METATRUST

Security Assessment for

DGRID

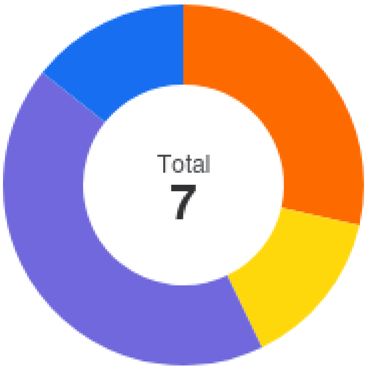
August 08, 2025

Executive Summary

Overview	
Project Name	DGRID
Codebase URL	https://github.com/dgridai/DGRID/tree/main
Scan Engine	Security Analyzer
Scan Time	2025/08/08 08:00:00
Commit Id	b41eb74072836c4b3e5a8c19221a50af9ebfe37f

Total	
Critical Issues	0
High risk Issues	2
Medium risk Issues	1
Low risk Issues	3
Informational Issues	1

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	29%	2
Medium risk Issues	14%	1
Low risk Issues	43%	3
Informational Issues	14%	1

Summary of Findings

MetaScan security assessment was performed on **August 08, 2025 08:00:00** on project **DGRID** 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 **7** vulnerabilities / security risks discovered during the scanning session, among which **2** high risk vulnerabilities, **1** medium risk vulnerabilities, **3** low risk vulnerabilities, **1** informational issues.

ID	Description	Severity	Alleviation
MSA-001	Contract owner has unlimited authority to burn any user's tokens without restrictions	High risk	Fixed
MSA-002	Hardcoded 1 USD assumption for non-native token payments creates pricing vulnerability	High risk	Fixed
MSA-003	Inverted public transfer check allows transfers when disabled and blocks when enabled	Medium risk	Fixed
MSA-004	Missing chain ID and contract address in signature scheme enables cross-chain replay attacks	Low risk	Fixed
MSA-005	Use `disableInitializers` to prevent front-running on the initialize function	Low risk	Fixed
MSA-006	Centralization Risk	Low risk	Acknowledged
MSA-007	Missing emit event for key state update	Informational	Acknowledged

Findings

High risk (2)

1. Contract owner has unlimited authority to burn any user's tokens without restrictions

High risk

Security Analyzer

The `burn()` function exhibits dangerous centralization risks by allowing the contract owner to forcibly burn tokens from any address without:

1. User consent or authorization
2. Any operational constraints
3. Governance oversight
4. Emergency safeguards

Potential Impacts:

- Complete loss of user funds through arbitrary burning
- Loss of trust in the protocol's token economics
- Possible regulatory compliance issues
- Governance attacks if tokens represent voting power
- Negative market perception leading to devaluation

```
function burn(address from, uint256 id, uint256 amount) external onlyOwner {
    _burn(from, id, amount);  // Owner can burn from ANY address
    totalSupply[id] -= amount;
}
```

File(s) Affected

DgridNode.sol #41-44

```
41      function burn(address from, uint256 id, uint256 amount) external onlyOwner {
42          _burn(from, id, amount);
43          totalSupply[id] -= amount;
44      }
```

Alleviation Fixed

The team fixed this issue by remove the burn function, in the commit 7e64a6dca23b1a02b8799353fd8eb89d68b8dea5.

2. Hardcoded 1 USD assumption for non-native token payments creates pricing vulnerability

High risk

Security Analyzer

The contract contains a critical pricing vulnerability in the non-native token payment path:

1. **Issue Details**:
   - When `asset != address(0)`, the contract calculates:

     ```
     uint256 paymentAmountInAsset = (paymentAmount * 10 ** assetInfo.decimals) / 1e18;
     ```

   - This effectively assumes 1 unit of the token = 1 USD
   - There's no actual price verification for the ERC20 token

2. **Impact**:

- Severe financial miscalculations if token price ≠ 1 USD
- Potential scenarios:
  - If token price = $0.10: Users pay 10x less than intended
  - If token price = $10: Users pay 10x more than intended
- Arbitrage opportunities exploiting price discrepancies
- Protocol revenue loss or user overpayment

