METATRUST

Pre Report for

QAMarketplace

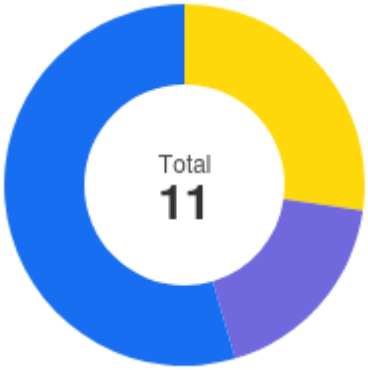
March 17, 2025

Executive Summary

Overview	
Project Name	QAMarketplace
Codebase URL	https://github.com/XNect/QAMarketplace
Scan Engine	Security Analyzer
Scan Time	2025/03/17 08:00:00
Commit Id	d4781559d60ea3459279183fc4f5227ea6d356e9

Total	
Critical Issues	0
High risk Issues	0
Medium risk Issues	3
Low risk Issues	2
Informational Issues	6

Critical Issues	The issue can cause large economic losses, large-scale data disorder, loss of control of authority management, failure of key functions, or indirectly affect the correct operation of other smart contracts interacting with it.
High Risk Issues	The issue puts a large number of users' sensitive information at risk or is reasonably likely to lead to catastrophic impacts on clients' reputations or serious financial implications for clients and users.
Medium Risk Issues	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk Issues	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational Issue	The issue does not pose an immediate risk but is relevant to security best practices or Defence in Depth.



Critical Issues	0%	0
High risk Issues	0%	0
Medium risk Issues	27%	3
Low risk Issues	18%	2
Informational Issues	55%	6

Summary of Findings

MetaScan security assessment was performed on **March 17, 2025 08:00:00** on project **QAMarketplace** with the repository on branch **default branch**. The assessment was carried out by scanning the project's codebase using the scan engine **Security Analyzer**. There are in total **11** vulnerabilities / security risks discovered during the scanning session, among which **3** medium risk vulnerabilities, **2** low risk vulnerabilities, **6** informational issues.

ID	Description	Severity
MSA-001	Discussion: Should users pay for eliminated or resolved question?	Medium risk
MSA-002	The `register()` function missing validate the `uidToAddress`	Medium risk
MSA-003	The `submitQuestion()` function missing check the `_questionId`, `_askerId`, and `_answererId`	Medium risk
MSA-004	The centralized function `withdraw()` may result in the contract malfunctional	Low risk
MSA-005	The `answererEarnings` for un-registered users would be greater than exptected	Low risk
MSA-006	Missing the sanity check	Informational
MSA-007	The `answererViewRewardPercentage` is unused	Informational
MSA-008	Duplicated events emitted	Informational
MSA-009	Centralization risk	Informational
MSA-010	Consider updating the fee parameter in the same function	Informational
MSA-011	Missing invoke `__ReentrancyGuard_init()` for the upgradeable contract	Informational

Findings

Medium risk (3)

1. Discussion: Should users pay for eliminated or resolved question?

Medium risk Security Analyzer

The `viewQuestion()` function charges users for the specified question, the discussion point is that, is it an intended design for this function charges users for those eliminated or resolved questions?

Is it an intended design for this function charges users for those ongoing question?

Note that this function does not check question status, so any question viewed in this function will charge users fee, but some of them maybe eliminated or onging.

