Guard

Smart contracts security assessment

Final report

Tariff: Standard

Me3

March 2024

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for **Me3**.

Me3Token is an ERC-20 standard token with delegation, checkpoint tracking, and permit extensions. The token has no mint functionality, no taxes.

The code is available at the @me3wallet/Me3-contracts GitHub repository and was audited after the commit 649f50c8601f56750efdd38f677619c732502b9d.

The audit scope includes the following contracts:

Me3Token.sol and its dependencies.

Report Update.

The contract's code was updated according to this report and rechecked after the commit 5130602deef582bc502ac49f207548a2ee8472f3.

The token has been deployed to the BSC network at address 0x42b4daa9210102707373f4618852f926a4150277.

Name	Me3
Audit date	2024-03-01 - 2024-03-05
Language	Solidity
Platform	Binance Smart Chain

Contracts checked

Name	Address
Me3Token	0x42b4daa9210102707373f4618852f926a4150277

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyze smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed

Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity — High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.

Medium severity — Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.

Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. Mint is open for owner (Me3Token)

Status: Fixed

The contract privileged addresses selected by the owner can mint an arbitrary number of tokens.

```
function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {
    _mint(_to, _amount);
    return true;
}

function grantAccess(address minter) public onlyOwner {
    bool hasAccess = minters[minter];

    require(hasAccess == false, "minter has already been granted access");
    minters[minter] = true;

    emit AccessGranted(msg.sender, minter);
}
```

Recommendation: Secure ownership by using the Timelock contract with a Multisig admin.

Remediation: The team has removed the minting functionality of the token. All supply is minted on the token deployment.

2. Token can be burnt without permission (Me3Token)

Status: Fixed

The contract owner can burn user's tokens without an explicit permission or approval.

```
function destroyBlackFunds(address _blackListedUser) public onlyOwner {
    require(isBlackListed[_blackListedUser]);
    uint256 dirtyFunds = balanceOf(_blackListedUser);
    _balances[_blackListedUser] = 0;
    _totalSupply -= dirtyFunds;
    emit DestroyedBlackFunds(_blackListedUser, dirtyFunds);
}
```

Recommendation: Secure ownership by using the Timelock contract with a Multisig admin.

Remediation: The function has been removed from the contract.

Medium severity issues

1. Dangerous safecast (Me3Token)

Status: Fixed

Part of the calculations is forked from the Compound COMP token, which uses `uint96` for balances. The audited contract Me3Token stores balances as `uint256` and most of the calculations are moved to `uint256` math, but the function `safe96`, used for safecasting `uint256` to `uint96`, remains the same. This function is called in the `transfer`, `transferFrom`, `burnFrom`, `approve` functions. In the current state the Me3Token total supply should not exceed 2^96 ~= 79B with 18 decimals.

```
function safe96(uint256 n, string memory errorMessage) internal pure returns (uint256)
{
    require(n < 2**96, errorMessage);
    return uint256(n);
}
```

Recommendation: Remove the `safe96` function.

Remediation: The supply of the token has been capped by the initial supply.

2. Standard violation (Me3Token)

Status: Fixed

The ERC-20 token standard requires that Transfer event must trigger when tokens are transferred. The function destroyBlackFunds are used by the owner to burn tokens without user's permission, but the Transfer event is not emitted.

```
function destroyBlackFunds(address _blackListedUser) public onlyOwner {
    require(isBlackListed[_blackListedUser]);
    uint256 dirtyFunds = balanceOf(_blackListedUser);
    _balances[_blackListedUser] = 0;
    _totalSupply -= dirtyFunds;
    emit DestroyedBlackFunds(_blackListedUser, dirtyFunds);
}
```

Recommendation: Add `emit Transfer(_blackListedUser, address(0), dirtyFunds)`.

Remediation: The function has been removed from the token contract.

Low severity issues

1. Permit and delegateBySig nonce collision (Me3Token)

Status: Open

The fucntions `permit` and `delegateBySig` allow setting allowance and delegatee by user's signature via EIP-712. Both functions share the same `mapping(address user => uint256 nonce) public nonces` that is included into signed structure. If user wants to sign transactions for permit and delegation, he must assume the execution order as the same nonce `nonces[user]` will be iterated after each successful transaction of either `permit` or `delegateBySig` functions.

```
mapping(address => uint256) public nonces;

function delegateBySig(
    address delegatee,
    uint256 nonce,
    uint256 expiry,
    uint8 v,
    bytes32 r,
```

```
    bytes32 s
) public {
    bytes32 structHash = keccak256(abi.encode(DELEGATION_TYPEHASH, delegatee, nonce,
expiry));
    bytes32 digest = keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR(),
structHash));
    address signatory = ecrecover(digest, v, r, s);
    require(signatory != address(0), "Me3Token::delegateBySig: invalid signature");
    require(nonce == nonces[signatory]++, "Me3Token::delegateBySig: invalid nonce");
    require(block.timestamp <= expiry, "Me3Token::delegateBySig: signature expired");
    _delegate(signatory, delegatee);
}

function permit(
    address owner,
    address spender,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) public {
    bytes32 structHash = keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value,
nonces[owner]++, deadline));
    bytes32 digest = keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR(),
structHash));
    require(owner == ecrecover(digest, v, r, s), "Me3Token::permit: invalid
signature");
    require(owner != address(0), "Me3Token::permit: invalid signature");
    require(block.timestamp <= deadline, "Me3Token::permit: signature expired");
    _approve(owner, spender, value);
}
```

Conclusion

Me3 Me3Token contract was audited. 2 high, 2 medium, 1 low severity issues were found.

