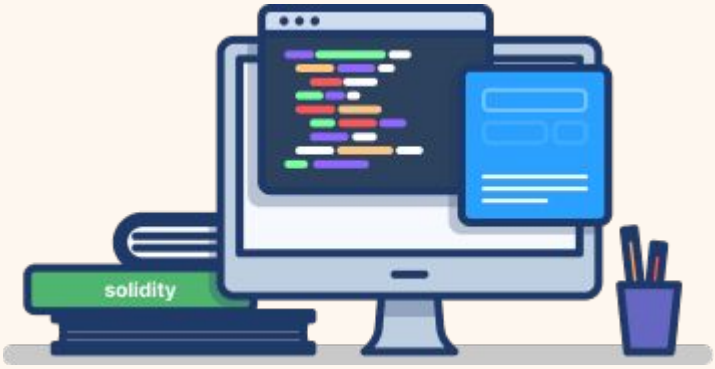
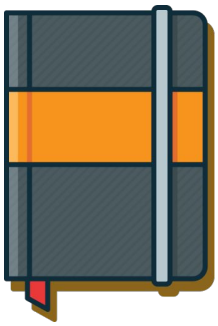
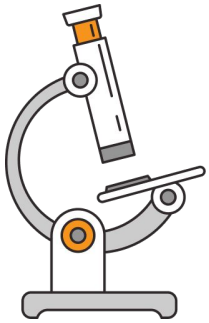
Coinbae Audit

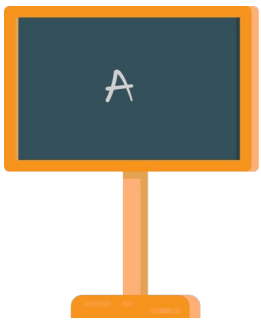


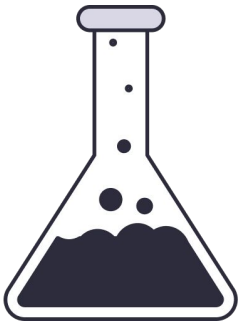
TapcoinToken Contract Audit June 2021

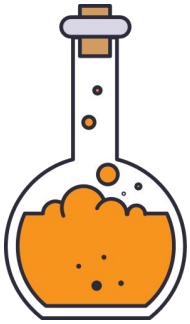
Contents

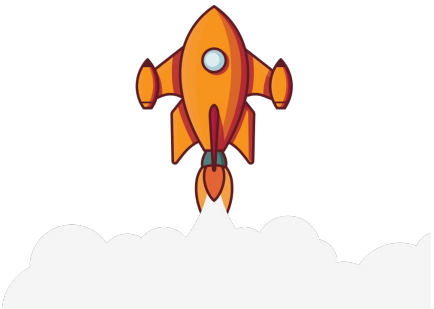












Introduction

Audit:

In June 2021 Coinbae's audit report division performed an audit for the Tapcoins Token Contract.
https://etherscan.io/address/0x9F599410D207f3D2828a8712e5e543AC2E040382

Tapcoins:

Tapcoins ('TTT') are digital tokens built on the Ethereum Blockchain. They are the foundation for blockchain gaming on the Tap Platform. Tapcoins allow gamers to transfer their in-game currency from one game to the next – making it possible to get "paid to play!"

Tapcoins can be used on the Tap Platform to upgrade Tapkeys. Tapkeys provide access to Steam PC Gaming, and the Great NFT Hunt. Using Tapkeys in the Great NFT Hunt unlocks NFT Collectible Card Packs!

