CD SECURITY

AUDIT REPORT

Donut DAO
January 2025

Prepared by
tsvetanovv
Pelz

Introduction

A time-boxed security review of the **Donut DAO** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **Donut DAO**

Donut DAO is a decentralized organization focused on expanding the Donut ecosystem. Its mission is to grow DONUT's use cases, promote decentralization through SocialFi, and bridge Web3 with real-world impact.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - d17c74aec740537fb1af35d6f2de7fa400850af7**

Scope

The following folders were in scope of the audit:

- `Token.sol`
- `TokenManager.sol`

The following number of issues were found, categorized by their severity:

- Critical & High: 1 issues
- Medium: 0 issues
- Low & Info: 10 issues

Findings Summary

ID	Title	Severity	Status
[H-01]	Missing `transferable` Check in `send` Function Breaks Core Invariant	High	Acknowledged
[L-01]	`proxyPayment` Function Can Lock Ether Due to Missing Access Control	Low	Acknowledged
[L-02]	Irreversible Disabling of `allowChangeDonutController` Limits Future Flexibility	Low	Acknowledged
[L-03]	Lack of Validation on Controller Changes	Low	Fixed
[L-04]	Centralized Control Risk in `TokenManager` and `Token` Contracts	Low	Fixed
[I-01]	Redundant `pragma solidity` Declaration	Informational	Acknowledged
[I-02]	Remove Redundant Comments for Code Neatness	Informational	Acknowledged
[I-03]	Consider Emitting an Event for `enableTransfers` State Changes	Informational	Acknowledged
[I-04]	Consider using `try/catch` for ERC777 Transfer Handling	Informational	Acknowledged
[I-05]	Non-Fixed Pragma Version	Informational	Fixed
[I-06]	Lack of Event Emissions	Informational	Fixed

Detailed Findings

[H-01] Missing `transferable` Check in `send` Function Breaks Core Invariant

Severity

Impact: Medium
Likelihood: High

Description

The `Token.sol` contract is designed to prevent token transfers unless `transfersEnabled` is `true` or `msg.sender` is the `controller`. This restriction is enforced through the `transferable` modifier:

```
modifier transferable() {
    require(msg.sender == controller || transfersEnabled,
"NON_TRANSFERABLE");
    _;
}
```

However, the `send` function does not apply this modifier, allowing token transfers even when `transfersEnabled` is `false`:

```
function send(address to, uint256 value, bytes data) external {
    _transfer(msg.sender, to, value); // @audit-issue: No transferable
check, bypassing transfer restriction
    emit Sent(msg.sender, msg.sender, to, value, data, "");

    if (isContract(to))
        IERC777Recipient(to).tokensReceived(msg.sender, msg.sender, to,
value, data, "");
}
```

This oversight allows **anyone** to transfer tokens even when transfers are explicitly disabled, breaking the intended invariant of the contract.

Recommendations

Add the `transferable` modifier to the `send` function to ensure transfer restrictions are enforced:

```
function send(address to, uint256 value, bytes data) external transferable
{
    _transfer(msg.sender, to, value);
    emit Sent(msg.sender, msg.sender, to, value, data, "");

    if (isContract(to))
        IERC777Recipient(to).tokensReceived(msg.sender, msg.sender, to,
value, data, "");
}
```

This change ensures that only the `controller` or users sending tokens when `transfersEnabled` is `true` can execute transfers, maintaining the intended access control.

[L-01] `proxyPayment` Function Can Lock Ether Due to Missing Access Control

Description

The `TokenController` contract defines a payable function `proxyPayment` as follows:

```
function proxyPayment(address) external payable returns (bool) {
    return false;
}
```

Issues:

- The function is **publicly callable**, meaning **anyone** can send Ether to it.
- There is no check on `msg.value`, allowing unintended Ether deposits.
- Since the function **always returns false** and there is **no withdrawal mechanism**, any Ether sent will be **permanently locked** in the contract.

Recommendations

- If `proxyPayment` should only be called by specific addresses, enforce an access control mechanism:

```
function proxyPayment(address) external payable onlyOwner returns (bool) {
    return false;
}
```

- If this function **should not** receive Ether at all, make it non-payable:

```
function proxyPayment(address) external returns (bool) {
    return false;
}
```

- If receiving Ether is necessary, allow withdrawal:

```
function withdrawEther(address payable recipient) external onlyOwner {
    recipient.transfer(address(this).balance);
}
```

[L-02] Irreversible Disabling of `allowChangeDonutController` Limits Future

Flexibility

Description

The `TokenManager` contract defines a variable `allowChangeDonutController`, which determines whether the `DONUT` controller address can be changed:

```
bool public allowChangeDonutController = true;
```

This variable is used in the `changeDonutController` function:

```
function changeDonutController(address newController) public multisig {
    require(allowChangeDonutController, "NOT_ALLOWED");
    DONUT.changeController(newController);
}
```

To disable further changes, the contract provides the following function:

```
function disableChangeDonutController() public multisig {
    allowChangeDonutController = false;
}
```

Issue: One-Way Reset Without Re-Enable Functionality

- Once `disableChangeDonutController()` is called, **there is no way to set `allowChangeDonutController` back to `true`**.
- This creates an **irreversible governance action**, which may be problematic if there is a need to update the `DONUT` controller in the future.

Recommendations

If the intention is to **permanently lock** the controller change, this behavior is acceptable. However, if flexibility is desired, consider introducing a **re-enable function**:

```
function enableChangeDonutController() public multisig {
    allowChangeDonutController = true;
}
```

[L-03] Lack of Validation on Controller Changes

Severity

Impact: Low **Likelihood:** Low

Description

The `changeDonutController()` function allows the `MULTISIG` to update the controller of the DONUT token contract.

```
    function changeDonutController(address newController) public multisig
{
        require(allowChangeDonutController, "NOT_ALLOWED");
        DONUT.changeController(newController);
    }
```

However, there is no validation to ensure that the `newController` address is valid or zero.

