

Europe stablecoin
Security Review

Cantina Managed review by:

MiloTruck, Lead Security Researcher
Haxatron, Security Researcher

June 6, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Medium Risk . . . 4
  - 3.1.1 Blocklist can be bypassed by renouncing the `BLOCKED_ROLE` . . . 4
  - 3.1.2 Admin requires allowance to burn tokens from an `account` through `burnFrom()` . . . 5
- 3.2 Low Risk . . . 5
  - 3.2.1 ERC20 tokens that do not return a `bool` on `transfer` cannot be rescued . . . 5
- 3.3 Informational . . . 6
  - 3.3.1 Minor code improvements . . . 6

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Perper is dedicated to simplifying the integration of its payment system into existing business platforms.

From May 29th to May 30th the Cantina team conducted a review of europe-stablecoin on commit hash aefac378. The team identified a total of **4** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	1	1	0
Gas Optimizations	0	0	0
Informational	1	1	0
Total	**4**	**4**	**0**

3 Findings

3.1 Medium Risk

3.1.1 Blocklist can be bypassed by renouncing the `BLOCKED_ROLE`

Severity: Medium Risk

Context: EUROPE.sol#L100-L111, AccessControlUpgradeable.sol#L162-L184

Description: The EUROPE contract implements a blocklist for token transfers using `AccessControlUpgradeable` and assigning the `BLOCKED_ROLE` to blocklisted users.

- EUROPE.sol#L100-L111:

```
// Override transfer function to block transfers for blocked users
function transfer(address to, uint256 amount) public override returns (bool) {
    require(!hasRole(BLOCKED_ROLE, _msgSender()), "ERC20: sender is blocked");
    require(!hasRole(BLOCKED_ROLE, to), "ERC20: recipient is blocked");
    return super.transfer(to, amount);
}

// Override transferFrom function to block transfers for blocked users
function transferFrom(address from, address to, uint256 amount) public override returns (bool) {
    require(!hasRole(BLOCKED_ROLE, from), "ERC20: sender is blocked");
    require(!hasRole(BLOCKED_ROLE, to), "ERC20: recipient is blocked");
    return super.transferFrom(from, to, amount);
}
```

The problem is that since `AccessControlUpgradeable` contains the `renounceRole` function which allows the `msg.sender` to remove their own roles, a blocklisted user can renounce their own `BLOCKED_ROLE` and thus remove themself from the blocklist.

- AccessControlUpgradeable.sol#L162-L184:

```
/**
 * @dev Revokes `role` from the calling account.
 *
 * Roles are often managed via {grantRole} and {revokeRole}: this function's
 * purpose is to provide a mechanism for accounts to lose their privileges
 * if they are compromised (such as when a trusted device is misplaced).
 *
 * If the calling account had been revoked `role`, emits a {RoleRevoked}
 * event.
 *
 * Requirements:
 *
 * - the caller must be `callerConfirmation`.
 *
 * May emit a {RoleRevoked} event.
 */
function renounceRole(bytes32 role, address callerConfirmation) public virtual {
    if (callerConfirmation != _msgSender()) {
        revert AccessControlBadConfirmation();
    }

    _revokeRole(role, callerConfirmation);
}
```

Recommendation: Prevent renouncing the `BLOCKED_ROLE`.

- EUROPE.sol#L100-L111:

```
function renounceRole(bytes32 role, address account) public override {
    require(role != BLOCKED_ROLE, "Cannot renounce BLOCKED_ROLE");
    super.renounceRole(role, account);
}
```

Perper: Fixed in commit 425c9f6.

Cantina Managed: Verified, `renounceRole()` has been overridden to always revert.

3.1.2 Admin requires allowance to burn tokens from an `account` through `burnFrom()`

Severity: Medium Risk

Context: EUROPE.sol#L64-L70, ERC20BurnableUpgradeable.sol#L41-L44

Description: The contract includes a `burnFrom()` function, presumably for an admin with `BURNER_ROLE` to burn tokens from malicious addresses:

```
function burnFrom(address account, uint256 amount)
    public
    override
    onlyRole(BURNER_ROLE)
{
    super.burnFrom(account, amount);
}
```

However, `ERC20BurnableUpgradeable.burnFrom()` (which is called via `super.burnFrom()` above) requires the caller to have allowance from `account` to burn tokens:

```
function burnFrom(address account, uint256 value) public virtual {
    _spendAllowance(account, _msgSender(), value);
    _burn(account, value);
}
```

As a result, the admin will not be able to tokens from an address, unless that address explicitly grants allowance to the admin.

Recommendation: Modify `burnFrom()` to directly call `ERC20Upgradeable._burn()`:

```
  function burnFrom(address account, uint256 amount)
      public
      override
      onlyRole(BURNER_ROLE)
  {
-     super.burnFrom(account, amount);
+     _burn(account, amount);
  }
```

Perper: Fixed in commit 425c9f6.

Cantina Managed: Verified, the recommendation was implemented.

3.2 Low Risk

3.2.1 ERC20 tokens that do not return a `bool` on `transfer` cannot be rescued

Severity: Low Risk

Context: EUROPE.sol#L113-L122, IERC20.sol#L41.

Description: Certain non-compliant ERC20 tokens such as USDT do not return a `bool` on `transfer`. The rescue function uses `IERC20.transfer` to rescue the tokens that expects a `bool` as return value.

- EUROPE.sol#L113-L122:

```
/**
 * @dev Rescues ERC20 tokens sent to the contract by mistake.
 * @param token The ERC20 token contract to rescue.
 * @param to The address to send the rescued tokens to.
 * @param amount The amount of tokens to rescue.
 */
function rescueERC20(IERC20 token, address to, uint256 amount) external onlyRole(RESCUER_ROLE) {
    require(address(token) != address(this), "Cannot rescue the same token as this contract");
    token.transfer(to, amount);
}
```

- IERC20.sol#L41:

```
function transfer(address to, uint256 value) external returns (bool);
```

Therefore, when attempting to rescue a token such as USDT, the function will revert as the `IERC20` interface will expect a `bool` to be returned but `USDT.transfer` doesn't return a `bool`.

Recommendation: In `rescueERC20`, consider using `SafeERC20.safeTransfer` to allow handling of non-compliant ERC20 tokens such as USDT.

Perper: Fixed in commit 425c9f6.

Cantina Managed: Verified, the recommendation was implemented.

3.3 Informational

3.3.1 Minor code improvements

Severity: Informational

Context: *(See each case below)*

Description/Recommendation:

1. EUROPE.sol#L15 - `Initializable` and `ERC20Upgradeable` do not needed to be inherited by `EUROPE` as the other contracts already inherit them.
2. EUROPE.sol#L107-L111 - In `transferFrom()`, consider checking that `msg.sender` does not have `BLOCKED_ROLE` as well to prevent blacklisted addresses performing transfers via allowances.
3. EUROPE.sol#L120 - Consider removing the `address(token) != address(this)` check in `rescueERC20()` to allow `EUROPE` to be rescued from the contract.
4. EUROPE.sol#L82-L88 - Consider removing the `whenNotBlocked` modifier as it is not used in the contract.

Perper: Fixed in commit 425c9f6 and 745e39f.

Cantina Managed: Verified, the recommendations were implemented.