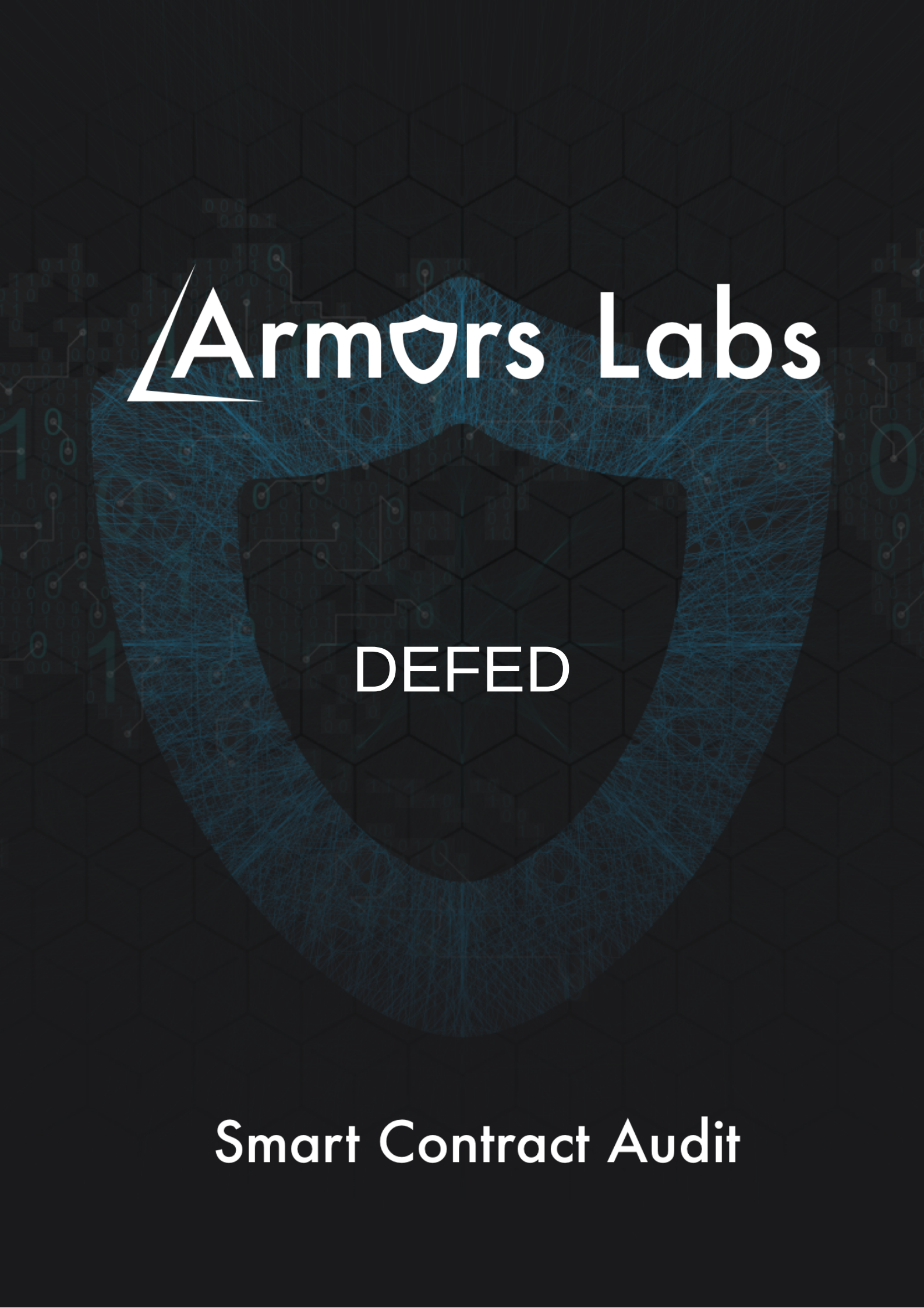

Armors Labs
DEFED
Smart Contract Audit

DEFED Audit Summary

Project name : DEFED Contract

Project address: None

Code URL : https://etherscan.io/address/0xd849476e251fdFDe7819aCDd1F260e708c60ffAf#code

Code URL : https://etherscan.io/address/0x3FF651Ac6e99F476FAD92BaC76307D7891411665#code

Code URL : https://polygonscan.com/address/0x012d15ca0b4362B9fB2308023453abe5CdccfB3b#code

Code URL : https://polygonscan.com/address/0xa626905698ac42b92C5e429c47C4D1c972b50e42#code

Code URL : https://polygonscan.com/address/0x032BB8Db7199952bb802E8e48819C0F463722019#code

Code URL : https://polygonscan.com/address/0xbc42F065C3f79dD12bA637085499045DB1bcE3C2#code

Code URL : https://polygonscan.com/address/0x86190771d9C4F74E3dEa5B0A7d5936BBDf7c9c0E#code

Commit : None

Project target : DEFED Contract Audit

Blockchain : Ethereum，Polygon

Test result : PASSED

Audit Info

Audit NO : 0X202209190026

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

DEFED Audit

The DEFED team asked us to review and audit their DEFED contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
DEFED Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2022-09-19

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the DEFED contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or "asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'…)

Armors Labs Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Audited target file

file	md5
./UserProxyFactory.sol	090db5b1c5d80a980b19707b90c1bc4e
./VToken.sol	c0ca992d47cbd0eade0c754314c5c4e3
./VTokenFactory.sol	ed640b70b9fcc1b3768737004ac862f2
./DefeToken.sol	154254582f3a120ca336c9566e581a6a
./UserProxy.sol	82a1fe81e21b3f7d92db25a4551484ed
./BridgeControl.sol	79f474d50198082ce8b29b9c4813282e
./TokenController.sol	4720438f0d48d96766ede9d4e6369225
./AssetManagement.sol	aa0a3b302571db50c0a6806642990fc5