2 high, 2 medium severity issues have been fixed in the update.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability)set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

Slither output

```
INFO:Detectors:
Me3Token._writeCheckpoint(address,uint32,uint256,uint256) (contracts/contracts/token/
Me3Token.sol#403-419) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock ==
blockNumber (contracts/contracts/token/Me3Token.sol#411)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-
strict-equalities
INFO:Detectors:
Me3Token.allowance(address,address).owner (contracts/contracts/token/Me3Token.sol#136)
shadows:
        - Ownable.owner (contracts/contracts/libs/Ownable.sol#20) (state variable)
Me3Token._approve(address,address,uint256).owner (contracts/contracts/token/
Me3Token.sol#165) shadows:
        - Ownable.owner (contracts/contracts/libs/Ownable.sol#20) (state variable)
Me3Token.permit(address,address,uint256,uint256,uint8,bytes32,bytes32).owner (contracts/
contracts/token/Me3Token.sol#289) shadows:
        - Ownable.owner (contracts/contracts/libs/Ownable.sol#20) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-
shadowing
INFO:Detectors:
Me3Token.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (contracts/
contracts/token/Me3Token.sol#262-277) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,Me3Token::delegateBySig:
signature expired) (contracts/contracts/token/Me3Token.sol#275)
Me3Token.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (contracts/
contracts/token/Me3Token.sol#288-303) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,Me3Token::permit: signature
expired) (contracts/contracts/token/Me3Token.sol#301)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-
timestamp
INFO:Detectors:
Me3Token.getChainId() (contracts/contracts/token/Me3Token.sol#450-457) uses assembly
        - INLINE ASM (contracts/contracts/token/Me3Token.sol#453-455)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
```

```
Context._msgData() (contracts/contracts/libs/Context.sol#20-23) is never used and
should be removed
Me3Token._setupDecimals(uint8) (contracts/contracts/token/Me3Token.sol#147-149) is
never used and should be removed
INFO:Detectors:
Pragma version^0.7.0 (contracts/contracts/token/Me3Token.sol#3) allows old versions
solc-0.7.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-
versions-of-solidity
INFO:Detectors:
Function Me3Token.DOMAIN_SEPARATOR() (contracts/contracts/token/Me3Token.sol#129-131)
is not in mixedCase
Parameter Me3Token.getBlackListStatus(address)._maker (contracts/contracts/token/
Me3Token.sol#459) is not in mixedCase
Parameter Me3Token.addBlackList(address)._evilUser (contracts/contracts/token/
Me3Token.sol#463) is not in mixedCase
Parameter Me3Token.removeBlackList(address)._clearedUser (contracts/contracts/token/
Me3Token.sol#468) is not in mixedCase
Parameter Me3Token.destroyBlackFunds(address)._blackListedUser (contracts/contracts/
token/Me3Token.sol#473) is not in mixedCase
Parameter Me3Token.mint(address,uint256)._to (contracts/contracts/token/
Me3Token.sol#509) is not in mixedCase
Parameter Me3Token.mint(address,uint256)._amount (contracts/contracts/token/
Me3Token.sol#509) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-
solidity-naming-conventions
INFO:Detectors:
Variable Me3Token._totalSupply (contracts/contracts/token/Me3Token.sol#55) is too
similar to Me3Token.constructor(string,string,uint256).totalSupply_ (contracts/
contracts/token/Me3Token.sol#73)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-
too-similar
```

Slither ERC conformance

```
# Check Me3Token

## Check functions
[] totalSupply() is present
        [] totalSupply() -> (uint256) (correct return type)
        [] totalSupply() is view
[] balanceOf(address) is present
        [] balanceOf(address) -> (uint256) (correct return type)
        [] balanceOf(address) is view
[] transfer(address,uint256) is present
        [] transfer(address,uint256) -> (bool) (correct return type)
        [] Transfer(address,address,uint256) is emitted
[] transferFrom(address,address,uint256) is present
        [] transferFrom(address,address,uint256) -> (bool) (correct return type)
        [] Transfer(address,address,uint256) is emitted
[] approve(address,uint256) is present
        [] approve(address,uint256) -> (bool) (correct return type)
        [] Approval(address,address,uint256) is emitted
[] allowance(address,address) is present
        [] allowance(address,address) -> (uint256) (correct return type)
        [] allowance(address,address) is view
[] name() is present
        [] name() -> (string) (correct return type)
        [] name() is view
[] symbol() is present
        [] symbol() -> (string) (correct return type)
        [] symbol() is view
[] decimals() is present
        [] decimals() -> (uint8) (correct return type)
        [] decimals() is view

## Check events
[] Transfer(address,address,uint256) is present
        [] parameter 0 is indexed
        [] parameter 1 is indexed
[] Approval(address,address,uint256) is present
        [] parameter 0 is indexed
```

```
        [☒] parameter 1 is indexed


        [ ] Me3Token is not protected for the ERC20 approval race condition
```

0x Guard