File(s) Affected

contracts/QAMarketplace.sol #305-331

```
305     function viewQuestion(
306         uint256 _questionId,
307         string calldata _viewerId
308     ) external payable whenNotPaused nonReentrant {
309         QASession memory q = questions[_questionId];
310         require(q.paymentAddress != address(0), "Question does not exist");
311 
312         uint256 reward = (q.reward * viewRewardPercentage) / 100;
313         require(msg.value >= reward, "Value must be greater than viewReward");
314 
315         uint256 fee = (msg.value * viewFee) / 100; // 10% fee
316         uint256 rewardToAsker = (msg.value * askerViewRewardPercentage) / 100; // 50% to asker
317         uint256 rewardToAnswerer = msg.value - rewardToAsker - fee; // 40% to answerer
318 
319         _rewardAnswerer(q.answererId, rewardToAnswerer);
320         _rewardAsker(q.askerId, q.paymentAddress, rewardToAsker);
321 
322         emit QuestionViewed(
323             _questionId,
324             _viewerId,
325             q.askerId,
326             q.answererId,
327             rewardToAsker,
328             rewardToAnswerer,
329             fee
330         );
331     }
```

2. The `register()` function missing validate the `uidToAddress`

Medium risk Security Analyzer

The `register()` function can register users' wallet addresses and user ids with the signature generated by the server:

```
    function register(
        string calldata uid,
        address user,
        uint256 expirationTime,
        bytes calldata signature
    ) external whenNotPaused {
        require(!registeredAddresses[user], "Address Already Used");

        // store the mapping relationship
        addressToUid[user] = uid;
        uidToAddress[uid] = user; //@here
        registeredAddresses[user] = true;
        registeredUIds[uid] = true;
```

However, the **`register`** function only check if the wallet address is registered or not, lack of checking if the user id, **`uid`**, is registered or not.

Thus, two users can be registered with the same **`uid`**, which would results in unexpected results.

Example:

```
//first call to register wallet address 0x1 with uid 1
addressToUid[0x1] = 1
uidToAddress[1] = 0x1
//second call to register wallet address 0x2 with uid 1
addressToUid[0x2] = 1
uidToAddress[1] = 0x2
//as a result, there is no uid maps to the address 0x1
```

File(s) Affected

contracts/QAMarketplace.sol #130-173

```
130     function register(
131         string calldata uid,
132         address user,
133         uint256 expirationTime,
134         bytes calldata signature
135     ) external whenNotPaused {
136         // check if the address is already registered
137         require(!registeredAddresses[user], "Address Already Used");
138 
139         // check if the expierTime is greater than current timestamp
140         require(
141             expirationTime > block.timestamp,
142             "ExpirationTime must be greater than current timestamp"
143         );
144 
145         // get the eth signed message hash
146         bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash(
147             abi.encode(uid, user, expirationTime)
148         );
149         // recover the signer address from the signature
150         address signer = ECDSA.recover(ethSignedMessageHash, signature);
151 
152         // check if the signer is the authorized server address
153         require(signer == server, "Invalid Signature");
154 
155         // // check if the user is the same as the msg.sender
156         // require(user == msg.sender, "Address is not valid");
157 
158         // store the mapping relationship
159         addressToUid[user] = uid;
160         uidToAddress[uid] = user;
161         registeredAddresses[user] = true;
162         registeredUIds[uid] = true;
163 
164         // claim the reward if there is any
165         uint256 reward = pendingRewards[uid];
166         if (reward > 0) {
167             pendingRewards[uid] = 0;
168             _rewardAnswerer(uid, reward);
169             emit AnswererRewarded(uid, user, reward);
170         }
171 
172         emit Registered(uid, user);
173     }
```

Recommendation

Consider validating if the user id, **`uid`**, registered or not.

3. The `submitQuestion()` function missing check the `_questionId`, `_askerId`, and `_answererId`

Medium risk Security Analyzer

The **`submitQuestion()`** function submits question that contains the **`_questionId`**, **`_askerId`**, and **`_answererId`**:

```
function submitQuestion(
    uint256 _questionId,
    string calldata _askerId,
    string calldata _answererId,
    string calldata _questionContent,
    uint256 _minReward,
    uint256 _expiryTimestamp,
    bytes calldata _signature
) external payable whenNotPaused {
...
    QASession memory q = QASession({
        id: _questionId,  //@here
        askerId: _askerId,
        answererId: _answererId,
        questionContent: _questionContent,
        reward: msg.value,
        paymentAddress: msg.sender,
        resolved: false,
        terminated: false,
        creationTimestamp: block.timestamp,
        expiryTimestamp: _expiryTimestamp
    });
    questions[_questionId] = q;
```

However, the function does not check the **`_questionId`**, **`_askerId`**, and **`_answererId`**. As a result, the non-exist user id may be used as the **`_askerId`** or the **`_answererId`**, which results in the fund loss.