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

```
// SPDX-License-Identifier: agpl-3.0
pragma solidity ^0.8.0;

import "./interfaces/IUserProxyFactory.sol";
import "./UserProxy.sol";

contract UserProxyFactory is IUserProxyFactory {
    mapping(address => address) public override getProxy;

    // // keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContrac
    // bytes32 private constant DOMAIN_SEPARATOR_TYPEHASH = 0x8b73c3c69bb8fe3d512ecc4cf759cc79239f7b1

    // keccak256("EIP712Domain(string name,string version,address verifyingContract)");
    bytes32 private constant DOMAIN_SEPARATOR_TYPEHASH =
    0x91ab3d17e3a50a9d89e63fd30b92be7f5336b03b287bb946787a83a9d62a2766;

    bytes32 public DOMAIN_SEPARATOR;
    string public constant name = "User Proxy Factory V1";
    string public constant VERSION = "1";

    constructor() {
        // uint chainId;
        // assembly {
        //     chainId := chainid()
```

```
        // }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                DOMAIN_SEPARATOR_TYPEHASH,
                keccak256(bytes(name)),
                keccak256(bytes(VERSION)),
                address(this)
            )
        );
    }

    function createProxy(address owner)
    external
    override
    returns (address proxy)
    {
        require(owner != address(0), "ZERO_ADDRESS");
        require(getProxy[owner] == address(0), "PROXY_EXISTS");
        bytes memory bytecode = proxyCreationCode();
        bytes32 salt = keccak256(abi.encodePacked(address(this), owner));
        assembly {
            proxy := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        getProxy[owner] = proxy;
        IUserProxy(proxy).initialize(owner, DOMAIN_SEPARATOR);
        emit ProxyCreated(owner, proxy);
    }

    function proxyRuntimeCode() public pure returns (bytes memory) {
        return type(UserProxy).runtimeCode;
    }

    function proxyCreationCode() public pure returns (bytes memory) {
        return type(UserProxy).creationCode;
    }

    function proxyCreationCodeHash() public pure returns (bytes32) {
        return keccak256(proxyCreationCode());
    }
}


// SPDX-License-Identifier: agpl-3.0
pragma solidity ^0.8.0;

import "./libraries/SafeMath.sol";
import "./libraries/Address.sol";
import "./libraries/Context.sol";
import "./interfaces/IVToken.sol";
import "./interfaces/IVTokenFactory.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract VToken is IVToken, Context, IERC20 {
    address public factory;
    address public override ETHToken;
    using SafeMath for uint256;
    using Address for address;
    mapping(address => uint256) private _balances;
    mapping(address => mapping(address => uint256)) private _allowances;
    uint256 private _totalSupply;
    string private _name;
    string private _symbol;
    uint8 private _decimals;
    event Mint(address indexed from, uint256 value);
    event Burn(address indexed from, uint256 value);
```

```
    constructor() {
        factory = msg.sender;
    }

    function initialize(
        address token,
        string memory tokenName,
        string memory tokenSymbol,
        uint8 tokenDecimals
    ) external override {
        require(msg.sender == factory, "FORBIDDEN");
        require(token != address(0));
        ETHToken = token;
        _name = tokenName;
        _symbol = tokenSymbol;
        _decimals = tokenDecimals;
    }

    uint256 private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, "LOCKED");
        unlocked = 0;
        _;
        unlocked = 1;
    }

    bytes4 private constant SELECTOR =
    bytes4(keccak256(bytes("transfer(address,uint256)")));

    function _safeTransfer(
        address token,
        address to,
        uint256 value
    ) private {
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(SELECTOR, to, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TRANSFER_FAILED"
        );
    }

    function mint(address spender, uint256 amount) external override {
        require(
            IVTokenFactory(factory).bridgeControl() == msg.sender,
            "Ownable: caller is not the owner"
        );
        _mint(spender, amount);
        emit Mint(spender, amount);
    }

    function burn(address spender, uint256 amount) external override {
        require(
            IVTokenFactory(factory).bridgeControl() == msg.sender,
            "Ownable: caller is not the owner"
        );
        _burn(spender, amount);
        emit Burn(spender, amount);
    }

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

    function symbol() public view returns (string memory) {
```

```
        return _symbol;
    }

    function decimals() public view override returns (uint8) {
        return _decimals;
    }

    function totalSupply() public view override returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
    }

    function transfer(address recipient, uint256 amount)
    public
    virtual
    override
    returns (bool)
    {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function allowance(address owner, address spender)
    public
    view
    virtual
    override
    returns (uint256)
    {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount)
    public
    virtual
    override
    returns (bool)
    {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(
            sender,
            _msgSender(),
            _allowances[sender][_msgSender()].sub(
                amount,
                "ERC20: transfer amount exceeds allowance"
            )
        );
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue)
    public
    virtual
    returns (bool)
```

```
    {
        _approve(
            _msgSender(),
            spender,
            _allowances[_msgSender()][spender].add(addedValue)
        );
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue)
    public
    virtual
    returns (bool)
    {
        _approve(
            _msgSender(),
            spender,
            _allowances[_msgSender()][spender].sub(
                subtractedValue,