Tapcoins are the native token on the Tap Platform and are rewards when farming Uniswap V2 tokens and staking for platform revenue

Introduction

Overview:

Information:
Name: Tapcoins

Pool, Asset or Contract address:
https://etherscan.io/address/0x9F599410D207f3D2828a8712e5e543AC2E040382

Supply:
Current: 1,500,000,000 TTT

Explorers:
https://etherscan.io/token/0x9f599410d207f3d2828a8712e5e543ac2e040382?a=0x9F599410D207f3D2828a8712e5e543AC2E040382

Websites:
https://www.tapproject.net/

Links:
https://www.tapproject.net/

Introduction

Compiler related issues:

It is best practice to use the latest version of the solidity compiler supported by the toolset you use. This so it includes all the latest bug fixes of the solidity compiler. When you use for instance the openzeppelin contracts in your code the solidity version you should use should be 0.8.0 because this is the latest version supported.

Caution:

The solidity versions used for the audited contracts can be 0.6.0 --> 0.8.0 these versions have for instance the following known bugs so the compiled contract might be susceptible to:

EmptyByteArrayCopy – Medium risk

Copying an empty byte array (or string) from memory or calldata to storage can result in data corruption if the target array's length is increased subsequently without storing new data.
https://etherscan.io/solcbuginfo?a=EmptyByteArrayCopy

DynamicArrayCleanup – Medium risk

When assigning a dynamically-sized array with types of size at most 16 bytes in storage causing the assigned array to shrink, some parts of deleted slots were not zeroed out.
https://etherscan.io/solcbuginfo?a=DynamicArrayCleanup

Advice:

Update the contracts to the latest supported version of solidity by your contract. And set it as a fixed parameter not a floating pragma.

Audit Report Scope

Assertions and Property Checking:

1. Solidity assert violation.
2. Solidity AssertionFailed event.

ERC Standards:

1. Incorrect ERC20 implementation.

Solidity Coding Best Practices:

1. Outdated compiler version.
2. No or floating compiler version set.
3. Use of right-to-left-override control character.
4. Shadowing of built-in symbol.
5. Incorrect constructor name.
6. State variable shadows another state variable.
7. Local variable shadows a state variable.
8. Function parameter shadows a state variable.
9. Named return value shadows a state variable.
10. Unary operation without effect Solidity code analysis.
11. Unary operation directly after assignment.
12. Unused state variable.
13. Unused local variable.
14. Function visibility is not set.
15. State variable visibility is not set.
16. Use of deprecated functions: call code(), sha3(), ...
17. Use of deprecated global variables (msg.gas, ...).
18. Use of deprecated keywords (throw, var).
19. Incorrect function state mutability.
20. Does the code conform to the Solidity styleguide.

Convert code to conform Solidity styleguide:

1. Convert all code so that it is structured accordingly the Solidity styleguide.

Audit Report Scope

Categories:

High Severity:

High severity issues opens the contract up for exploitation from malicious actors. We do not recommend deploying contracts with high severity issues.

Medium Severity Issues:

Medium severity issues are errors found in contracts that hampers the effectiveness of the contract and may cause outcomes when interacting with the contract. It is still recommended to fix these issues.

Low Severity Issues:

Low severity issues are warning of minor impact on the overall integrity of the contract. These can be fixed with less urgency.

Audit Report

6	6	0
Identified	Confirmed	Critical

0	0	6
High	Medium	Low

Optimization issues identified: 18

Analysis:
https://etherscan.io/address/0x9F599410D207f3D2828a8712e5e543AC2E040382

Risk:
LOW

Low issues identified:

Shadowing Builtin:

SafeMath.assert(bool) (TapcoinToken.sol#47-51) (function) shadows built-in symbol"

Events Access:

ReleasableToken.setReleaseAgent(address) (TapcoinToken.sol#267-271) should emit an event for:
- releaseAgent = addr (TapcoinToken.sol#270)

Missing Zero Check:

UpgradeableToken.UpgradeableToken(address)._upgradeMaster (TapcoinToken.sol#335) lacks a zero-check on :
- upgradeMaster = _upgradeMaster (TapcoinToken.sol#336)

ReleasableToken.setReleaseAgent(address).addr (TapcoinToken.sol#267) lacks a zero-check on :
- releaseAgent = addr (TapcoinToken.sol#270)