The question id, **`_questionId`**, is not checked, so, the same question id may be used for the different question and results in the old question malfunctional.

File(s) Affected

contracts/QAMarketplace.sol #175-239

```
175     function submitQuestion(
176         uint256 _questionId,
177         string calldata _askerId,
178         string calldata _answererId,
179         string calldata _questionContent,
180         uint256 _minReward,
181         uint256 _expiryTimestamp,
182         bytes calldata _signature
183     ) external payable whenNotPaused {
184         require(
185             msg.value >= MIN_REWARD,
186             "Reward must be greater than MIN_REWARD"
187         );
188         require(
189             _expiryTimestamp > block.timestamp,
190             "ExpirationTime must be greater than current timestamp"
191         );
192         require(
193             bytes(_questionContent).length <= 2000,
194             "Question must be greater than 0 and less than 2000 characters"
195         );
196 
197         // check if the signature is valid
198         bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash(
199             abi.encode(
200                 _questionId,
201                 _askerId,
202                 _answererId,
203                 _questionContent,
204                 _minReward,
205                 _expiryTimestamp
206             )
207         );
208         address signer = ECDSA.recover(ethSignedMessageHash, _signature);
209         require(signer == server, "Invalid Signature");
210 
211         // check if the reward is greater than the minAnswerReward
212         require(
213             msg.value >= _minReward,
214             "Reward must be greater than minReward"
215         );
216 
217         QASession memory q = QASession({
218             id: _questionId,
219             askerId: _askerId,
220             answererId: _answererId,
221             questionContent: _questionContent,
222             reward: msg.value,
223             paymentAddress: msg.sender,
224             resolved: false,
225             terminated: false,
226             creationTimestamp: block.timestamp,
227             expiryTimestamp: _expiryTimestamp
228         });
229         questions[_questionId] = q;
230 
231         emit QuestionSubmitted(
```

```
232             _questionId,
233             _askerId,
234             _answererId,
235             msg.sender,
236             msg.value,
237             _expiryTimestamp
238         );
239     }
```

Recommendation

The `_askerId` and the `_answererId` should be exist and the `_questionId` should not be used before.

Low risk (2)

1. The centralized function `withdraw()` may result in the contract malfunctional

Low risk Security Analyzer

The `withdraw()` function allows the owner withdraw all the native token to a specified address. The point is that what if there are still some user rewards pending to be distributed for askers and answerers, if the owner withdraws all the native tokens, then, the contract can not work as expected due to lack of funds to distribute reward.

File(s) Affected

contracts/QAMarketplace.sol #333-334

```
333     function withdraw(address to) external onlyOwner {
334         _safeTransfer(to, address(this).balance);
```

Recommendation

When a question is processed or viewed, there is a fee charged for the owner. Accumulating the fee into a variable, like `totalFee`, and only withdraw fee under the `totalFee`.

2. The `answererEarnings` for un-registered users would be greater than exptected

Low risk Security Analyzer

The `_rewardAnswerer()` function records users' reward into the `pendingRewards[_uid]` if an user id is un-registered:

```
function _rewardAnswerer(string memory _uid, uint256 _reward) internal {
    answererEarnings[_uid] += _reward;    //@here
    address answererAddress = uidToAddress[_uid];
    if (answererAddress != address(0)) {
        _safeTransfer(answererAddress, _reward);
    } else {
        pendingRewards[_uid] += _reward;    //@here
    }
    emit AnswererRewarded(_uid, answererAddress, _reward);
}
```