                "ERC20: decreased allowance below zero"
            )
        );
        return true;
    }

    function _transfer(
        address sender,
        address recipient,
        uint256 amount
    ) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(
            amount,
            "ERC20: transfer amount exceeds balance"
        );
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(
            amount,
            "ERC20: burn amount exceeds balance"
        );
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
```

```
    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    function _setupDecimals(uint8 decimals_) internal {
        _decimals = decimals_;
    }

    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
}


// SPDX-License-Identifier: agpl-3.0
pragma solidity ^0.8.0;

import "./interfaces/IVTokenFactory.sol";
import "./VToken.sol";
import "./libraries/Ownable.sol";

contract VTokenFactory is IVTokenFactory, Ownable {
    mapping(address => address) public override getVToken;
    address public override bridgeControl;

    function createVToken(
        address token,
        string memory tokenName,
        string memory tokenSymbol,
        uint8 tokenDecimals
    ) public override onlyOwner returns (address vToken) {
        require(bridgeControl != address(0), "ZERO_ADDRESS");
        require(token != address(0), "ZERO_ADDRESS");
        require(getVToken[token] == address(0), "VTOKEN_EXISTS");
        bytes memory bytecode = creationCode();
        bytes32 salt = keccak256(abi.encodePacked(address(this), token));
        assembly {
            vToken := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        getVToken[token] = vToken;
        VToken(vToken).initialize(token, tokenName, tokenSymbol, tokenDecimals);
        emit VTokenCreated(token, vToken);
    }

    function setBridgeControl(address _bridgeControl)
    public
    override
    onlyOwner
    {
        require(_bridgeControl != address(0));
        bridgeControl = _bridgeControl;
    }

    function runtimeCode() public pure returns (bytes memory) {
        return type(VToken).runtimeCode;
    }
```

```
    function creationCode() public pure returns (bytes memory) {
        return type(VToken).creationCode;
    }

    function creationCodeHash() public pure returns (bytes32) {
        return keccak256(creationCode());
    }
}


// SPDX-License-Identifier: agpl-3.0
pragma solidity 0.8.0;
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

/**
 * @notice implementation of the DEFED token contract
 * @author DEFED
 */
contract DefeToken is ERC20 {
    string internal constant NAME = "DEFE Token";
    string internal constant SYMBOL = "DEFE";

    uint256 internal constant TOTAL_SUPPLY = 1e28;

    constructor(address misc) public ERC20(NAME, SYMBOL) {
        _mint(misc, TOTAL_SUPPLY);
    }
}


// SPDX-License-Identifier: agpl-3.0
pragma solidity ^0.8.0;

import "./interfaces/IUserProxy.sol";
import "./libraries/ECDSA.sol";
import "@openzeppelin/contracts/utils/StorageSlot.sol";

contract UserProxy is IUserProxy {
    mapping(uint256 => bool) public nonces;

    // keccak256("ExecTransaction(address to,uint256 value,bytes data,uint8 operation,uint256 nonce)"
    bytes32 internal constant EXEC_TX_TYPEHASH =
    0xa609e999e2804ed92314c0c662cfdb3c1d8107df2fb6f2e4039093f20d5e6250;
    // bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1)
    bytes32 internal constant ADMIN_SLOT =
    0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103;
    // bytes32(uint256(keccak256('eip1967.proxy.domain')) - 1)
    bytes32 internal constant DOMAIN_SLOT =
    0x5d29634e15c15fa29be556decae8ee5a34c9fee5f209623aed08a64bf865b694;

    function initialize(address _owner, bytes32 _DOMAIN_SEPARATOR)
    external
    override
    {
        require(owner() == address(0), "initialize error");
        require(_owner != address(0), "ERC1967: new owner is the zero address");
        StorageSlot.getAddressSlot(ADMIN_SLOT).value = _owner;
        StorageSlot.getBytes32Slot(DOMAIN_SLOT).value = _DOMAIN_SEPARATOR;
    }

    function owner() public view override returns (address) {
        return StorageSlot.getAddressSlot(ADMIN_SLOT).value;
    }

    function domain() public view returns (bytes32) {
```

```
        return StorageSlot.getBytes32Slot(DOMAIN_SLOT).value;
    }

    function execTransaction(
        address to,
        uint256 value,
        bytes calldata data,
        Operation operation,
        uint256 nonce,
        bytes memory signature
    ) external override {
        require(!nonces[nonce], "nonce had used");
        nonces[nonce] = true;
        bytes32 digest = keccak256(
            abi.encodePacked(
                "\x19\x01",
                domain(),
                keccak256(
                    abi.encode(
                        EXEC_TX_TYPEHASH,
                        to,
                        value,
                        keccak256(data),
                        operation,
                        nonce
                    )
                )
            )
        );
        address recoveredAddress = ECDSA.recover(digest, signature);
        require(
            recoveredAddress != address(0) && recoveredAddress == owner(),
            "ECDSA: invalid signature"
        );
        execute(to, value, data, operation);
    }

    receive() external payable {}

    fallback() external payable {}

    function execTransaction(
        address to,
        uint256 value,
        bytes calldata data,
        Operation operation
    ) external override {
        require(msg.sender == owner(), "ECDSA: invalid signature");
        execute(to, value, data, operation);
    }

    function execute(
        address to,
        uint256 value,
        bytes memory data,
        Operation operation
    ) internal {
        if (operation == Operation.DelegateCall) {
            assembly {
                let result := delegatecall(
                gas(),
                to,
                add(data, 0x20),