File(s) Affected

Dgrid.sol #146-163

```
146             Asset memory assetInfo = assetInfos[asset];
147             uint256 paymentAmountInAsset = (paymentAmount *
148                 10 ** assetInfo.decimals) / 1e18;
149             uint256 allowance = ERC20(asset).allowance(
150                 msg.sender,
151                 address(this)
152             );
153             require(
154                 allowance >= paymentAmountInAsset,
155                 "Buy Node: Insufficient allowance"
156             );
157             payValue = paymentAmountInAsset;
158             if (parent != address(0)) {
159                 commissionAmount =
160                     (paymentAmountInAsset * commissionRate) /
161                     100;
162                 commission[parent][asset] += commissionAmount;
163             }
```

Recommendation

Implement Token Price Oracle

```
// Add price feed mapping
mapping(address => AggregatorV3Interface) public priceFeeds;

function setPriceFeed(address token, address priceFeed) external onlyOwner {
    priceFeeds[token] = AggregatorV3Interface(priceFeed);
}

// In buyNode():
if (asset == address(0)) {
    // Existing BNB logic
} else {
    AggregatorV3Interface priceFeed = priceFeeds[asset];
    require(address(priceFeed) != address(0), "Price feed not set");

    (,int256 price,,,) = priceFeed.latestRoundData();
    uint256 paymentAmountInAsset = (paymentAmount * 10 ** assetInfo.decimals) / uint256(price);
    // Rest of payment logic
}
```

Alleviation Fixed

The team fixed this issue, in the commit 7e64a6dca23b1a02b8799353fd8eb89d68b8dea5.

Medium risk (1)

1. Inverted public transfer check allows transfers when disabled and blocks when enabled

Medium risk

Security Analyzer

The `_update` function incorrectly checks if `publicTransferEnabled` is true to revert transfers between users. This inverts the intended logic, allowing transfers only when `publicTransferEnabled` is false and blocking them when enabled. This breaks the contract's transfer functionality, making public transfers impossible when intended and allowing them when disabled.

File(s) Affected

DgridNode.sol #47-61

```
    function _update(
        address from,
        address to,
        uint256[] memory ids,
        uint256[] memory values
    ) internal override {
        if (from != address(0) && to != address(0) && publicTransferEnabled) {
            revert("Only owner can transfer");
        }
        super._update(from, to, ids, values);
    }

    function setPublicTransferEnabled(bool enabled) external onlyOwner {
        publicTransferEnabled = enabled;
    }
```

Recommendation

Invert the condition to check `!publicTransferEnabled` instead. Change the line to `if (from != address(0) && to != address(0) && !publicTransferEnabled)` to correctly enforce the public transfer flag.

Alleviation Fixed

The team fixed this finding, in the commit 7e64a6dca23b1a02b8799353fd8eb89d68b8dea5.

Low risk (3)

1. Missing chain ID and contract address in signature scheme enables cross-chain replay attacks

Low risk

Security Analyzer

The signature verification in `buyNode()` is vulnerable to cross-contract and cross-chain replay attacks because:

1. **Missing Chain ID**:
   - The signed message doesn't include the chain ID, allowing signatures to be replayed on different EVM chains
   - Example: A signature valid on Ethereum Mainnet could be reused on BSC or Polygon
2. **Missing Contract Address**:
   - The signed message doesn't include the contract address, enabling:
     - Replay attacks if the contract is redeployed
     - Reuse of signatures across multiple instances of the same contract
3. **Impact**:
   - Unauthorized node purchases using copied signatures

- Financial losses from duplicated orders

```
bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash(
    abi.encode(orderId, user, parent, nodeCount, expireTime) // Vulnerable encoding
);
```

File(s) Affected

Dgrid.sol #109-111

```
109         bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash(
110             abi.encode(orderId, user, parent, nodeCount, expireTime)
111         );
```

Recommendation

1. **Include Chain ID and Contract Address**: `solidity bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash( abi.encode( orderId, user, parent, nodeCount, expireTime, block.chainid, // Current chain ID address(this) // Current contract address ) );`
2. **Additional Protection Measures**: ```solidity // Mark orders as fulfilled to prevent replay on same chain mapping(uint256 ⇒ bool) public fulfilledOrders;

function buyNode(...) public payable nonReentrant { require(!fulfilledOrders[orderId], "Order already fulfilled"); // ... signature verification ... fulfilledOrders[orderId] = true; } ```

Alleviation Fixed

The team fixed this finding, in the commit 7e64a6dca23b1a02b8799353fd8eb89d68b8dea5.