Once the user id is registered, the pending reward will be distributed to the user soon:

```
function register(
    string calldata uid,
    address user,
    uint256 expirationTime,
```

```
        bytes calldata signature
    ) external whenNotPaused {
        ...
        // store the mapping relationship
        addressToUid[user] = uid;
        uidToAddress[uid] = user;

        // claim the reward if there is any
        uint256 reward = pendingRewards[uid];  //@here
        if (reward > 0) {
            pendingRewards[uid] = 0;
            _rewardAnswerer(uid, reward); //@here
            emit AnswererRewarded(uid, user, reward);
        }
        ...
    }
```

However, the register function calls the **`_rewardAnswerer()`** function for an un-registered user will repeatedly increase the **`answererEarnings`**, which results in the **`answererEarnings`** for the un-registered users being greater than expected once the users registered later.

File(s) Affected

contracts/QAMarketplace.sol #130-173

```
130     function register(
131         string calldata uid,
132         address user,
133         uint256 expirationTime,
134         bytes calldata signature
135     ) external whenNotPaused {
136         // check if the address is already registered
137         require(!registeredAddresses[user], "Address Already Used");
138 
139         // check if the expierTime is greater than current timestamp
140         require(
141             expirationTime > block.timestamp,
142             "ExpirationTime must be greater than current timestamp"
143         );
144 
145         // get the eth signed message hash
146         bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash(
147             abi.encode(uid, user, expirationTime)
148         );
149         // recover the signer address from the signature
150         address signer = ECDSA.recover(ethSignedMessageHash, signature);
151 
152         // check if the signer is the authorized server address
153         require(signer == server, "Invalid Signature");
154 
155         // // check if the user is the same as the msg.sender
156         // require(user == msg.sender, "Address is not valid");
157 
158         // store the mapping relationship
159         addressToUid[user] = uid;
160         uidToAddress[uid] = user;
161         registeredAddresses[user] = true;
162         registeredUIds[uid] = true;
163 
164         // claim the reward if there is any
165         uint256 reward = pendingRewards[uid];
166         if (reward > 0) {
167             pendingRewards[uid] = 0;
168             _rewardAnswerer(uid, reward);
169             emit AnswererRewarded(uid, user, reward);
170         }
171 
172         emit Registered(uid, user);
173     }
```

contracts/QAMarketplace.sol #337-346

```
337     function _rewardAnswerer(string memory _uid, uint256 _reward) internal {
338         answererEarnings[_uid] += _reward;
339         address answererAddress = uidToAddress[_uid];
340         if (answererAddress != address(0)) {
341             _safeTransfer(answererAddress, _reward);
342         } else {
343             pendingRewards[_uid] += _reward;
344         }
345         emit AnswererRewarded(_uid, answererAddress, _reward);
346     }
```

Recommendation

In the `_rewardAnswerer()` function, consider only increasing the `answererEarnings` for the specified address when the `answererAddress` is not zero address, i.e., the user registered.

Informational (6)

1. Missing the sanity check

Informational Security Analyzer

The `initialize()` function and the `setServer()` function set the key state variable `server` but missing the sanity check. It is responsible to checking the signature and is important.

File(s) Affected

contracts/QAMarketplace.sol #111-113

```
111     function initialize(address _server) external initializer {
112         __Ownable_init(msg.sender);
113         server = _server;
```

contracts/QAMarketplace.sol #375-377

```
375     function setServer(address _server) external onlyOwner {
376         server = _server;
377     }
```

Recommendation

Consider adding the non-zero check for the `server` and other state variables.

2. The `answererViewRewardPercentage` is unused

Informational Security Analyzer

The state variable `answererViewRewardPercentage` is declared and assigned to be 40, in the `_setDefaultParameters()` function, but it is never used.

File(s) Affected

contracts/QAMarketplace.sol #126-126

```
126         answererViewRewardPercentage = 40;
```

Recommendation

Consider removing the unused variable `answererViewRewardPercentage` to save gas.