                mload(data),
                0,
                0
```

```
                )
                returndatacopy(0, 0, returndatasize())
                switch result
                case 0 {
                    revert(0, returndatasize())
                }
                default {
                    return(0, returndatasize())
                }
            }
        } else {
            assembly {
                let result := call(
                gas(),
                to,
                value,
                add(data, 0x20),
                mload(data),
                0,
                0
                )
                returndatacopy(0, 0, returndatasize())
                switch result
                case 0 {
                    revert(0, returndatasize())
                }
                default {
                    return(0, returndatasize())
                }
            }
        }
    }
}


// SPDX-License-Identifier: agpl-3.0
pragma solidity ^0.8.0;
pragma experimental ABIEncoderV2;

import "./interfaces/IUserProxy.sol";
import "./interfaces/IUserProxyFactory.sol";
import "./interfaces/IVTokenFactory.sol";
import "./interfaces/IVToken.sol";
import "./interfaces/ILendingPool.sol";
import "./libraries/Ownable.sol";
import "./interfaces/IBridgeFeeController.sol";
import "./interfaces/IIncentivesController.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

interface IVeDEFE {
    struct LockedBalance {
        int256 amount;
        uint256 end;
    }

    function createLockFor(
        address _beneficiary,
        uint256 _value,
        uint256 _unlockTime
    ) external;

    function increaseAmountFor(address _beneficiary, uint256 _value) external;

    function getLocked(address _addr)
    external
```

```
    view
    returns (LockedBalance memory);
}

contract BridgeControl is Ownable {
    using SafeERC20 for IERC20;
    address public proxyFactory;
    address public vTokenFactory;
    address public lendingPool;
    address public virtualDefedToken;
    address public bridgeFeeController;
    address public veDEFE;
    mapping(bytes32 => bool) transactions;

    event TransferToEthereum(
        address indexed fromEthAdr,
        address indexed toEthAdr,
        address indexed toProxyAdr,
        address token,
        address vToken,
        uint256 value,
        uint256 action
    );
    event TransferFromEthereum(
        address indexed fromEthAdr,
        address indexed fromProxyAdr,
        address token,
        address vToken,
        uint256 value,
        bytes32 transactionId
    );
    event TransferFromEthereumForDeposit(
        address indexed fromEthAdr,
        address indexed fromProxyAdr,
        address token,
        address vToken,
        uint256 value,
        bytes32 transactionId
    );
    event TransferFromEthereumForRepay(
        address indexed fromEthAdr,
        address indexed fromProxyAdr,
        address token,
        address vToken,
        uint256 value,
        bytes32 transactionId
    );
    event lockFromEthereumLog(
        address indexed fromEthAdr,
        address indexed fromProxyAdr,
        address virtualDefedToken,
        uint256 value,
        uint256 time,
        bytes32 transactionId
    );
    event BridgeFeeLog(
        address indexed fromUserProxy,
        address token,
        uint256 fee
    );

    constructor(
        address _proxyFactory,
        address _vTokenFactory,
        address _lendingPool,
        address _bridgeFeeController
```

```
    ) {
        require(_proxyFactory != address(0));
        require(_vTokenFactory != address(0));
        require(_lendingPool != address(0));
        require(_bridgeFeeController != address(0));
        proxyFactory = _proxyFactory;
        vTokenFactory = _vTokenFactory;
        lendingPool = _lendingPool;
        bridgeFeeController = _bridgeFeeController;
    }

    function setVirtualDefedToken(address _virtualDefedToken, address _veDEFE)
    external
    onlyOwner
    {
        require(_virtualDefedToken != address(0));
        require(_veDEFE != address(0));
        virtualDefedToken = _virtualDefedToken;
        veDEFE = _veDEFE;
    }

    function turnOutToken(address token, uint256 amount) public onlyOwner {
        IERC20(token).safeTransfer(msg.sender, amount);
    }

    function transferToEthereum(
        address from,
        address vToken,
        address to,
        uint256 amount,
        uint256 action
    ) external {
        address fromEthAddr = IUserProxy(from).owner();
        address toEthAddr = IUserProxy(to).owner();
        require(fromEthAddr != address(0), "from PROXY_EXISTS");
        require(toEthAddr != address(0), "to PROXY_EXISTS");
        address token = IVToken(vToken).ETHToken();
        require(token != address(0), "unknow token");
        (uint256 fee, address bridgeFeeVault) = IBridgeFeeController(
            bridgeFeeController
        ).getBridgeFee(vToken, amount);
        if (fee > 0) {
            IERC20(vToken).safeTransfer(bridgeFeeVault, fee);
            emit BridgeFeeLog(from, vToken, fee);
        }
        uint256 targetAmount = amount - fee;
        IVToken(vToken).burn(address(this), targetAmount);
        emit TransferToEthereum(
            fromEthAddr,
            toEthAddr,
            to,
            token,
            vToken,
            targetAmount,
            action
        );
    }

    function transferFromEthereumForDeposit(
        bytes32 transactionId,
        address token,
        address to,
        uint256 amount
    ) public onlyOwner {
        require(!transactions[transactionId], "transactionId already exec");
        transactions[transactionId] = true;
```

```
        address vToken = IVTokenFactory(vTokenFactory).getVToken(token);
        require(vToken != address(0), "unknow token");
        address proxyAddr = IUserProxyFactory(proxyFactory).getProxy(to);
        if (proxyAddr == address(0)) {
            proxyAddr = IUserProxyFactory(proxyFactory).createProxy(to);
