Smart Contract Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the AWE Network (AWE) on 2025.02.18. The following are the details and results of this smart contract security audit:

Token Name :

AWE Network (AWE)

The contract address :

AWE Token

https://basescan.org/address/0x1b4617734c43f6159f3a70b7e06d883647512778 (Proxy)

https://basescan.org/address/0xf3c2540c95a81d9c06d3db779c71d65ae66b04eb (Implementation)

AWEGovernor

https://basescan.org/address/0xae18aed3dd3c9cd1d0a180315b6b5fcd61ef20f1

AWETimelockController

https://basescan.org/address/0x91876f0f9ba79a165422286fd9e4620238c42929

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability Audit	Low Risk
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed

NO.	Audit Items	Result
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Low Risk

Audit Number : 0X002502200001

Audit Date : 2025.02.18 - 2025.02.20

Audit Team : SlowMist Security Team

Summary conclusion : This upgradeable token contract contains the Governor and Timelock sections and does not contain the dark coin functions. The total amount of contract tokens remains unchangeable. The contract does not have the Overflow and the Race Conditions issue.

The project team has transferred the DEFAULT_ADMIN_ROLE and the UPGRADER_ROLE of the AWE token contract to the AWETimelockController in the 0xd2a0bf5efb0f6a32726fa81d91eb092df85e918df266c7013191b45983b57555 and 0x07d6f1f4d95b37788ad4bf84793c51574d1a8779206b65b5b60a51314546a61a transactions.

During the audit, we found the following issue:

1. The VETO_GUARDIAN can directly cancel a proposal through the cancel function in the AWEGovernor contract when the proposal is in the Pending, Active, Succeeded, or Queued state. This will lead to the risk of over-privileged. And the VETO_GUARDIAN is a 3/5 multisig contract owned by 5 EOA addresses.

The source code:

AWE.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import {ERC20Upgradeable} from "@openzeppelin/contracts-
upgradeable/token/ERC20/ERC20Upgradeable.sol";
```

```
import {
    ERC20PermitUpgradeable,
    NoncesUpgradeable
} from "@openzeppelin/contracts-
upgradeable/token/ERC20/extensions/ERC20PermitUpgradeable.sol";

import {AccessControlUpgradeable} from "@openzeppelin/contracts-
upgradeable/access/AccessControlUpgradeable.sol";

import {ERC20VotesUpgradeable} from
    "@openzeppelin/contracts-
upgradeable/token/ERC20/extensions/ERC20VotesUpgradeable.sol";

import {Initializable} from "@openzeppelin/contracts-
upgradeable/proxy/utils/Initializable.sol";
import {UUPSUpgradeable} from "@openzeppelin/contracts-
upgradeable/proxy/utils/UUPSUpgradeable.sol";
import {VotesUpgradeable} from "@openzeppelin/contracts-
upgradeable/governance/utils/VotesUpgradeable.sol";

//SlowMist// The contract does not have the Overflow and the Race Conditions issue
/// @title AWE
/// @author AWE Network
contract AWE is
    Initializable,
    ERC20Upgradeable,
    ERC20PermitUpgradeable,
    ERC20VotesUpgradeable,
    AccessControlUpgradeable,
    UUPSUpgradeable
{
    bytes32 public constant UPGRADER_ROLE = keccak256("UPGRADER_ROLE");

    /// @custom:oz-upgrades-unsafe-allow constructor
    constructor() {
        _disableInitializers();
    }

    /// @notice Initializes the token and inherited contracts.
    function initialize() external initializer {
        __ERC20_init("AWE Network", "AWE");
        __ERC20Permit_init("AWE Network");
        __ERC20Votes_init();
        __AccessControl_init();
        __UUPSUpgradeable_init();

        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _grantRole(UPGRADER_ROLE, msg.sender);
```

```
        _mint(msg.sender, 2_000_000_000 * 10 ** decimals());
    }

    /// @inheritdoc VotesUpgradeable
    function clock() public view override returns (uint48) {
        return uint48(block.timestamp);
    }

    /// @inheritdoc VotesUpgradeable
    function CLOCK_MODE() public pure override returns (string memory) {
        return "mode=timestamp";
    }

    /// @inheritdoc ERC20PermitUpgradeable
    function nonces(address owner) public view override(ERC20PermitUpgradeable,
NoncesUpgradeable) returns (uint256) {
        return super.nonces(owner);
    }