3. Duplicated events emitted

Informational Security Analyzer

The `register()` function calls the `_rewardAnswerer()` function if the `reward` is greater than 0:

```
function register(
    string calldata uid,
    address user,
    uint256 expirationTime,
    bytes calldata signature
) external whenNotPaused {
...
    if (reward > 0) {
        pendingRewards[uid] = 0;
        _rewardAnswerer(uid, reward);
        emit AnswererRewarded(uid, user, reward);//@here
    }


function _rewardAnswerer(string memory _uid, uint256 _reward) internal {
    answererEarnings[_uid] += _reward;
    address answererAddress = uidToAddress[_uid];
    ...
    emit AnswererRewarded(_uid, answererAddress, _reward);//@here
}
```

As the above codes shown, the event `AnswererRewarded` is emitted twice outer and inner the `_rewardAnswerer()` function.

File(s) Affected

contracts/QAMarketplace.sol #130-173

```
130     function register(
131         string calldata uid,
132         address user,
133         uint256 expirationTime,
134         bytes calldata signature
135     ) external whenNotPaused {
136         // check if the address is already registered
137         require(!registeredAddresses[user], "Address Already Used");
138 
139         // check if the expierTime is greater than current timestamp
140         require(
141             expirationTime > block.timestamp,
142             "ExpirationTime must be greater than current timestamp"
143         );
144 
145         // get the eth signed message hash
146         bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash(
147             abi.encode(uid, user, expirationTime)
148         );
149         // recover the signer address from the signature
150         address signer = ECDSA.recover(ethSignedMessageHash, signature);
151 
152         // check if the signer is the authorized server address
153         require(signer == server, "Invalid Signature");
154 
155         // // check if the user is the same as the msg.sender
156         // require(user == msg.sender, "Address is not valid");
157 
158         // store the mapping relationship
159         addressToUid[user] = uid;
160         uidToAddress[uid] = user;
161         registeredAddresses[user] = true;
162         registeredUIds[uid] = true;
163 
164         // claim the reward if there is any
165         uint256 reward = pendingRewards[uid];
166         if (reward > 0) {
167             pendingRewards[uid] = 0;
168             _rewardAnswerer(uid, reward);
169             emit AnswererRewarded(uid, user, reward);
170         }
171 
172         emit Registered(uid, user);
173     }
```

Recommendation

Only emitting the event from the `_rewardAnswerer()` function.

4. Centralization risk

Informational Security Analyzer

In the `QAMarketplace` contract, the owner has the privilege of the following functions:

- `setMinReward`: Set the minimum reward amount for submitting a question;
- `setQuestionFee`: Set the fee percentage for processing a question.

- **`setRefundFee`**: Set the fee percentage for processing an expired question.
- **`setServer`**: Set the authorized server address.
- **`setViewFee`**: Set the fee percentage for viewing a question.
- **`setAskerRefundPercentage`**: Set the percentage of refund amount for the asker in case of an expired question.
- **`setAnswererRefundPercentage`**: Set the percentage of refund amount for the answerer in case of an expired question.
- **`setViewRewardPercentage`**: Set the percentage of reward amount for viewing a question.

File(s) Affected

contracts/QAMarketplace.sol #363-407

```
363     function setMinReward(uint256 _minReward) external onlyOwner {
364         MIN_REWARD = _minReward;
365     }
366
367     function setQuestionFee(uint8 _questionFee) external onlyOwner {
368         questionFee = _questionFee;
369     }
370
371     function setRefundFee(uint8 _refundFee) external onlyOwner {
372         refundFee = _refundFee;
373     }
374
375     function setServer(address _server) external onlyOwner {
376         server = _server;
377     }
378
379     function setViewFee(uint8 _viewFee) external onlyOwner {
380         viewFee = _viewFee;
381     }
382
383     function setAskerRefundPercentage(
384         uint8 _askerRefundPercentage
385     ) external onlyOwner {
386         askerRefundPercentage = _askerRefundPercentage;
387     }
388
389     function setAnswererRefundPercentage(
390         uint8 _answererRefundPercentage
391     ) external onlyOwner {
392         answererRefundPercentage = _answererRefundPercentage;
393     }
394
395     function setViewRewardPercentage(
396         uint8 _viewRewardPercentage
397     ) external onlyOwner {
398         viewRewardPercentage = _viewRewardPercentage;
399     }
400
401     function pause() external onlyOwner {
402         paused = true;
403     }
404
405     function unpause() external onlyOwner {
406         paused = false;
407     }
```