        }
        IVToken(vToken).mint(address(this), amount);
        IERC20(vToken).approve(lendingPool, amount);
        ILendingPool(lendingPool).deposit(vToken, amount, proxyAddr, 0);
        emit TransferFromEthereumForDeposit(
            to,
            proxyAddr,
            token,
            vToken,
            amount,
            transactionId
        );
    }

    function transferFromEthereumForRepay(
        bytes32 transactionId,
        address token,
        address to,
        uint256 amount,
        uint256 rateMode
    ) public onlyOwner {
        require(!transactions[transactionId], "transactionId already exec");
        transactions[transactionId] = true;

        address vToken = IVTokenFactory(vTokenFactory).getVToken(token);
        require(vToken != address(0), "unknow token");
        address proxyAddr = IUserProxyFactory(proxyFactory).getProxy(to);
        if (proxyAddr == address(0)) {
            proxyAddr = IUserProxyFactory(proxyFactory).createProxy(to);
        }
        IVToken(vToken).mint(address(this), amount);
        IERC20(vToken).approve(lendingPool, amount);
        ILendingPool(lendingPool).repay(vToken, amount, rateMode, proxyAddr);
        uint256 balanceAfterRepay = IERC20(vToken).balanceOf(address(this));
        if (balanceAfterRepay > 0) {
            ILendingPool(lendingPool).deposit(
                vToken,
                balanceAfterRepay,
                proxyAddr,
                0
            );
        }
        emit TransferFromEthereumForRepay(
            to,
            proxyAddr,
            token,
            vToken,
            amount,
            transactionId
        );
    }

    function transferFromEthereum(
        bytes32 transactionId,
        address token,
        address to,
        uint256 amount
    ) public onlyOwner {
        require(!transactions[transactionId], "transactionId already exec");
        transactions[transactionId] = true;
```

```
        address vToken = IVTokenFactory(vTokenFactory).getVToken(token);
        require(vToken != address(0), "unknow token");
        address proxyAddr = IUserProxyFactory(proxyFactory).getProxy(to);
        if (proxyAddr == address(0)) {
            proxyAddr = IUserProxyFactory(proxyFactory).createProxy(to);
        }
        IVToken(vToken).mint(proxyAddr, amount);
        emit TransferFromEthereum(
            to,
            proxyAddr,
            token,
            vToken,
            amount,
            transactionId
        );
    }

    function lockFromEthereum(
        bytes32 transactionId,
        address user,
        uint256 amount,
        uint256 time
    ) public onlyOwner {
        require(!transactions[transactionId], "transactionId already exec");
        transactions[transactionId] = true;

        address proxyAddr = IUserProxyFactory(proxyFactory).getProxy(user);
        if (proxyAddr == address(0)) {
            proxyAddr = IUserProxyFactory(proxyFactory).createProxy(user);
        }
        lockDefe(proxyAddr, amount, time);
        emit lockFromEthereumLog(
            user,
            proxyAddr,
            virtualDefedToken,
            amount,
            time,
            transactionId
        );
    }

    function lockDefe(
        address proxyAddr,
        uint256 amount,
        uint256 time
    ) internal {
        IVeDEFE.LockedBalance memory locked = IVeDEFE(veDEFE).getLocked(
            proxyAddr
        );
        IERC20(virtualDefedToken).approve(veDEFE, amount);
        if (locked.amount == 0) {
            IVeDEFE(veDEFE).createLockFor(proxyAddr, amount, time);
        } else {
            IVeDEFE(veDEFE).increaseAmountFor(proxyAddr, amount);
        }
    }
}


// SPDX-License-Identifier: agpl-3.0
pragma solidity ^0.8.0;
pragma experimental ABIEncoderV2;

import "./interfaces/IUserProxyFactory.sol";
import "./interfaces/IUserProxy.sol";
```

```
import "./interfaces/IVTokenFactory.sol";
import "./interfaces/ILendingPool.sol";
import "./interfaces/IBridgeControl.sol";
import "./interfaces/ITokenController.sol";
import "./interfaces/INetworkFeeController.sol";
import "./interfaces/IIncentivesController.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract TokenController {
    using SafeERC20 for IERC20;
    struct Params {
        address lendingPool;
        address bridgeControl;
        address vTokenFactory;
        address proxyFactory;
        address networkFeeController;
    }
    mapping(address => Params) public addressParams;

    event BorrowToEthereum(address asset, uint256 value, address toEthAdr);

    event Borrow(address asset, uint256 value, address toEthAdr);

    event Repay(address asset, uint256 value, uint256 rateMode);

    event WithdrawToEthereum(address asset, uint256 value, address toEthAdr);

    event Transfer(address asset, uint256 value, address toEthAdr);

    event TransferToEthereum(address asset, uint256 value, address toEthAdr);

    event TransferCredit(
        address asset,
        uint256 value,
        address toEthAdr,
        uint256 interestRateMode,
        uint16 referralCode
    );

    event TransferCreditToEthereum(
        address asset,
        uint256 value,
        address toEthAdr,
        uint256 interestRateMode,
        uint16 referralCode
    );

    event NetworkFeeLog(
        address fromUserProxy,
        address token,
        uint256 fee,
        uint256 action
    );

    constructor(
        address _lendingPOOL,
        address _bridgeControl,
        address _vTokenFactory,
        address _proxyFactory,
        address _networkFeeController
    ) {
        address tokenController = address(this);
        addressParams[tokenController].lendingPool = _lendingPOOL;
        addressParams[tokenController].bridgeControl = _bridgeControl;
        addressParams[tokenController].vTokenFactory = _vTokenFactory;
```

```
        addressParams[tokenController].proxyFactory = _proxyFactory;
        addressParams[tokenController]
        .networkFeeController = _networkFeeController;