Recommendations

Ensure that `newController` is not a zero address.

[L-04] Centralized Control Risk in `TokenManager` and `Token` Contracts

Severity

Impact: High
Likelihood: Low

Description

Both `TokenManager.sol` and `Token.sol` implement centralized control mechanisms, where a single privileged entity (a multisig or controller) has absolute authority over critical functions such as minting, burning, and transfer permissions.

Centralization Risks in `TokenManager.sol`

The `MULTISIG` address has exclusive control over:

- Minting new tokens via `mintBatch()`.
- Changing the DONUT token controller via `changeDonutController()`.
- Permanently disabling controller changes via `disableChangeDonutController()`.

Centralization Risks in `Token.sol`

The contract inherits from `Controlled`, meaning only the controller can:

- Mint new tokens (`generateTokens()`).

- Burn tokens (`destroyTokens()`).
- Enable or disable transfers (`enableTransfers()`).

Recommendations

You can add Timelock for these functions.

[I-01] Redundant `pragma solidity` Declaration

The `Token` contract defines the `pragma solidity` directive **twice** in the same file:

```
pragma solidity ^0.4.24;

/* import "./Controlled.sol"; */
/* import "./ITokenController.sol"; */
import "@aragon/apps-shared-minime/contracts/MiniMeToken.sol";
import "@aragon/apps-shared-minime/contracts/ITokenController.sol";
import "@aragon/os/contracts/lib/math/SafeMath.sol";
import "./IERC777Recipient.sol";

pragma solidity ^0.4.24; // @audit-issue pragma defined twice
```

Issue:

- Solidity **only considers the first valid `pragma` directive**, making the second declaration **redundant**.

Recommendations

- **Remove the duplicate `pragma solidity ^0.4.24;` declaration** to maintain **clean and readable code**.

[I-02] Remove Redundant Comments for Code Neatness

The `transferFrom` function contains a commented-out block of code that is **redundant and unnecessary**:

```
function transferFrom(address from, address to, uint256 value) public
transferable returns (bool) {
    _transfer(from, to, value);
    /* if(msg.sender != controller) {   // @audit-info remove unecessary
comments
      _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
    } */
    _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
```

```
        return true;
    }
```

Recommendations

- **Remove the redundant commented-out code** if it is not required.
- If the commented logic **might be useful later**, consider adding **context** explaining why it was removed or how it might be restored.

[I-03] Consider Emitting an Event for `enableTransfers` State Changes

The `enableTransfers` function modifies the `transfersEnabled` state variable but **does not emit an event** to notify external observers:

```
function enableTransfers(bool _transfersEnabled) public onlyController {
// @audit-info consider emitting events for state changes.
    transfersEnabled = _transfersEnabled;
}
```

Recommendations

- **Emit an event** whenever `transfersEnabled` is modified. Example:

```
event TransfersEnabled(bool enabled);

function enableTransfers(bool _transfersEnabled) public onlyController {
    transfersEnabled = _transfersEnabled;
    emit TransfersEnabled(_transfersEnabled);
}
```

[I-04]Consider using `try/catch` for ERC777 Transfer Handling

The `send` function attempts to call the `tokensReceived` function on the recipient address if the recipient is a contract. However, it does so without any failure handling:

```
function send(address to, uint256 value, bytes data) external {
    _transfer(msg.sender, to, value);
    emit Sent(msg.sender, msg.sender, to, value, data, "");
    if (isContract(to))
```

```
        IERC777Recipient(to).tokensReceived(msg.sender, msg.sender, to,
value, data, "");
}
```

Issue:

- If the recipient contract's `tokensReceived` function fails (e.g., due to a revert), the entire transaction will fail, wasting gas for the sender.
- This creates a **denial of service** risk for any contract that does not properly handle `tokensReceived`.

Recommendations

- Use `try/catch` to handle the potential failure of `tokensReceived` .

Example:

```
function send(address to, uint256 value, bytes data) external {
    _transfer(msg.sender, to, value);
    emit Sent(msg.sender, msg.sender, to, value, data, "");

    if (isContract(to)) {
        try IERC777Recipient(to).tokensReceived(msg.sender, msg.sender,
to, value, data, "") {
            // Success - no action needed
        } catch {
            // Handle failure silently to prevent reverts
        }
    }
}
```

This ensures that even if `tokensReceived` fails, the transaction will not revert.

[I-05] Non-Fixed Pragma Version

The contracts specify a **floating pragma version** (`^0.8.27`), allowing the use of **any compiler version** starting from **0.8.27** up to the last version.

While this provides flexibility, it can introduce **unintended behavior** if a newer compiler version introduces changes or deprecations.

Recommendations

Set a **specific compiler version** to ensure consistent compilation and avoid potential risks. For example, in `TokenManager.sol`:

```
pragma solidity 0.8.27;
```

[I-06] Lack of Event Emissions

The `TokenManager` contract performs several critical operations, such as **minting tokens**, **changing the controller**, and **disabling controller changes**, but it does **not emit events** for these actions.

```
    function changeDonutController(address newController) public multisig
{
        require(allowChangeDonutController, "NOT_ALLOWED");
        DONUT.changeController(newController);
    }

    function disableChangeDonutController() public multisig {
        allowChangeDonutController = false;
    }

    function mintBatch() public multisig {
        require(block.timestamp > lastBatch + BATCH_INTERVAL, "TOO_SOON");
        lastBatch = block.timestamp;
        DONUT.generateTokens(MULTISIG, BATCH_AMOUNT);
    }
```

Recommendations

Add event emissions for these functions.