Recommendation

Consider implementing a decentralized governance mechanism or a multi-signature scheme that requires consensus among multiple parties before pausing or unpausing the contract. This can help mitigate the centralization risk associated with a single owner controlling critical contract functions. Alternatively, you can provide a clear justification for the centralization aspect and ensure that users are aware of the potential risks associated with a single point of control.

5. Consider updating the fee parameter in the same function

Informational Security Analyzer

The sum of the `refundFee` and the `askerRefundPercentage` should not be greater than 100.

The sum of the `viewFee` and the `askerViewRewardPercentage` should not be greater than 100.

File(s) Affected

contracts/QAMarketplace.sol #278-290

```
278     function processExpiredQuestion(uint256 _questionId) public whenNotPaused {
279         QASession storage q = questions[_questionId];
280         require(!q.resolved, "Question is answered");
281         require(!q.terminated, "Question is already canceled");
282         require(
283             q.expiryTimestamp <= block.timestamp,
284             "Question is not expired"
285         );
286 
287         q.terminated = true;
288         uint256 fee = (q.reward * refundFee) / 100; // 10% fee
289         uint256 refundValue = (q.reward * askerRefundPercentage) / 100; // 45% to asker
290         uint256 rewardToAnswerer = q.reward - fee - refundValue; // 45% to answerer
```

contracts/QAMarketplace.sol #305-317

```
305     function viewQuestion(
306         uint256 _questionId,
307         string calldata _viewerId
308     ) external payable whenNotPaused nonReentrant {
309         QASession memory q = questions[_questionId];
310         require(q.paymentAddress != address(0), "Question does not exist");
311 
312         uint256 reward = (q.reward * viewRewardPercentage) / 100;
313         require(msg.value >= reward, "Value must be greater than viewReward");
314 
315         uint256 fee = (msg.value * viewFee) / 100; // 10% fee
316         uint256 rewardToAsker = (msg.value * askerViewRewardPercentage) / 100; // 50% to asker
317         uint256 rewardToAnswerer = msg.value - rewardToAsker - fee; // 40% to answerer
```

Recommendation

Consider updating the `refundFee` and the `askerRefundPercentage` in the same centralized function and make the sum of them no greater than 100, and updating the `viewFee` and the `askerViewRewardPercentage` in the same centralized function and make the sum of them no greater than 100.

6. Missing invoke `__ReentrancyGuard_init()` for the upgradeable contract

Informational Security Analyzer

The `QAMarketplace` contract inherits the `ReentrancyGuardUpgradeable` contract but its `initialize()` function missing invoke the `__ReentrancyGuard_init()` function of the `ReentrancyGuardUpgradeable` contract.

```
contract QAMarketplace is
    Initializable,
    OwnableUpgradeable,
    ReentrancyGuardUpgradeable //@here
    ...
    function initialize(address _server) external initializer {
        __Ownable_init(msg.sender);
        server = _server;
        paused = false;
        _setDefaultParameters();
    }

    //contracts/utils/ReentrancyGuardUpgradeable.sol
    function __ReentrancyGuard_init() internal onlyInitializing {
        __ReentrancyGuard_init_unchained();
    }

    function __ReentrancyGuard_init_unchained() internal onlyInitializing {
        ReentrancyGuardStorage storage $ = _getReentrancyGuardStorage();
        $._status = NOT_ENTERED;
    }
```