    }

    function withdrawToEthereum(
        address tokenController,
        address asset,
        uint256 amount
    ) public {
        bytes4 method = bytes4(
            keccak256("withdrawToEthereum(address,address,uint256)")
        );
        Params memory params = TokenController(tokenController).getParams();
        address vToken = IVTokenFactory(params.vTokenFactory).getVToken(asset);
        address ethUser = IUserProxy(address(this)).owner();
        require(vToken != address(0), "unknow token");
        ILendingPool(params.lendingPool).withdraw(
            vToken,
            amount,
            address(this)
        );
        (uint256 fee, address networkFeeVault) = INetworkFeeController(
            params.networkFeeController
        ).getNetworkFee(ethUser, method, vToken, amount);
        if (fee > 0) {
            IERC20(vToken).safeTransfer(networkFeeVault, fee);
            emit NetworkFeeLog(address(this), vToken, fee, 1);
        }
        uint256 targetAmount = amount - fee;
        IERC20(vToken).safeTransfer(params.bridgeControl, targetAmount);
        IBridgeControl(params.bridgeControl).transferToEthereum(
            address(this),
            vToken,
            address(this),
            targetAmount,
            1
        );
        emit WithdrawToEthereum(asset, targetAmount, ethUser);
    }

    function borrowToEthereum(
        address tokenController,
        address asset,
        uint256 amount,
        uint256 interestRateMode,
        uint16 referralCode
    ) public {
        bytes4 method = bytes4(
            keccak256(
                "borrowToEthereum(address,address,uint256,uint256,uint16)"
            )
        );
        Params memory params = TokenController(tokenController).getParams();
        address vToken = IVTokenFactory(params.vTokenFactory).getVToken(asset);
        require(vToken != address(0), "unknow token");
        address ethUser = IUserProxy(address(this)).owner();
        ILendingPool(params.lendingPool).borrow(
            vToken,
            amount,
            interestRateMode,
            referralCode,
            address(this)
        );
        (uint256 fee, address networkFeeVault) = INetworkFeeController(
            params.networkFeeController
```

```
        ).getNetworkFee(ethUser, method, vToken, amount);
        if (fee > 0) {
            IERC20(vToken).safeTransfer(networkFeeVault, fee);
            emit NetworkFeeLog(address(this), vToken, fee, 2);
        }
        uint256 targetAmount = amount - fee;
        IERC20(vToken).safeTransfer(params.bridgeControl, targetAmount);
        IBridgeControl(params.bridgeControl).transferToEthereum(
            address(this),
            vToken,
            address(this),
            targetAmount,
            2
        );
        emit BorrowToEthereum(asset, targetAmount, ethUser);
    }

    function borrow(
        address tokenController,
        address asset,
        uint256 amount,
        uint256 interestRateMode,
        uint16 referralCode
    ) public {
        bytes4 method = bytes4(
            keccak256("borrow(address,address,uint256,uint256,uint16)")
        );
        Params memory params = TokenController(tokenController).getParams();
        address vToken = IVTokenFactory(params.vTokenFactory).getVToken(asset);
        require(vToken != address(0), "unknow token");
        address ethUser = IUserProxy(address(this)).owner();
        ILendingPool(params.lendingPool).borrow(
            vToken,
            amount,
            interestRateMode,
            referralCode,
            address(this)
        );
        (uint256 fee, address networkFeeVault) = INetworkFeeController(
            params.networkFeeController
        ).getNetworkFee(ethUser, method, vToken, amount);
        if (fee > 0) {
            IERC20(vToken).safeTransfer(networkFeeVault, fee);
            emit NetworkFeeLog(address(this), vToken, fee, 3);
        }
        uint256 targetAmount = amount - fee;
        IERC20(vToken).approve(params.lendingPool, targetAmount);
        ILendingPool(params.lendingPool).deposit(
            vToken,
            targetAmount,
            address(this),
            referralCode
        );
        emit Borrow(asset, targetAmount, ethUser);
    }

    function transfer(
        address tokenController,
        address asset,
        uint256 amount,
        address to
    ) public {
        bytes4 method = bytes4(
            keccak256("transfer(address,address,uint256,address)")
        );
        Params memory params = TokenController(tokenController).getParams();
```

```
        address vToken = IVTokenFactory(params.vTokenFactory).getVToken(asset);
        require(vToken != address(0), "unknow token");
        address proxyAddr = IUserProxyFactory(params.proxyFactory).getProxy(to);
        if (proxyAddr == address(0)) {
            proxyAddr = IUserProxyFactory(params.proxyFactory).createProxy(to);
        }
        address ethUser = IUserProxy(address(this)).owner();
        (uint256 fee, address networkFeeVault) = INetworkFeeController(
            params.networkFeeController
        ).getNetworkFee(ethUser, method, vToken, amount);
        if (fee > 0) {
            ILendingPool(params.lendingPool).withdraw(
                vToken,
                fee,
                networkFeeVault
            );
            emit NetworkFeeLog(address(this), vToken, fee, 4);
        }
        uint256 targetAmount = amount - fee;
        (, , , , , , , address aToken, , , , ) = ILendingPool(
            params.lendingPool
        ).getReserveData(vToken);
        IERC20(aToken).safeTransfer(proxyAddr, targetAmount);
        emit Transfer(asset, targetAmount, to);
    }

    function transferToEthereum(
        address tokenController,
        address asset,
        uint256 amount,
        address to
    ) public {
        bytes4 method = bytes4(
            keccak256("transferToEthereum(address,address,uint256,address)")
        );
        Params memory params = TokenController(tokenController).getParams();