2. Use `disableInitializers` to prevent front-running on the initialize function

Low risk   Security Analyzer

The contract `Dgrid`, `DgridNode` are an upgradeable implementation contracts:

```
contract DgridNode is ERC1155Upgradeable, OwnableUpgradeable {
```

```
contract Dgrid is
    Initializable,
    ReentrancyGuardUpgradeable,
    OwnableUpgradeable
```

The implementation contract behind a proxy can be initialized by any address. This is not a security problem in the sense that it impacts the system directly, as the attacker will not be able to cause any contract to self-destruct or modify any value in the proxy contract.

But, taking ownership of the implementation contract can open other attack vectors, like social engineer or phishing attack.

Reference: https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable-_disableInitializers--

File(s) Affected

Dgrid.sol #12-15

```
12  contract Dgrid is
13      Initializable,
14      ReentrancyGuardUpgradeable,
15      OwnableUpgradeable
```

DgridNode.sol #8-8

```
contract DgridNode is ERC1155Upgradeable, OwnableUpgradeable {
```

Recommendation

Consider invoking the `disableInitializers()` function to the constructor of the implementation contract: `solidity constructor() { _disableInitializers(); }`

Alleviation Fixed

The team fixed this finding, in the commit 7e64a6dca23b1a02b8799353fd8eb89d68b8dea5.

3. Centralization Risk

Low risk Security Analyzer

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

- `mint`: Allows creating new tokens (restricted to onlyDgrid role);
- `burn`: Allows destroying tokens from any address (restricted to onlyOwner);
- `setPublicTransferEnabled`: Allows enabling/disabling token transfers between users (restricted to onlyOwner).

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

- `setCommissionRate`: Allows changing the commission rate applied to transactions;
- `setServer`: Allows changing the server address (critical infrastructure control);
- `setDev`: Allows changing the developer address (privileged access control);
- `setPriceFeed`: Allows changing the Chainlink price feed oracle (critical financial data control);
- `setPriceSteps`: Allows modifying the pricing structure (business logic control).

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

- `setPriceFeed`: Allows update price feed.

File(s) Affected

DgridNode.sol #8-8

```
contract DgridNode is ERC1155Upgradeable, OwnableUpgradeable {
```

Dgrid.sol #12-15

```
contract Dgrid is
    Initializable,
    ReentrancyGuardUpgradeable,
    OwnableUpgradeable
```

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.

Alleviation Acknowledged

The team acknowledged this finding.

Informational (1)

1. Missing emit event for key state update

Informational Security Analyzer

Key state update should emit corresponding event to help the off-chain systems to track state update.

File(s) Affected

DgridNode.sol #59-61

```
59      function setPublicTransferEnabled(bool enabled) external onlyOwner {
60          publicTransferEnabled = enabled;
61      }
```

Dgrid.sol #273-296

```
273     function setCommissionRate(uint256 _commissionRate) public onlyOwner {
274         commissionRate = _commissionRate;
275     }
276
277     function setServer(address _server) public onlyOwner {
278         server = _server;
279     }
280
281     function setDev(address _dev) public onlyOwner {
282         dev = _dev;
283     }
284
285     function setPriceFeed(address _priceFeed) public onlyOwner {
286         priceFeed = ChainlinkPriceFeed(_priceFeed);
287     }
288
289     function setPriceSteps(
290         uint256[] memory _ranges,
291         uint256[] memory _prices
292     ) public onlyOwner {
293         require(_ranges.length == _prices.length, "Length mismatch");
294         stepRanges = _ranges;
295         priceSteps = _prices;
296     }
```

Alleviation Acknowledged

The team acknowledged this finding.

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.