Reentrancy Events:

Reentrancy in UpgradeableToken.setUpgradeAgent(address) (TapcoinToken.sol#367-388):
External calls:
- ! upgradeAgent.isUpgradeAgent() (TapcoinToken.sol#383)
Event emitted after the call(s):
- UpgradeAgentSet(upgradeAgent) (TapcoinToken.sol#387)

Low issues identified:

Reentrancy Events:

Reentrancy in UpgradeableToken.setUpgradeAgent(address)
(TapcoinToken.sol#367-388):
External calls:
- ! upgradeAgent.isUpgradeAgent() (TapcoinToken.sol#383)
Event emitted after the call(s):
- UpgradeAgentSet(upgradeAgent) (TapcoinToken.sol#387)

Reentrancy in UpgradeableToken.upgrade(uint256) (TapcoinToken.sol#342-362):
External calls:
- upgradeAgent.upgradeFrom(msg.sender,value) (TapcoinToken.sol#360)
Event emitted after the call(s):
- Upgrade(msg.sender,upgradeAgent,value) (TapcoinToken.sol#361)

Optimization issues identified:

Constable States:

FractionalERC20.decimals (TapcoinToken.sol#103) should be constant

UpgradeAgent.originalSupply (TapcoinToken.sol#502) should be constant

External Function:

transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (TapcoinToken.sol#74-78)

balanceOf(address) should be declared external:
- ERC20.balanceOf(address) (TapcoinToken.sol#88)
- StandardToken.balanceOf(address) (TapcoinToken.sol#146-148)

allowance(address,address) should be declared external:
- ERC20.allowance(address,address) (TapcoinToken.sol#89)
- StandardToken.allowance(address,address) (TapcoinToken.sol#163-165)

approve(address,uint256) should be declared external:
- ERC20.approve(address,uint256) (TapcoinToken.sol#93)
- StandardToken.approve(address,uint256) (TapcoinToken.sol#150-161)

isToken() should be declared external:
- StandardToken.isToken() (TapcoinToken.sol#125-127)

increaseApproval(address,uint256) should be declared external:
- StandardToken.increaseApproval(address,uint256) (TapcoinToken.sol#167-172)

decreaseApproval(address,uint256) should be declared external:
- StandardToken.decreaseApproval(address,uint256) (TapcoinToken.sol#174-184)

Optimization issues identified:

External Function:

mint(address,uint256) should be declared external:
- MintableToken.mint(address,uint256) (TapcoinToken.sol#205-214)

setMintAgent(address,bool) should be declared external:
- MintableToken.setMintAgent(address,bool) (TapcoinToken.sol#219-222)

setReleaseAgent(address) should be declared external:
- ReleasableToken.setReleaseAgent(address) (TapcoinToken.sol#267-271)

setTransferAgent(address,bool) should be declared external:
- ReleasableToken.setTransferAgent(address,bool) (TapcoinToken.sol#273-275)

upgrade(uint256) should be declared external:
- UpgradeableToken.upgrade(uint256) (TapcoinToken.sol#342-362)

setUpgradeMaster(address) should be declared external:
- UpgradeableToken.setUpgradeMaster(address) (TapcoinToken.sol#405-409)

setTokenInformation(string,string) should be declared external:
- CrowdsaleToken.setTokenInformation(string,string) (TapcoinToken.sol#485-490)

isUpgradeAgent() should be declared external:
- UpgradeAgent.isUpgradeAgent() (TapcoinToken.sol#505-507)