        address vToken = IVTokenFactory(params.vTokenFactory).getVToken(asset);
        require(vToken != address(0), "unknow token");
        address proxyAddr = IUserProxyFactory(params.proxyFactory).getProxy(to);
        if (proxyAddr == address(0)) {
            proxyAddr = IUserProxyFactory(params.proxyFactory).createProxy(to);
        }
        address ethUser = IUserProxy(address(this)).owner();
        ILendingPool(params.lendingPool).withdraw(
            vToken,
            amount,
            address(this)
        );
        (uint256 fee, address networkFeeVault) = INetworkFeeController(
            params.networkFeeController
        ).getNetworkFee(ethUser, method, vToken, amount);
        if (fee > 0) {
            IERC20(vToken).safeTransfer(networkFeeVault, fee);
            emit NetworkFeeLog(address(this), vToken, fee, 5);
        }
        uint256 targetAmount = amount - fee;
        IERC20(vToken).safeTransfer(params.bridgeControl, targetAmount);
        IBridgeControl(params.bridgeControl).transferToEthereum(
            address(this),
            vToken,
            proxyAddr,
            targetAmount,
            3
        );
        emit TransferToEthereum(asset, targetAmount, to);
    }
```

```
function transferCredit(
    address tokenController,
    address asset,
    uint256 amount,
    address to,
    uint256 interestRateMode,
    uint16 referralCode
) public {
    bytes4 method = bytes4(
        keccak256(
            "transferCredit(address,address,uint256,address,uint256,uint16)"
        )
    );
    Params memory params = TokenController(tokenController).getParams();
    address vToken = IVTokenFactory(params.vTokenFactory).getVToken(asset);
    require(vToken != address(0), "unknow token");
    address proxyAddr = IUserProxyFactory(params.proxyFactory).getProxy(to);
    address ethUser = IUserProxy(address(this)).owner();
    if (proxyAddr == address(0)) {
        proxyAddr = IUserProxyFactory(params.proxyFactory).createProxy(to);
    }
    ILendingPool(params.lendingPool).borrow(
        vToken,
        amount,
        interestRateMode,
        referralCode,
        address(this)
    );
    (uint256 fee, address networkFeeVault) = INetworkFeeController(
        params.networkFeeController
    ).getNetworkFee(ethUser, method, vToken, amount);
    if (fee > 0) {
        IERC20(vToken).safeTransfer(networkFeeVault, fee);
        emit NetworkFeeLog(address(this), vToken, fee, 6);
    }
    uint256 targetAmount = amount - fee;
    IERC20(vToken).approve(params.lendingPool, targetAmount);
    ILendingPool(params.lendingPool).deposit(
        vToken,
        targetAmount,
        proxyAddr,
        referralCode
    );
    emit TransferCredit(
        asset,
        targetAmount,
        to,
        interestRateMode,
        referralCode
    );
}

function transferCreditToEthereum(
    address tokenController,
    address asset,
    uint256 amount,
    address to,
    uint256 interestRateMode,
    uint16 referralCode
) public {
    bytes4 method = bytes4(
        keccak256(
            "transferCreditToEthereum(address,address,uint256,address,uint256,uint16)"
        )
    );
```

```
        Params memory params = TokenController(tokenController).getParams();
        address vToken = IVTokenFactory(params.vTokenFactory).getVToken(asset);
        require(vToken != address(0), "unknow token");
        address proxyAddr = IUserProxyFactory(params.proxyFactory).getProxy(to);
        address ethUser = IUserProxy(address(this)).owner();
        if (proxyAddr == address(0)) {
            proxyAddr = IUserProxyFactory(params.proxyFactory).createProxy(to);
        }
        ILendingPool(params.lendingPool).borrow(
            vToken,
            amount,
            interestRateMode,
            referralCode,
            address(this)
        );
        (uint256 fee, address networkFeeVault) = INetworkFeeController(
            params.networkFeeController
        ).getNetworkFee(ethUser, method, vToken, amount);
        if (fee > 0) {
            IERC20(vToken).safeTransfer(networkFeeVault, fee);
            emit NetworkFeeLog(address(this), vToken, fee, 7);
        }
        uint256 targetAmount = amount - fee;
        IERC20(vToken).safeTransfer(params.bridgeControl, targetAmount);
        IBridgeControl(params.bridgeControl).transferToEthereum(
            address(this),
            vToken,
            proxyAddr,
            targetAmount,
            4
        );
        emit TransferCreditToEthereum(
            asset,
            targetAmount,
            to,
            interestRateMode,
            referralCode
        );
    }

    function repay(
        address tokenController,
        address asset,
        uint256 amount,
        uint256 rateMode
    ) public {
        bytes4 method = bytes4(
            keccak256("repay(address,address,uint256,uint256)")
        );
        Params memory params = TokenController(tokenController).getParams();
        address vToken = IVTokenFactory(params.vTokenFactory).getVToken(asset);
        require(vToken != address(0), "unknow token");
        ILendingPool(params.lendingPool).withdraw(
            vToken,
            amount,
            address(this)
        );
        address ethUser = IUserProxy(address(this)).owner();
        (uint256 fee, address networkFeeVault) = INetworkFeeController(
            params.networkFeeController
        ).getNetworkFee(ethUser, method, vToken, amount);
        if (fee > 0) {
            IERC20(vToken).safeTransfer(networkFeeVault, fee);
            emit NetworkFeeLog(address(this), vToken, fee, 8);
        }
        uint256 targetAmount = amount - fee;
```

```
        ILendingPool(params.lendingPool).repay(
            vToken,
            targetAmount,
            rateMode,
            address(this)
        );
        uint256 balanceAfterRepay = IERC20(vToken).balanceOf(address(this));
        if (balanceAfterRepay != 0) {
            ILendingPool(params.lendingPool).deposit(
                vToken,
