20.sol @ SaleERC1155.sol</p> <p>Acknowledged Not Fixed</p> | <p><code>projectToken</code> design is out of control</p> <p>Description</p> <p>Sale contracts do much <code>projectToken</code> transfers, but the code implementation is unpredictable. In addition, it is hardly applicable to have the list of allowed tokens.</p> <p>Recommendation</p> <p>One of the options is to demand <code>projectTokens</code> to be cloned from one single contract with the expected design.</p> <p>Status</p> <p>The team acknowledged the issue but did not changed the code, with the reason that <code>projectTokens</code> will be checked manually.</p> |
| <p>2/3 INFO</p> <p>@ SaleERC20.sol @ SaleERC1155.sol @ Staking.sol</p> <p>Acknowledged Fixed</p> | <p>Missing event for parameter changes</p> <pre> @ SaleERC20.sol, line 406 and @ SaleERC1155.sol, line 368 serviceFeePercent = newFeePercent; @ Staking.sol, lines 334, 340, 346, 353-357 pools[poolId].allocPoints = allocPoints; penaltyPercent = penaltyPercent_; raisePerBlock = newRaisePerBlock; if (tier == Tier.Fan) requiredTierStakeInfo.fan = requiredStake; else if (tier == Tier.Merchant) requiredTierStakeInfo.merchant = requiredStake; else if (tier == Tier.Dealer) requiredTierStakeInfo.dealer = requiredStake; else if (tier == Tier.Broker) requiredTierStakeInfo.broker = requiredStake; </pre> |

| | |
|--|--|
| | <pre>else if (tier == Tier.Tycoon) requiredTierStakeInfo.tycoon = requiredStake;</pre> <p>Lines [link], [link], [link], [link]</p> <p>Recommendation Consider checking functions for relevant event emitting.</p> <p>Status Info finding was fixed by the team at commits 783ac0b99a1a596ee5accfaa8c43c6a861713d21 and 18bcceb06cbd6f115a370044416c694c37dfec70</p> <div style="border: 1px solid black; padding: 10px;"> <pre>@ SaleERC20.sol and @ SaleERC1155.sol + event ServiceFeeSet(uint8 newFeePercent); + emit ServiceFeeSet(newFeePercent); @ Staking.sol + event AllocationPointsSet(uint256 poolId, uint256 allocPoints); + event PenaltyPercentSet(uint8 penaltyPercent_); + event RaisePerBlockSet(uint256 newRaisePerBlock); + emit AllocationPointsSet(poolId, allocPoints); + emit PenaltyPercentSet(penaltyPercent_); + emit RaisePerBlockSet(newRaisePerBlock);</pre> </div> |
| <p>3/3 INFO</p> <p>@ SaleERC20.sol @ SaleFactory.sol</p> <p>Acknowledged Fixed</p> | <p>Typos</p> <div style="background-color: #e6f2ff; padding: 10px;"> <pre>@ SaleERC20.sol, lines 166, 167 Maximum - Maximum /// @param maxAllocation Maximum amount of project tokens can be bought in the round /// @param maxAllocationPerUser Maximum amount of project tokens can be bought be one user in the round @ SaleFactory.sol, line 60 implemetation - implementation /// @notice Updates sale contract implemetation</pre> </div> <p>Lines [link], [link], [link]</p> <p>Info finding was fixed by team in 783ac0b99a1a596ee5accfaa8c43c6a861713d21 commit</p> <div style="border: 1px solid black; padding: 10px;"> <pre>@ SaleERC20.sol - /// @param maxAllocation Maximum amount of project tokens can be bought in the round - /// @param maxAllocationPerUser Maximum amount of project tokens can be bought be one user in the round + /// @param maxAllocation Maximum amount of project tokens can be bought in the round + /// @param maxAllocationPerUser Maximum amount of project tokens can be bought be one user in the round @ SaleFactory.sol - /// @notice Updates sale contract implemetation + /// @notice Updates sale contract implementation</pre> </div> |

CODE IN THE SCOPE

GITHUB

| | |
|-----------|---|
| Link | https://github.com/RaiseDAO/contracts/tree/4a25e8399848795ca557c69a0e1364557ea7512b |
| Contracts | RaiseStore.sol SaleERC20.sol SaleERC1155.sol SaleFactory.sol Staking.sol |

REPORT CHANGELOG

The first report version was published on HackMD:
<https://hackmd.io/@iZKFKn8zRW-F0K4IQ8L-eQ/SkwYpWqHj>
 The later versions are published on Odd Sequence Github page, where changes can be tracked:
<https://github.com/oddsequence>

FINDINGS STATUS

HIGH

| | |
|-----|-------------------------------------|
| 1/1 | Acknowledged, Fixed, then Published |
|-----|-------------------------------------|

MEDIUM

| | |
|-----|-------------------------------------|
| 1/3 | Acknowledged, Fixed, then Published |
| 2/3 | Acknowledged, Fixed, then Published |
| 3/3 | Acknowledged, Fixed, then Published |

LOW

| | |
|-----|--|
| 1/6 | Acknowledged, Not Fixed, but Published |
| 2/6 | Not acknowledged, Not fixed, but Published |
| 3/6 | Acknowledged, Not Fixed, but Published |
| 4/6 | Acknowledged, Not Fixed, but Published |
| 5/6 | Acknowledged, Fixed, then Published |
| 6/6 | Acknowledged, Not Fixed, but Published |

INFO

| | |
|-----|--|
| 1/3 | Acknowledged, Not Fixed, but Published |
| 2/3 | Acknowledged, Fixed, then Published |
| 3/3 | Acknowledged, Fixed, then Published |

GENERAL DISCLOSURES

Odd Sequence typically receives compensation from one or more clients (the "Clients") for performing the analysis contained in these reports (the "Reports"). The Reports may be distributed through other means, including via Github, website mainpage and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. Odd Sequence owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS

The Reports and the analysis described therein are created solely for Clients and published with their consent. This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any code itself presents unique and unquantifiable risks as the programming languages itself remain under development and is subject to unknown risks and flaws. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

Odd Sequence makes the Reports available to parties other than the Clients (i.e., "third parties") – on its website. Odd Sequence hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Odd Sequence. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Odd Sequence is not responsible for the content or operation of such Web sites, and that Odd Sequence shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that Odd Sequence endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. Odd Sequence assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Odd Sequence; however, Odd Sequence does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

NOTICE OF CONFIDENTIALITY

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Odd Sequence. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Odd Sequence. Odd Sequence assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software. In some instances, Odd Sequence may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.