DeHacker

Code Security Assessment

PTON

August 22th, 2024

Contents

Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best prac tices. Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow/underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service/logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Issue Categories

Every issue in this report was assigned a severity level from the following:

Critical severity issues

A vulnerability that can disrupt the contract functioning in a number of scenarios or creates a risk that the contract may be broken.

Major severity issues

A vulnerability that affects the desired outcome when using a contract or provides the opportunity to use a contract in an unintended way.

Medium severity issues

A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

Minor severity issues

A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

Informational

A vulnerability that has informational character but is not affecting any of the code.

Overview

Project Summary

Project Name	PTON
Platform	Other
Website	pton.fi/
Type	DeFi
Language	Solidity
Codebase	https://github.com/pton-fi/pton-smart-contracts/

Vulnerability Summary

Vulnerability Level	Total	Mitigated	Declined	Acknowledged	Partially Resolved	Resolved
Critical	1	0	0	0	0	1
Major	0	0	0	0	0	0
Medium	1	0	0	0	0	1
Minor	4	0	0	2	1	1
Informational	2	0	0	2	0	0
Discussion	0	0	0	0	0	0

Audit scope

ID	File	SHA256 Checksum
ERC	Pton-fi/pton-smart-contracts	6e4c7e65b061743984798ab1874f7c8a1f2654bc7bdf15e95aed5e2612ac5c31

Findings

ID	Title	Severity	Status
STO-01	No Update For totalUnderlying	Critical	Resolved
CON-01	Lack Of Storage Gap In Upgredeable Contract	Medium	Resolved
ERC-01	Inconsistency Between The maxFlashLoan() And flashLoan() Functions	Minor	Resolved
PTO-01	Possible Inflation Attack Arising From Inherited ERC4626 Implementation	Minor	Acknowledged
STO-03	Potential Precision Loss In The balance of() Function	Minor	Acknowledged
STO-05	Potential Rounding Error In The _underlyingToShares() Function	Minor	Partially Resolved
ERP-02	Unnecessary Inheritance Of ERC20FlashMintUpgreadeable Contract	Informational	Acknowledged
STO-04	Total Reward Delta Can Be Negative And Greater Than totalUnderlying	Informational	Acknowledged

CRITICAL

STO-01 | No Update For totalUnderlying

Issue	Severity	Location	Status
Logical Issue	Critical	Contracts/stTon.sol (d899e939): 196~197	Resolved

Description

The flashLoan() function allows users to borrow tokens and perform actions with those tokens within a single transaction. Before executing the flash loan, the function validates the requested amount of tokens to be borrowed by converting it to the corresponding amount of shares using the _validateShares() function. Subsequently, the inherited flashLoan functionmints these shares and burns them at the end of the transaction.

```
190 function flashLoan(
191       IERC3156FlashBorrowerUpgradeable receiver,
192       address token,
193       uint256 amountUnderlying,
194       bytes calldata data
195 ) public virtual override onlyRole(LIQUIDATOR_ROLE) returns (bool) {
196       uint256 shares = _validateShares(amountUnderlying);
197       return super.flashLoan(receiver, token, shares, data);
198 }
```

However, the flashLoan() function doesn't update the totalUnderlying prior to executing a flash loan, which may causethe tokens to deflate. Consequently, an attacker may mint tokens during flash loan operations to obtain more tokens due tothe calculation in the _underlyingToShares() function. The amountUnderlying and _totalUnderlying variables arefixed, with only the currentSupply variable being subject to a significant increase.

Description

```
function _underlyingToShares(uint256 amountUnderlying) internal view returns (uint256) {
    uint256 currentSupply = ERC20Upgradeable.totalSupply();
    return
        currentSupply == 0
            ? amountUnderlying
            : (amountUnderlying * currentSupply) / _totalUnderlying();
}
```

Recommendation

The audit team recommends updating the totalUnderlying before performing the flash loan operations.