                balanceAfterRepay,
                address(this),
                0
            );
        }
        emit Repay(asset, targetAmount, rateMode);
    }

    function getParams() external view returns (Params memory) {
        return addressParams[address(this)];
    }
}


// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
pragma experimental ABIEncoderV2;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";

interface IWETH {
    function deposit() external payable;

    function withdraw(uint256) external;
}

interface IERC721 {
    function mint(address to) external;
}

contract AssetManagement is Ownable {
    using SafeERC20 for IERC20;
    mapping(address => bool) public activeTokens;
    address[] private contracts;
    mapping(address => bool) public deposited;
    mapping(bytes32 => bool) transactions;
    address public WETH;
    address public BANKCARDNFT;
    uint256 public lastTokenId;

    event Deposit(address sender, address token, uint256 value);
    event DepositForRepay(address sender, address token, uint256 value);
    event Widthdraw(
        address reciver,
        address token,
        uint256 value,
        string action,
        bytes32 transactionId
    );
    event WidthdrawETH(
        address reciver,
        uint256 value,
        string action,
        bytes32 transactionId
```

```
    );
    event ActiveToken(address token);
    event PauseToken(address token);
    event ChangeSigner(address signer, bool flag);
    event FeeChange(uint256 fee);

    constructor(address _weth, address _bankCardNFT) {
        require(_weth != address(0));
        require(_bankCardNFT != address(0));
        activeTokens[_weth] = true;
        contracts.push(_weth);
        WETH = _weth;
        BANKCARDNFT = _bankCardNFT;
    }

    function deposit(address token, uint256 amount) external {
        require(amount > 0, "Deposit: amount can not be 0");
        if (!deposited[msg.sender]) {
            deposited[msg.sender] = true;
            _mintNFT(msg.sender);
        }
        require(activeTokens[token], "Deposit: token not support");
        IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
        emit Deposit(msg.sender, token, amount);
    }

    function depositForRepay(address token, uint256 amount) external {
        require(amount > 0, "DepositForRepay: amount can not be 0");
        if (!deposited[msg.sender]) {
            deposited[msg.sender] = true;
            _mintNFT(msg.sender);
        }
        require(activeTokens[token], "DepositForRepay: token not support");
        IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
        emit DepositForRepay(msg.sender, token, amount);
    }

    function depositETHForRepay() external payable {
        require(msg.value > 0, "DepositETHForRepay: amount  zero");
        if (!deposited[msg.sender]) {
            deposited[msg.sender] = true;
            _mintNFT(msg.sender);
        }
        IWETH(WETH).deposit{value: msg.value}();
        emit DepositForRepay(msg.sender, WETH, msg.value);
    }

    function depositETH() external payable {
        require(msg.value > 0, "DepositETH: amount  zero");
        if (!deposited[msg.sender]) {
            deposited[msg.sender] = true;
            _mintNFT(msg.sender);
        }
        IWETH(WETH).deposit{value: msg.value}();
        emit Deposit(msg.sender, WETH, msg.value);
    }

    function withdraw(
        bytes32 transactionId,
        address token,
        address to,
        uint256 amount,
        string memory action
    ) public onlyOwner {
        require(!transactions[transactionId], "repeat transactionId ");
        transactions[transactionId] = true;
```

```
        IERC20(token).safeTransfer(to, amount);
        emit Widthdraw(to, token, amount, action, transactionId);
    }

    function withdrawETH(
        bytes32 transactionId,
        address to,
        uint256 amount,
        string memory action
    ) public onlyOwner {
        require(!transactions[transactionId], "repeat transactionId ");
        transactions[transactionId] = true;
        IWETH(WETH).withdraw(amount);
        _safeTransferETH(to, amount);
        emit WidthdrawETH(to, amount, action, transactionId);
    }

    function activeToken(address token) external onlyOwner {
        require(!activeTokens[token], "AddToken: token already supported");
        contracts.push(token);
        activeTokens[token] = true;
        emit ActiveToken(token);
    }

    function pauseToken(address token) external onlyOwner {
        require(activeTokens[token], "PauseToken: token not active");
        activeTokens[token] = false;
        emit PauseToken(token);
    }

    function supportTokens() public view returns (address[] memory) {
        return contracts;
    }

    function userWalletBalance(address user)
    public
    view
    returns (
        address[] memory,
        uint256[] memory,
        uint256
    )
    {
        uint256[] memory balances = new uint256[](contracts.length);
        for (uint256 i = 0; i < contracts.length; i++) {
            balances[i] = IERC20(contracts[i]).balanceOf(user);
        }
        uint256 ETHBalance = user.balance;
        return (contracts, balances, ETHBalance);
    }

    function _safeTransferETH(address to, uint256 value) internal {
        (bool success, ) = to.call{value: value}(new bytes(0));
        require(success, "ETH_TRANSFER_FAILED");
    }

    function _mintNFT(address to) internal {
        IERC721(BANKCARDNFT).mint(to);
    }

    fallback() external payable {
        revert("Fallback not allowed");
    }

    /**
     * @dev Only WETH contract is allowed to transfer ETH here. Prevent other addresses to send Ether
```

```
     */
    receive() external payable {
        require(msg.sender == address(WETH), "Receive not allowed");
    }
}
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

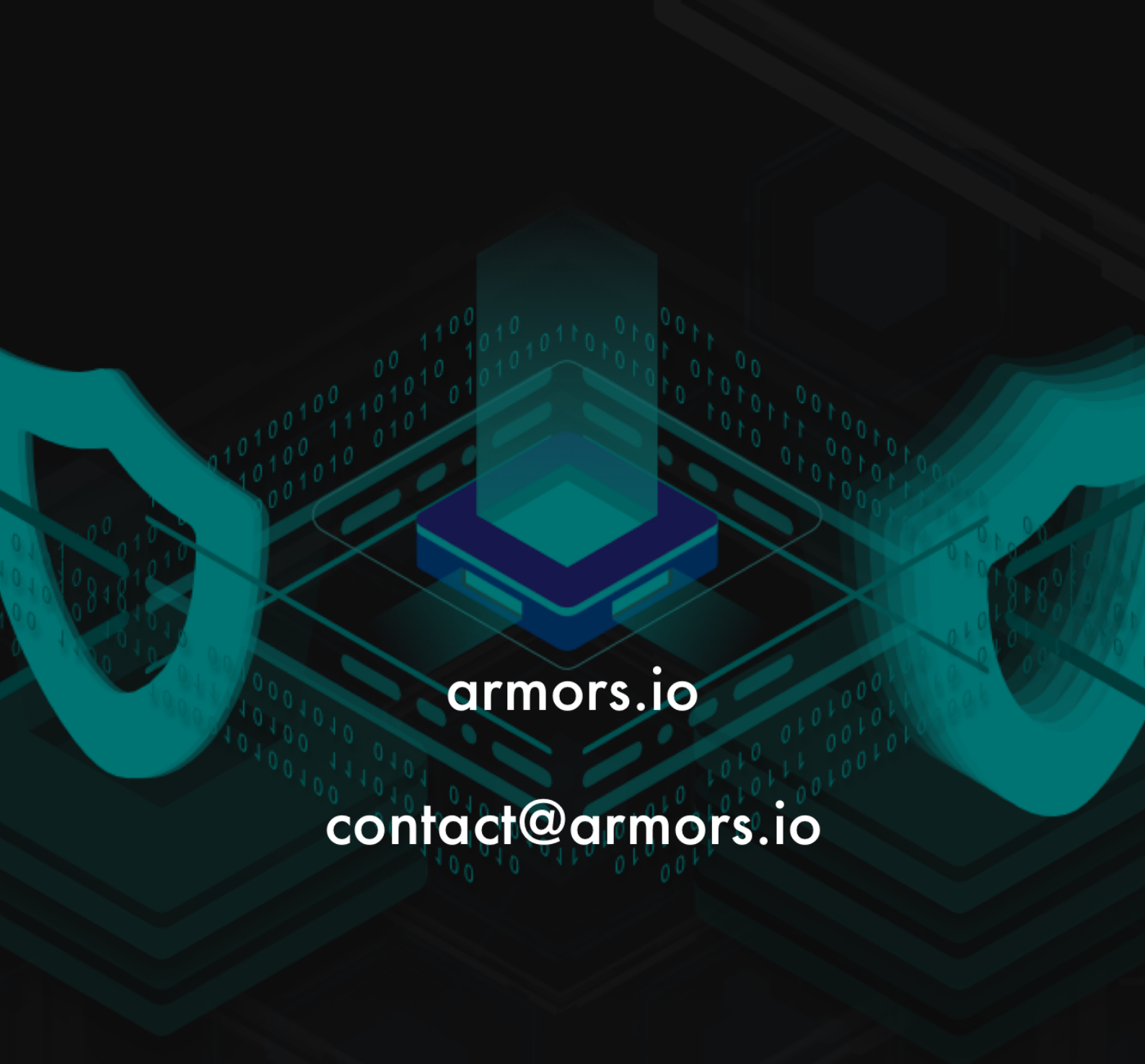
- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.



armors.io

contact@armors.io