    /// @inheritdoc ERC20Upgradeable
    function _update(address from, address to, uint256 value)
        internal
        override(ERC20Upgradeable, ERC20VotesUpgradeable)
    {
        super._update(from, to, value);
    }

    /// @inheritdoc UUPSUpgradeable
    function _authorizeUpgrade(address newImplementation) internal override
onlyRole(UPGRADER_ROLE) {}
}
```

AWETimelockController.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/governance/TimelockController.sol";
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
contract AWETimelockController is TimelockController {
    string private _name;

    constructor(
        string memory name_,
        uint256 minDelay,
        address[] memory proposers,
```

```
        address[] memory executors,
        address admin
    ) TimelockController(minDelay, proposers, executors, admin) {
        _name = name_;
    }

    function name() public view returns (string memory) {
        return _name;
    }
}
```

AWEGovernor.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import {Governor} from "@openzeppelin/contracts/governance/Governor.sol";
import {GovernorVotes} from
"@openzeppelin/contracts/governance/extensions/GovernorVotes.sol";
import {GovernorTimelockControl} from
  "@openzeppelin/contracts/governance/extensions/GovernorTimelockControl.sol";
import {GovernorSettings} from
"@openzeppelin/contracts/governance/extensions/GovernorSettings.sol";
import {GovernorVotesQuorumFraction} from
  "@openzeppelin/contracts/governance/extensions/GovernorVotesQuorumFraction.sol";
import {IVotes} from "@openzeppelin/contracts/governance/utils/IVotes.sol";
import {TimelockController} from
"@openzeppelin/contracts/governance/TimelockController.sol";
import {GovernorCountingFractional} from
"@openzeppelin/contracts/governance/extensions/GovernorCountingFractional.sol";

//SlowMist// The contract does not have the Overflow and the Race Conditions issue
/// @title AWEGovernor
/// @author AWE Network
/// @notice A governance contract to govern AWE protocol decision making.
/// @custom:security-contact
contract AWEGovernor is
  GovernorCountingFractional,
  GovernorVotes,
  GovernorTimelockControl,
  GovernorSettings,
  GovernorVotesQuorumFraction
{
  /// @notice Human readable name of this Governor.
  string private constant GOVERNOR_NAME = "AWE Network Governor";

  /// @notice An immutable address that can veto proposals.
```

```
  address public immutable VETO_GUARDIAN;

  /// @param _token The token used for voting on proposals.
  /// @param _initialQuorumNumerator The initial percentage of total votes needed to
pass a
  /// proposal.
  /// @param _initialVotingDelay The delay before voting on a proposal begins.
  /// @param _initialVotingPeriod The period of time voting will take place.
  /// @param _initialProposalThreshold The number of tokens needed to create a
proposal.
  /// @param _timelock The timelock used for managing proposals.
  /// @param _vetoGuardian The initial address that can cancel a proposal at different
points in the
  /// proposal lifecycle.
  constructor(
    IVotes _token,
    uint256 _initialQuorumNumerator,
    uint48 _initialVotingDelay,
    uint32 _initialVotingPeriod,
    uint256 _initialProposalThreshold,
    TimelockController _timelock,
    address _vetoGuardian
  )
    GovernorVotes(_token)
    GovernorVotesQuorumFraction(_initialQuorumNumerator)
    GovernorSettings(_initialVotingDelay, _initialVotingPeriod,
_initialProposalThreshold)
    GovernorTimelockControl(_timelock)
    Governor(GOVERNOR_NAME)
  {
    VETO_GUARDIAN = _vetoGuardian;
  }

  /// @inheritdoc Governor
  /// @dev We allow an immutable address to have the ability to cancel a proposal at
any stage
  /// of the proposal lifecycle except in the situation it was defeated or executed.
  //SlowMist// The VETO_GUARDIAN can directly cancel a proposal through the cancel
function when the proposal is in the Pending, Active, Succeeded, or Queued state. This
will lead to the risk of over-privileged
  function cancel(
    address[] memory _targets,
    uint256[] memory _values,
    bytes[] memory _calldatas,
    bytes32 _descriptionHash
  ) public virtual override returns (uint256) {
    // The proposalId will be recomputed in the `_cancel` call further down. However
we need the
    // value before we do the internal call, because we need to check the proposal
```