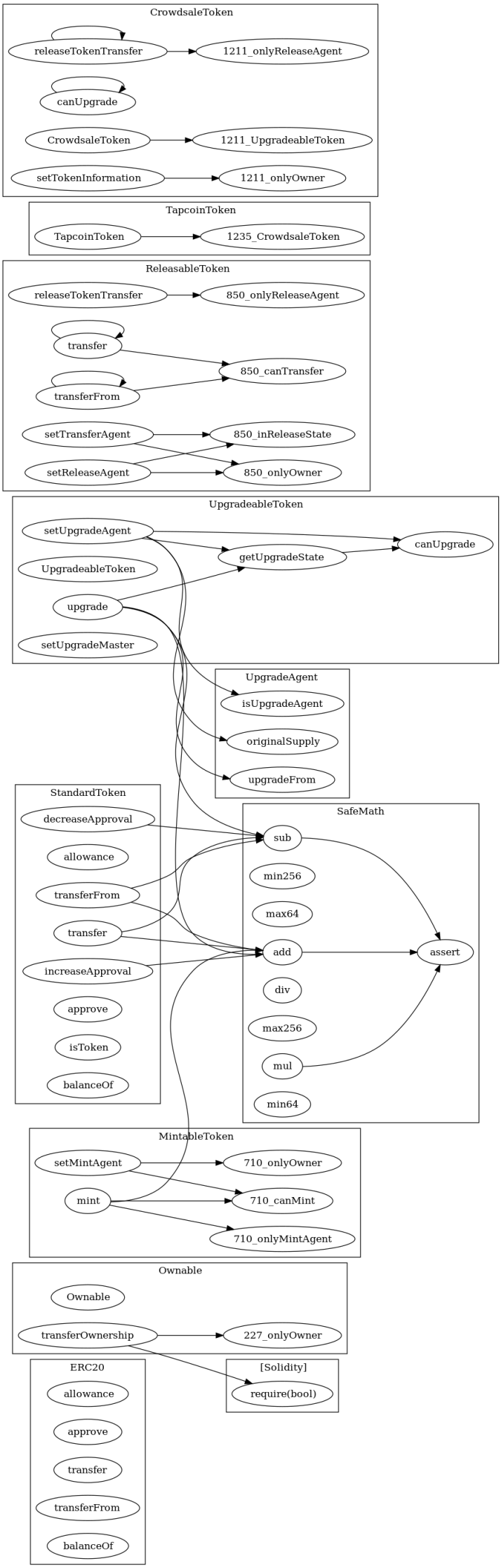
upgradeFrom(address,uint256) should be declared external:
- UpgradeAgent.upgradeFrom(address,uint256) (TapcoinToken.sol#509)

Conclusion

We performed the procedures as laid out in the scope of the audit and there were 6 findings, 0 high, 0 medium and 6 low. The overall risk level is **Low.**

Disclaimer

Coinbae audit is not a security warranty, investment advice, or an endorsement of the TapcoinToken contract. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the the TapcoinToken Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.


CrowdsaleToken
releaseTokenTransfer
1211_onlyReleaseAgent
canUpgrade
CrowdsaleToken
1211_UpgradeableToken
setTokenInformation
1211_onlyOwner
TapcoinToken
TapcoinToken
1235_CrowdsaleToken
ReleasableToken
releaseTokenTransfer
850_onlyReleaseAgent
transfer
850_canTransfer
transferFrom
setTransferAgent
850_inReleaseState
setReleaseAgent
850_onlyOwner
UpgradeableToken
setUpgradeAgent
canUpgrade
getUpgradeState
UpgradeableToken
upgrade
setUpgradeMaster
UpgradeAgent
isUpgradeAgent
originalSupply
upgradeFrom
StandardToken
decreaseApproval
allowance
transferFrom
transfer
increaseApproval
approve
isToken
balanceOf
SafeMath
sub
min256
max64
add
assert
div
max256
mul
min64
MintableToken
setMintAgent
710_onlyOwner
mint
710_canMint
710_onlyMintAgent
Ownable
Ownable
transferOwnership
227_onlyOwner
ERC20
allowance
approve
transfer
transferFrom
balanceOf
[Solidity]
require(bool)