MEDIUM

CON-01 | Lack Of Storage Gap In Upgradeable Contract

Issue	Severity	Location	Status
Logical Issue	Medium	Contracts/erc20/ERC20FlashMintUpgradeable.sol (d899e939): 21; contracts/stTON.sol (d899e939): 15	Resolved

Description

There is no storage gap preserved in the logic contract. Any logic contract that acts as a base contract that needs to beinherited by other upgradeable child should have a reasonable size of storage gap preserved for the new state variableintroduced by the future upgrades.

Recommendation

We recommend having a storage gap of a reasonable size preserved in the logic contract in case that new state variablesare introduced in future upgrades. For more information, please refer to:
https://docs.openzeppelin.com/contracts/3.x/upgradeable#storage_gaps

MINOR

ERC-01 | Inconsistency Between The maxFlashLoan() AND flashLoan() FUNCTIONS

Issue	Severity	Location	Status
Logical Issue	Minor	Contracts/erc20/ERC20FlashMintUpgradeable.sol (d899e939): 30	Resolved

Description

The flashloan() function is utilized to flash mint shares , instead of the amount of underlying TON tokens. However, the maxFlashLoan() function checks the maximum amount of tokens available for the loan based on the totalSupply() function, which returns the amount of total underlying TON tokens.

```
29 function maxFlashLoan(address token) public view virtual override returns (uint256) {
30     return token == address(this) ? type(uint256).max - totalSupply() : 0;
31 }
```

Recommendation

In the flashloan() function, the amount of tokens to be minted is in the form of "shares" within the contract, which shouldbe capped by ERC20Upgradeable.totalSupply() , instead of totalSupply() that represent all the underlying tokenamount. Therefore, we recommend using the ERC20Upgradeable.totalSupply() function instead.

MINOR

PTO-01 | Possible Inflation Attack Arising From Inherited ERC4626 Implementation

Issue	Severity	Location	Status
Mathematical Operations	Minor	Contracts/pTON.sol (d899e939): 19	Acknowledged

Description

Due to the inheritance of the ERC4626 implementation, the GLP vault contract is vulnerable to an inflation attack. Maliciousactors can exploit this vulnerability to steal initial deposits made into the pools, leading to substantial losses for unsuspecting investors.

This issue is also highlighted in the ERC4626 contract:

```
* CAUTION: When the vault is empty or nearly empty, deposits are at high risk ofbei
stolen through frontrunning with
* a "donation" to the vault that inflates the price of a share. This is variouslyknown a
donation or inflation
* attack and is essentially a problem of slippage. Vault deployers can protectagainst
attack by making an initial
* deposit of a non-trivial amount of the asset, such that price manipulationbecomes
infeasible. Withdrawals may
* similarly be affected by slippage. Users can protect against this attack as
wellunexpected slippage in general by
* verifying the amount received is as expected, using a wrapper that performs
thesechecks such as
* https://github.com/fei-protocol/ERC4626#erc4626router-and-base[ERC4626Router
```

Recommendation

Multiple methods can be adopted according to this pull request. One proposed solution is to deposit some assets duringinitialization.

MINOR

STO-03 | Potential Precision Loss In The balanceOf() Function

Issue	Severity	Location	Status
Mathematical Operations	Minor	Contracts/stTON.sol (d899e939): 79~81	Acknowledged

Description

The balanceOf function in the given contract calculates the balance of a specific account by multiplying its ERC20 tokenbalance by the total underlying balance and dividing it by the total ERC20 token supply.

```
78 function balanceOf(address account) public view override returns (uint256)
{
79 return
80 (ERC20Upgradeable.balanceOf(account) * _totalUnderlying()) /
81 ERC20Upgradeable.totalSupply();
82 }
```

While this function can approximate the actual assets that belong to an account, there is a possibility of truncation errors dueto integer division. This can lead to small discrepancies between the calculated balance and the actual balance that shouldbelong to an account. As a consequence of the precision loss that may occur with the balanceOf() function, the transfer() function may not function as expected, causing stTON unsuitable for certain Defi protocols. As stated in thewhitepaper of pTON, users are advised to use pTON instead of other Defi protocols.

Recommendation

The audit team recommends that clients address this truncation error by adopting the design of OpenZeppelin's ERC4626 and utilizing the mulDiv function from the math library. Furthermore, since the underlying logic of stTON is similar to that of ERC4626 , the audit team suggests directly implementing stTON by utilizing the ERC4626 .

MINOR

STO-05 | Potential Rounding Error In The _underlyingToShares() Function

Issue	Severity	Location	Status
Mathematical Operations	Minor	Contracts/stTON.sol (d899e939): 245	Partially Resolved

Description

There could be a rounding error in the _underlyingToShares() function. When the amountUnderlying is small, it mayresult in the value of the shares being calculated as zero, which could cause users to be unable to mint tokens.

```
240 function _underlyingToShares(uint256 amountUnderlying) internal view returns (uint256) {
241 uint256 currentSupply = ERC20Upgradeable.totalSupply();
242 return
243 currentSupply == 0
244 ? amountUnderlying
245 : (amountUnderlying * currentSupply) / _totalUnderlying();
246
```

Recommendation

The audit team would like to discuss with the team if the distribution of rewards will significantly affect the ratio of currentSupply and _totalUnderlying() .

INFORMATIONAL

ERP-02 | Unnecessary Inheritance Of ERC20FlashMintUpgradeable Contract

Issue	Severity	Location	Status
Coding Style	Informational	Contracts/erc20/ERC20PermitUpgradeable.sol (d899e939): 28	Aknowledged

Description

To inherit the ERC20FlashMintUpgradeable and ERC20PermitUpgradeable extensions, there is no need to force the ERC20PermitUpgradeable contract to inherit ERC20FlashMintUpgradeable . The stTON contract can simultaneously inherit the ERC20PermitUpgradeable and ERC20FlashMintUpgradeable contracts.

```
contract stTON is
    ERC20PermitUpgradeable,
    ERC20FlashMintUpgradeable,
    UUPSUpgradeable,
    AccessControlEnumerableUpgradeable,
    PausableUpgradeable,
    Multicall,
    IstTON
{
    ...
}
```

Recommendation

The audit team recommends removing the unnecessary inheritance of ERC20FlashMintUpgradeable contract.

INFORMATIONAL

STO-04 | Total Reward Delta Can Be Negative And Greater Than totalUnderlying

Issue	Severity	Location	Status
Logical Issue	Informational	contracts/stTON.sol (d899e939): 287	Aknowledged

Description

In the given contract, there is a possibility that the rewardsDelta could be negative and even greater than the original totalUnderlying , which would cause the user to lose their staked underlying TON tokens. This issue can be observed inthe _updateRewards() function, where the rewardsDelta is used to calculate the new reward rate. If the rewardsDelta is negative and large enough, it could cause the totalUnderlying to be decreased a lot as well when calling the _updateUnderlying() function with _pendingRewards() .

Recommendation

The audit prepared a POC below to show users may not transfer or withdraw their staked TON tokens. The audit team wouldlike to confirm with clients if it is the intended design and whether there is any mechanism to ensure that users can always withdraw their staked TON tokens.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Coding Style

Coding Style findings usually do not affect the generated bytecode but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block. timestamp works.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux “sha256sum" command against the target file.

About

DeHacker is a team of auditors and white hat hackers who perform security audits and assessments. With decades of experience in security and distributed systems, our experts focus on the ins and outs of system security. Our services follow clear and prudent industry standards. Whether it's reviewing the smallest modifications or a new platform, we'll provide an in-depth security survey at every stage of your company's project. We provide comprehensive vulnerability reports and identify structural inefficiencies in smart contract code, combining high-end security research with a real-world attacker mindset to reduce risk and harden code.

BLOCKCHAIINS

Ethereum
Cosmos
Eos
Substrate

TECH STACK

Python
Solidity
Rust
c++

CONTACTS

https://dehacker.io

https://twitter.com/dehackerio

https://github.com/dehacker/audits_public

https://t.me/dehackerio

https://blog.dehacker.io/

DeHacker

August 22th 2024