```
state BEFORE the
    // internal `_cancel` call changes it. The `hashProposal` duplication has a cost
that is
    // limited, and that we accept.
    uint256 proposalId = hashProposal(_targets, _values, _calldatas,
_descriptionHash);
    if (_msgSender() == VETO_GUARDIAN) {
      _validateStatus(
        proposalId,
        _encodeStateBitmap(ProposalState.Pending) |
_encodeStateBitmap(ProposalState.Active)
          | _encodeStateBitmap(ProposalState.Succeeded) |
_encodeStateBitmap(ProposalState.Queued)
      );
      return _cancel(_targets, _values, _calldatas, _descriptionHash);
    }

    _validateStatus(proposalId, _encodeStateBitmap(ProposalState.Pending));
    if (_msgSender() != proposalProposer(proposalId)) revert
GovernorOnlyProposer(_msgSender());
    return _cancel(_targets, _values, _calldatas, _descriptionHash);
  }

  /// @inheritdoc GovernorSettings
  /// @dev We override this function to resolve ambiguity between inherited contracts.
  function proposalThreshold()
    public
    view
    virtual
    override(Governor, GovernorSettings)
    returns (uint256)
  {
    return GovernorSettings.proposalThreshold();
  }

  /// @inheritdoc GovernorTimelockControl
  /// @dev We override this function to resolve ambiguity between inherited contracts.
  function state(uint256 proposalId)
    public
    view
    virtual
    override(Governor, GovernorTimelockControl)
    returns (ProposalState)
  {
    return GovernorTimelockControl.state(proposalId);
  }

  /// @inheritdoc GovernorTimelockControl
  /// @dev We override this function to resolve ambiguity between inherited contracts.
```

```
  function _executor()
    internal
    view
    virtual
    override(Governor, GovernorTimelockControl)
    returns (address)
  {
    return GovernorTimelockControl._executor();
  }

  /// @inheritdoc GovernorTimelockControl
  /// @dev We override this function to resolve ambiguity between inherited contracts.
  function _cancel(
    address[] memory _targets,
    uint256[] memory _values,
    bytes[] memory _calldatas,
    bytes32 _descriptionHash
  ) internal virtual override(Governor, GovernorTimelockControl) returns (uint256) {
    return GovernorTimelockControl._cancel(_targets, _values, _calldatas,
_descriptionHash);
  }

  /// @inheritdoc GovernorTimelockControl
  /// @dev We override this function to resolve ambiguity between inherited contracts.
  function _executeOperations(
    uint256 _proposalId,
    address[] memory _targets,
    uint256[] memory _values,
    bytes[] memory _calldatas,
    bytes32 _descriptionHash
  ) internal virtual override(Governor, GovernorTimelockControl) {
    return GovernorTimelockControl._executeOperations(
      _proposalId, _targets, _values, _calldatas, _descriptionHash
    );
  }

  /// @inheritdoc GovernorTimelockControl
  /// @dev We override this function to resolve ambiguity between inherited contracts.
  function _queueOperations(
    uint256 _proposalId,
    address[] memory _targets,
    uint256[] memory _values,
    bytes[] memory _calldatas,
    bytes32 _descriptionHash
  ) internal virtual override(Governor, GovernorTimelockControl) returns (uint48) {
    return GovernorTimelockControl._queueOperations(
      _proposalId, _targets, _values, _calldatas, _descriptionHash
    );
  }
```

```
  /// @inheritdoc GovernorTimelockControl
  /// @dev We override this function to resolve ambiguity between inherited contracts.
  function proposalNeedsQueuing(uint256 _proposalId)
    public
    view
    virtual
    override(Governor, GovernorTimelockControl)
    returns (bool)
  {
    return GovernorTimelockControl.proposalNeedsQueuing(_proposalId);
  }

  /// @notice A re-implementation of `_validateStateBitmap` on the Governor contract as that
  /// function is private. We used the code from
  /// [this](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/cae60c595b37b1e7ed7dd50ad0257387ec07c0cf/contracts/governance/Governor.sol#L734)
  /// line.
  function _validateStatus(uint256 _proposalId, bytes32 _allowedStates)
    internal
    view
    returns (ProposalState)
  {
    ProposalState currentState = state(_proposalId);
    if (_encodeStateBitmap(currentState) & _allowedStates == bytes32(0)) {
      revert GovernorUnexpectedProposalState(_proposalId, currentState, _allowedStates);
    }
    return currentState;
  }
}
```

Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist