



Smart contracts security assessment

Final report

[Tariff: Top](#)

pTON.fi

March 2023



0xguard.com



hello@0xguard.com

Contents

1. Introduction	3
2. Contracts checked	3
3. Procedure	3
4. Known vulnerabilities checked	4
5. Classification of issue severity	5
6. Issues	5
7. Conclusion	8
8. Disclaimer	9
9. Static analysis result	10

Introduction

The report has been prepared for **pTON.fi**.

The code is available at the [pton-fi/pton-smart-contracts](https://github.com/pton-fi/pton-smart-contracts) GitHub repository and was audited after the commit [caf1e27](#).

Update: the updated code was audited after the commit [2bdec30](#).

Name	pTON.fi
Audit date	2023-03-22 - 2023-03-30
Language	Solidity
Platform	Ethereum

Contracts checked

Name	Address
PooledTON	
StakedTON	

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
<u>Unencrypted Private Data On-Chain</u>	passed
<u>Code With No Effects</u>	passed
<u>Message call with hardcoded gas amount</u>	passed
<u>Typographical Error</u>	passed
<u>DoS With Block Gas Limit</u>	passed
<u>Presence of unused variables</u>	passed
<u>Incorrect Inheritance Order</u>	passed
<u>Requirement Violation</u>	passed
<u>Weak Sources of Randomness from Chain Attributes</u>	passed
<u>Shadowing State Variables</u>	passed
<u>Incorrect Constructor Name</u>	passed
<u>Block values as a proxy for time</u>	passed
<u>Authorization through tx.origin</u>	passed
<u>DoS with Failed Call</u>	passed
<u>Delegatecall to Untrusted Callee</u>	passed
<u>Use of Deprecated Solidity Functions</u>	passed
<u>Assert Violation</u>	passed
<u>State Variable Default Visibility</u>	passed
<u>Reentrancy</u>	passed

<u>Unprotected SELFDESTRUCT Instruction</u>	passed
<u>Unprotected Ether Withdrawal</u>	passed
<u>Unchecked Call Return Value</u>	passed
<u>Floating Pragma</u>	passed
<u>Outdated Compiler Version</u>	passed
<u>Integer Overflow and Underflow</u>	passed
<u>Function Default Visibility</u>	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. No sufficient mint restriction mechanism (StakedTON)

Status: Fixed

The token has a mint function to create new tokens when TONs are bridged. This function can be called by an addresses with a VALIDATOR role which imposes additional risks if a private key of address with such role is compromised.

```
function mint(address to, uint256 amountUnderlying)
    external
    whenNotPaused
    onlyRole(VALIDATOR_ROLE)
{
    if (to == address(0)) revert ZeroAddress();
    uint256 shares = _underlyingToShares(amountUnderlying);
    if (shares == 0) revert ZeroShares();

    _mint(to, amountUnderlying, shares);
    _updateUnderlying(amountUnderlying.toInt256());
}
```

Recommendation: Implement several validator roles to for minting. Allow to mint only if 2/3 validators have signed the mint message.

Medium severity issues

1. No constraints on rewards (StakedTON)

Status: Fixed

The contract has an `updateRewards()` function aimed to distribute a reward passed in the function parameters amongst users during a specified time. This function can be called with an arbitrary big value which may inappropriately increase user's balances.

Recommendation: Add a limit for the reward distribution.

Low severity issues

1. External protocol support for rebase a rebase token (StakedTON)

Status: Open

The reward distribution increases account's token balance without any interactions. If an account holds tokens and rewards are distributed, the account's stTON balance with increase. Some protocols may not support such mechanics.

Recommendation: These mechanics are by the token design, but it is important to check if a protocol supports tokens which balances may change before using them with stTON.

Conclusion

pTON.fi PooledTON, StakedTON contracts were audited. 1 high, 1 medium, 1 low severity issues were found.

1 high, 1 medium severity issues have been fixed in the update.

The stTON token is dependent on the owner's account. The contract is deployed via proxy and can be upgraded by the owner.

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 OxGuard 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 OxGuard 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.

OxGuard retains exclusive publishing rights for the results of this audit on its website and social networks.

Static analysis result

StakedTON._burn(address,uint256,bytes) (contracts/StakedTON.sol#171-182) uses a dangerous strict equality:

☒- shares == 0 (contracts/StakedTON.sol#177)

StakedTON._transfer(address,address,uint256) (contracts/StakedTON.sol#227-236) uses a dangerous strict equality:

☒- shares == 0 (contracts/StakedTON.sol#233)

StakedTON._underlyingToShares(uint256) (contracts/StakedTON.sol#219-225) uses a dangerous strict equality:

☒- currentSupply == 0 (contracts/StakedTON.sol#221-224)

StakedTON._updateUnderlying(int256) (contracts/StakedTON.sol#238-243) uses a dangerous strict equality:

☒- supplyDelta == 0 (contracts/StakedTON.sol#239)

StakedTON.mint(address,uint256) (contracts/StakedTON.sol#131-142) uses a dangerous strict equality:

☒- shares == 0 (contracts/StakedTON.sol#138)

StakedTON.mintWrapped(address,uint256) (contracts/StakedTON.sol#144-159) uses a dangerous strict equality:

☒- shares == 0 (contracts/StakedTON.sol#151)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities>

INFO:Detectors:

StakedTON._updateRewards(int256,uint64).rewardRemainder (contracts/StakedTON.sol#258) is a local variable never initialized

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables>

INFO:Detectors:

StakedTON.mintWrapped(address,uint256) (contracts/StakedTON.sol#144-159) ignores return value by IERC4626(wrapper).deposit(amountUnderlying,to) (contracts/StakedTON.sol#158)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return>

INFO:Detectors:

Reentrancy in StakedTON.burnWrapped(uint256,bytes) (contracts/StakedTON.sol#94-102):

☒External calls:

☒- amountUnderlying = IERC4626(pTON).redeem(amountWrapped,address(this),_msgSender()) (contracts/StakedTON.sol#96-100)

☒State variables written after the call(s):

☒- _burn(address(this),amountUnderlying,data) (contracts/StakedTON.sol#101)

☒☒- _balances[account] = accountBalance - amount (contracts/erc20/

ERC20Upgradeable.sol#315)

```

❏- _burn(address(this),amountUnderlying,data) (contracts/StakedTON.sol#101)
❏❏- _totalSupply -= amount (contracts/erc20/ERC20Upgradeable.sol#317)
❏- _burn(address(this),amountUnderlying,data) (contracts/StakedTON.sol#101)
❏❏- totalUnderlying = newSupply.toUint256() (contracts/StakedTON.sol#242)
Reentrancy in StakedTON.burnWrappedPermit(uint256,bytes) (contracts/
StakedTON.sol#104-129):
❏External calls:
❏- ERC20PermitUpgradeable(wrapper).permit(_msgSender(),address(this),amountWrapped,deadl
ine,v,r,s) (contracts/StakedTON.sol#114-122)
❏- amountUnderlying =
IERC4626(wrapper).redeem(amountWrapped,address(this),_msgSender()) (contracts/
StakedTON.sol#123-127)
❏State variables written after the call(s):
❏- _burn(address(this),amountUnderlying,data) (contracts/StakedTON.sol#128)
❏❏- _balances[account] = accountBalance - amount (contracts/erc20/
ERC20Upgradeable.sol#315)
❏- _burn(address(this),amountUnderlying,data) (contracts/StakedTON.sol#128)
❏❏- _totalSupply -= amount (contracts/erc20/ERC20Upgradeable.sol#317)
❏- _burn(address(this),amountUnderlying,data) (contracts/StakedTON.sol#128)
❏❏- totalUnderlying = newSupply.toUint256() (contracts/StakedTON.sol#242)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-2
INFO:Detectors:
Reentrancy in StakedTON.burnWrapped(uint256,bytes) (contracts/StakedTON.sol#94-102):
❏External calls:
❏- amountUnderlying = IERC4626(pTON).redeem(amountWrapped,address(this),_msgSender())
(contracts/StakedTON.sol#96-100)
❏Event emitted after the call(s):
❏- Burned(from,amountUnderlying,data) (contracts/StakedTON.sol#180)
❏❏- _burn(address(this),amountUnderlying,data) (contracts/StakedTON.sol#101)
❏- Transfer(from,address(0),amountUnderlying) (contracts/StakedTON.sol#181)
❏❏- _burn(address(this),amountUnderlying,data) (contracts/StakedTON.sol#101)
Reentrancy in StakedTON.burnWrappedPermit(uint256,bytes) (contracts/
StakedTON.sol#104-129):
❏External calls:
❏- ERC20PermitUpgradeable(wrapper).permit(_msgSender(),address(this),amountWrapped,deadl
ine,v,r,s) (contracts/StakedTON.sol#114-122)
❏- amountUnderlying =
IERC4626(wrapper).redeem(amountWrapped,address(this),_msgSender()) (contracts/
StakedTON.sol#123-127)
❏Event emitted after the call(s):

```

```

❑- Burned(from,amountUnderlying,data) (contracts/StakedTON.sol#180)
❑❑- _burn(address(this),amountUnderlying,data) (contracts/StakedTON.sol#128)
❑- Transfer(from,address(0),amountUnderlying) (contracts/StakedTON.sol#181)
❑❑- _burn(address(this),amountUnderlying,data) (contracts/StakedTON.sol#128)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
StakedTON.mint(address,uint256) (contracts/StakedTON.sol#131-142) uses timestamp for comparisons
❑Dangerous comparisons:
❑- shares == 0 (contracts/StakedTON.sol#138)
StakedTON.mintWrapped(address,uint256) (contracts/StakedTON.sol#144-159) uses timestamp for comparisons
❑Dangerous comparisons:
❑- shares == 0 (contracts/StakedTON.sol#151)
StakedTON._burn(address,uint256,bytes) (contracts/StakedTON.sol#171-182) uses timestamp for comparisons
❑Dangerous comparisons:
❑- shares == 0 (contracts/StakedTON.sol#177)
StakedTON._pendingRewards() (contracts/StakedTON.sol#193-209) uses timestamp for comparisons
❑Dangerous comparisons:
❑- timestamp < start (contracts/StakedTON.sol#200)
❑- timestamp > end (contracts/StakedTON.sol#204)
StakedTON._underlyingToShares(uint256) (contracts/StakedTON.sol#219-225) uses timestamp for comparisons
❑Dangerous comparisons:
❑- currentSupply == 0 (contracts/StakedTON.sol#221-224)
StakedTON._transfer(address,address,uint256) (contracts/StakedTON.sol#227-236) uses timestamp for comparisons
❑Dangerous comparisons:
❑- shares == 0 (contracts/StakedTON.sol#233)
StakedTON._updateUnderlying(int256) (contracts/StakedTON.sol#238-243) uses timestamp for comparisons
❑Dangerous comparisons:
❑- supplyDelta == 0 (contracts/StakedTON.sol#239)
StakedTON._updateRewards(int256,uint64) (contracts/StakedTON.sol#245-273) uses timestamp for comparisons
❑Dangerous comparisons:
❑- timestamp < start (contracts/StakedTON.sol#260)
❑- timestamp < end (contracts/StakedTON.sol#262)

```

ERC20PermitUpgradeable.permit(address,address,uint256,uint256,uint8,bytes32,bytes32)
(contracts/erc20/ERC20PermitUpgradeable.sol#64-85) uses timestamp for comparisons

⚠Dangerous comparisons:

⚠- require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline)
(contracts/erc20/ERC20PermitUpgradeable.sol#73)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp>

INFO:Detectors:

ERC20PermitUpgradeable.__ERC20Permit_init_unchained(string) (contracts/erc20/ERC20PermitUpgradeable.sol#59) is never used and should be removed

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code>

INFO:Detectors:

Pragma version0.8.17 (contracts/PooledTON.sol#7) allows old versions

Pragma version0.8.17 (contracts/StakedTON.sol#3) allows old versions

Pragma version^0.8.0 (contracts/erc20/ERC20PermitUpgradeable.sol#5) allows old versions

Pragma version^0.8.0 (contracts/erc20/ERC20Upgradeable.sol#6) allows old versions

Pragma version^0.8.4 (contracts/interfaces/ISTakedTON.sol#5) allows old versions

solc-0.8.17 is not recommended for deployment

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity>

INFO:Detectors:

Function ERC20PermitUpgradeable.__ERC20Permit_init(string) (contracts/erc20/ERC20PermitUpgradeable.sol#55-57) is not in mixedCase

Function ERC20PermitUpgradeable.__ERC20Permit_init_unchained(string) (contracts/erc20/ERC20PermitUpgradeable.sol#59) is not in mixedCase

Function ERC20PermitUpgradeable.DOMAIN_SEPARATOR() (contracts/erc20/ERC20PermitUpgradeable.sol#98-100) is not in mixedCase

Variable ERC20PermitUpgradeable._PERMIT_TYPEHASH_DEPRECATED_SLOT (contracts/erc20/ERC20PermitUpgradeable.sol#48) is not in mixedCase

Variable ERC20PermitUpgradeable.__gap (contracts/erc20/ERC20PermitUpgradeable.sol#118) is not in mixedCase

Function ERC20Upgradeable.__ERC20_init(string,string) (contracts/erc20/ERC20Upgradeable.sol#62-64) is not in mixedCase

Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (contracts/erc20/ERC20Upgradeable.sol#66-72) is not in mixedCase

Variable ERC20Upgradeable.__gap (contracts/erc20/ERC20Upgradeable.sol#415) is not in mixedCase

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions>

INFO:Slither:. analyzed (56 contracts with 85 detectors), 36 result(s) found



 Guard