File(s) Affected

contracts/QAMarketplace.sol #111-116

```
111     function initialize(address _server) external initializer {
112         __Ownable_init(msg.sender);
113         server = _server;
114         paused = false;
115         _setDefaultParameters();
116     }
```

contracts/QAMarketplace.sol #10-13

```
10  contract QAMarketplace is
11      Initializable,
12      OwnableUpgradeable,
13      ReentrancyGuardUpgradeable
```

Recommendation

Consider invoking the `__ReentrancyGuard_init()` function from the `QAMarketplace` contract's `initialize()` function.

Audit Scope

File	SHA256	File Path
QAMarketplace-main/contracts/QAMarketplace.sol	9ca573d9b3144055ec09fad0bfaf46746af89532e36703e2dbad3f9cc4263c65	/QAMarketplace-main/contracts/QAMarketplace-main/contracts/QAMarketplace.sol

Disclaimer

This report is governed by the stipulations (including but not limited to service descriptions, confidentiality, disclaimers, and liability limitations) outlined in the Services Agreement, or as detailed in the scope of services and terms provided to you, the Customer or Company, within the context of the Agreement. The Company is permitted to use this report only as allowed under the terms of the Agreement. Without explicit written permission from MetaTrust, this report must not be shared, disclosed, referenced, or depended upon by any third parties, nor should copies be distributed to anyone other than the Company.

It is important to clarify that this report neither endorses nor disapproves any specific project or team. It should not be viewed as a reflection of the economic value or potential of any product or asset developed by teams or projects engaging MetaTrust for security evaluations. This report does not guarantee that the technology assessed is completely free of bugs, nor does it comment on the business practices, models, or legal compliance of the technology's creators.

This report is not intended to serve as investment advice or a tool for investment decisions related to any project. It represents a thorough assessment process aimed at enhancing code quality and mitigating risks inherent in cryptographic tokens and blockchain technology. Blockchain and cryptographic assets inherently carry ongoing risks. MetaTrust's role is to support companies and individuals in their security diligence and to reduce risks associated with the use of emerging and evolving technologies. However, MetaTrust does not guarantee the security or functionality of the technologies it evaluates.

MetaTrust's assessment services are contingent on various dependencies and are continuously evolving. Accessing or using these services, including reports and materials, is at your own risk, on an as-is and as-available basis. Cryptographic tokens are novel technologies with inherent technical risks and uncertainties. The assessment reports may contain inaccuracies, such as false positives or negatives, and unpredictable outcomes. The services may rely on multiple third-party layers.

All services, labels, assessment reports, work products, and other materials, or any results from their use, are provided "as is" and "as available," with all faults and defects, without any warranty. MetaTrust expressly disclaims all warranties, whether express, implied, statutory, or otherwise, including but not limited to warranties of merchantability, fitness for a particular purpose, title, non-infringement, and any warranties arising from course of dealing, usage, or trade practice. MetaTrust does not guarantee that the services, reports, or materials will meet specific requirements, be error-free, or be compatible with other software, systems, or services.

Neither MetaTrust nor its agents make any representations or warranties regarding the accuracy, reliability, or currency of any content provided through the services. MetaTrust is not liable for any content inaccuracies, personal injuries, property damages, or any loss resulting from the use of the services, reports, or materials.

Third-party materials are provided "as is," and any warranty concerning them is strictly between the Customer and the third-party owner or distributor. The services, reports, and materials are intended solely for the Customer and should not be relied upon by others or shared without MetaTrust's consent. No third party or representative thereof shall have any rights or claims against MetaTrust regarding these services, reports, or materials.

The provisions and warranties of MetaTrust in this agreement are exclusively for the Customer's benefit. No third party has any rights or claims against MetaTrust regarding these provisions or warranties. For clarity, the services, including any assessment reports or materials, should not be used as financial, tax, legal, regulatory, or other forms of advice.