Confidential

DGLD Token Smart Contracts V2 Mainnet

Security Audit V2

Confidential
November 2025

info@jitadigital.com

Jita Ltd

Executive Summary

This blockchain security audit covers DGLD repository https://github.com/goldtokensa/gtsa-contracts , November 2025.

Code revision under audit: `a470014db4fd582182baab0769561f3d10489dd7`
Updated code revision: `38ec170e2242ef10c40584e8195acb5ac0578d5b`

Issues Found

- 0 critical issue ;
- 0 high severity issue ;
- 1 low severity issue ;
- 3 comments.

Key Audit Results

- 1 low severity issue resolved,
- 1 comments resolved

Issues Severity	Found	Resolved	Remaining
Critical	0	0	0
High	0	0	0
Low	1	1	0
Comments	3	1	2

DGLD Token on Ethereum	Address
Proxy address	`0xA9299C296d7830A99414d1E5546F5171fA01E9c8`
Logic V1 address	`0x9a42e29369628a9D979D1A4867D7A2ce9361A6D2`
Logic V2 address	`0xA9299C296d7830A99414d1E5546F5171fA01E9c8`

Audit result: Passed

1 Methodology **4**
1.1 Audit Result 4
1.2 Issues Severity Evaluation 4
2 Scope of Work **6**
2.1 DGLD Mainnet Token V2 6
2.1.1 Github repositories 6
2.2 Assumptions and limitations 6
2.2.1 Time and version reliance 6
2.3 Disclaimer 6
2.4 Copyright 6
3 Audit Findings **8**
3.1 /src 8
3.1.1 Comment: pause and unpause could be external 8
3.1.2 Low: pause and unpause could be virtual 8
3.1.3 Comment: disable optimizer when compiling V1 9
3.1.4 Comment: DGLDTokenV1 on-chain bytecode verification fails 10

1 Methodology

1.1 Audit Result

Jita's current methodology does not score audits, as such scores can be subjective, easy to manipulate, create a false sense of security and mislead people into comparing audits and security scores.

Instead, the final audit is only assigned a value PASS or FAIL.

Audit Result	
PASS	The issues found by the auditor(s) have all been resolved or explained by the development team.
FAIL	Some of the issues found by the auditor(s) have not all been resolved or explained by the development team.

1.2 Issues Severity Evaluation

The following section presents the audit findings.

Each issue is located as precisely as possible in the codebase, a description gives more information on the problem, and one or more recommendations are provided.

The client answer to each issue is included.

Each issue risk severity is estimated following the OWASP risk rating methodology and is assigned an overall risk severity which can take one of four values:

Issue Overall Risk Severity	
Critical	The issue compromises the project and must be fixed.
High	The issue is either very likely with low impact, or unlikely but with a big impact on the business. The issue must be fixed.
Low	The issue does not compromise the project, or it has a potentially minor impact on the system operation.
Comment	The issue does not compromise the project.

The issue overall risk severity is evaluated based on two metrics:
1) the likelihood of the issue being exploited (technical feasibility of exploit) and
2) the technical/business/financial impact of the issue.

An issue's overall risk severity is an estimate and may be revised higher or lower by the client company based on their own estimates of business impact.

2 Scope of Work

2.1 DGLD Mainnet Token V2

2.1.1 Github repositories

- git@github.com:goldtokensa/gtsa-smart-contracts.git
  Mainnet token for DGLD - V2
  Git revision a470014db4fd582182baab0769561f3d10489dd7

2.2 Assumptions and limitations

2.2.1 Time and version reliance

The security audits are conducted within a specified time frame and are reliant on the specific version of the target code, system and information provided by the client, its affiliates, or its partners. Therefore, the list of identified vulnerabilities and security issues identified during the audit should not be considered comprehensive and exhaustive. The audit results do not imply any guarantee that all potential vulnerabilities, flaws, or defects have been detected.

2.3 Disclaimer

Jita does not warrant that the Deliverables will identify all vulnerabilities.The Deliverables shall not be considered as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

Jita disclaims all other warranties express, implied, statutory, or otherwise, including, without limitation, the implied warranties of merchantability, non-infringement, and fitness for a particular purpose.

This audit is not financial advice.

2.4 Copyright

This report accessed through any source other than those controlled by Jita Digital or the Customer should not be considered authentic.

3 Audit Findings

3.1 /src

3.1.1 Comment: pause and unpause could be external

3.1.1.1 Severity

Comment

3.1.1.2 Files

`src/DGLDTokenV2.sol`

3.1.1.3 Functions

```
function pause()
function unpause()
```

3.1.1.4 Description

The functions pause and unpause are meant to be called directly by an administrator key with PAUSER_ROLE privileges.
Thus the function type could be external rather than public.
This prevents internal smart contracts functions from calling the function, for instance in a future update.

3.1.1.5 Recommendation

Consider using keyword external instead of public:

```
function pause() external onlyRole(PAUSER_ROLE) {
```

3.1.1.6 Customer

Implemented in `8b9a128b04dc2ceb510a8936b7534b65d67b0f81`

3.1.2 Low: pause and unpause could be virtual

3.1.2.1 Severity

Low

3.1.2.2 Files

`src/DGLDTokenV2.sol`

3.1.2.3 Functions

```
function pause() public onlyRole(PAUSER_ROLE) {
```

```
function unpause() public onlyRole(PAUSER_ROLE) {
```

3.1.2.4 Description

The functions pause and unpause are not marked virtual.
This prevents them from being overridden in a future upgrade if needed.

3.1.2.5 Recommendation

Consider adding the keyword virtual to the functions declaration so that a hypothetical V3 of the smart contracts could redefine these functions if needed:

```
function pause() external virtual onlyRole(PAUSER_ROLE) {
```

3.1.2.6 Customer

Implemented in `8b9a128b04dc2ceb510a8936b7534b65d67b0f81`
Also applied to Blacklistable.sol in `3e42843d02240be6e33af58cf8d02564205df7c2`

3.1.3 Comment: disable optimizer when compiling V1

3.1.3.1 Severity

Comment

3.1.3.2 Files

```
foundry.toml
truffle-config.js
```

3.1.3.3 Functions

```
N.A.
```

3.1.3.4 Description

The first version of DGLDToken.sol was compiled with Truffle.
When reviewing the configuration we find:
optimizer { enabled: false, runs: 200}
With the new version, the toolchain has been updated to foundry.
In its configuration file, we find
optimizer = true
Thus, we are working with a slightly different compiler configuration that could lead to using a different bytecode version of V1 in testing operations (locally, on sepolia and on mainnet) than the one that has been deployed on-chain and that will be used for actual smart contracts upgrade.

3.1.3.5 Recommendation

Consider using different compilation profiles to compile the old and the new smart contracts.

It might also be easier to place the old and new source files in separate directories.

This is a comment and will not impact the security of the system.

For instance:

```
# foundry.toml
[profile.v1]
src= “contracts/v1”
optimizer = false
optimizer_runs = 200

[profile.v2]
src=”contracts/v2”
optimizer = true
optimizer_runs = 200

#compilation
forge build -profile v1
forge build -profile v2
```

3.1.3.6 Customer

Comment not implemented as it does not impact the system security.

3.1.4 Comment: DGLDTokenV1 on-chain bytecode verification fails

3.1.4.1 Severity

Comment

3.1.4.2 Files

DGLDToken.sol

3.1.4.3 Functions

N.A.

3.1.4.4 Description

Bytecode verification fails between DGLDTokenV1 on-chain version and locally compiled version with new toolchain.
Version 1 of DGLDToken used Truffle V5 for compilation and deployment.
Version 2 uses Foundry / Forge, a more modern toolchain.
Some compilation options differ (see previous issue). Even after updating it, the bytecode verification fails. It seems that Version 1 as compiled with Foundry differs from Version 1 as deployed on mainnet. All test scripts use the Foundry compiled version of the smart contract. However, we did run the storage differences script with and without optimisation and found no memory layout differences.
It is unlikely that the bytecode version deployed on-chain differs significantly from the bytecode version used for testing.

Verification details:

1) Attempting bytecode verification fails (Proxy address):

```
forge verify-contract  --skip-is-verified-check --guess-constructor-args
--watch --chain 1 0xA9299C296d7830A99414d1E5546F5171fA01E9c8 DGLDToken
Start verifying contract `0xA9299C296d7830A99414d1E5546F5171fA01E9c8` deployed
on mainnet
Error: Local bytecode doesn't match on-chain bytecode
```

2) Attempting bytecode verification fails (logic implementation):

```
# Verification of DGLDToken implementation logic fails
forge verify-contract  --skip-is-verified-check --guess-constructor-args
--watch --chain 1 0x9a42e29369628a9D979D1A4867D7A2ce9361A6D2 DGLDToken
Start verifying contract `0x9a42e29369628a9D979D1A4867D7A2ce9361A6D2` deployed
on mainnet
Error: Local bytecode doesn't match on-chain bytecode
```

3.1.4.5 Recommendation

If possible, identify the reason why bytecode verification fails between the foundry generated bytecode for DGLDToken V1 and the on-chain version compiled with Truffle.
This does not impact system security as (1) the etherscan verification passed and the code matches the github codebase history and (2) upon inspection, the memory layout remains the same.

3.1.4.6 Customer

As the test suite forks Ethereum mainnet, we can verify that the upgrade works as planned even without being able to re-verify the V1 bytecode.