Paid Networks

Security Assessment

January 24th, 2021

For :
Paid Networks

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Angelos Apostolidis @ CertiK
angelos.apostolidis@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are 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 CertiK to perform a security review.

CertiK Reports do 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.

CertiK Reports 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.

CertiK 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.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Paid Networks
Description	An upgradeable ERC20 implementation with enhanced features.
Platform	Ethereum; Solidity, Yul
Codebase	Previous Repository Current Repository
Commits	Phase1: 427ef8f47d1a68c062b0719aa68de4370e09e8b6 Phase2: eb960293f187b95ba1789a0fffc10fca5ffc3d8c Phase3: c375960a834a061b4f5a2e79231c22073f2f4fd9 Phase4: 55570b9c989b9c7d652c6eaa38dc89bbedb22587

Audit Summary

Delivery Date	January 24th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	January 13th, 2021 - January 24th, 2021

Vulnerability Summary

Total Issues	17
Total Critical	0
Total Major	1
Total Medium	3
Total Minor	4
Total Informational	9

Executive Summary

This report represents the results of CertiK's engagement with Paid Networks on their implementation of the Paid Networks token smart contract.

The audit consists of four phases, with the first two referring to the previous codebase the Paid Networks team had implemented, while the remaining reference to the current one.

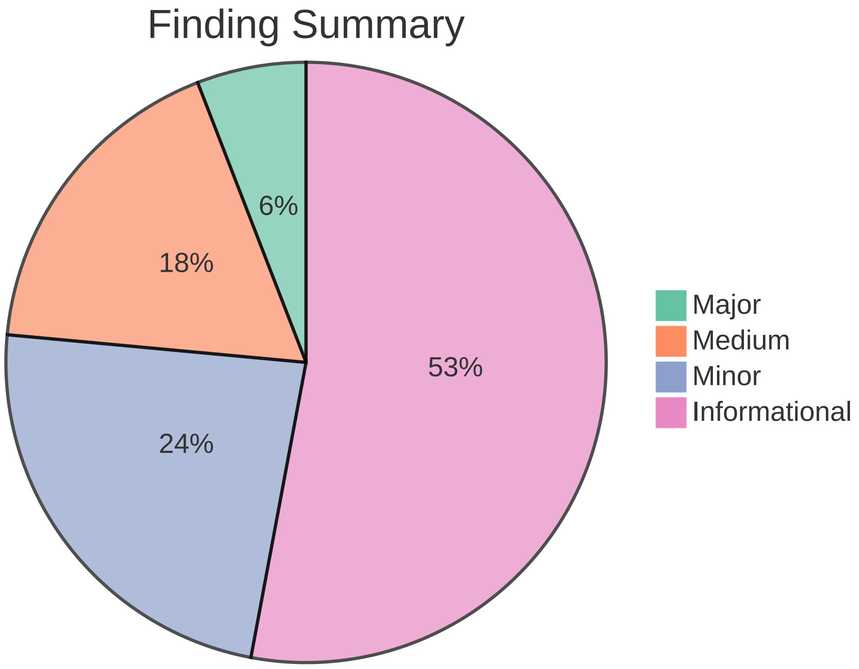
Phase1 consisted of analyzing the previous repository and proposing our findings to the team. Phase2 reviewed the fixes applied to the team's codebase, based on our exhibits. After Phase2, the team opted to make some changes to the smart contract's implementation, which led to Phase3. Phase3 was essentially going back to Phase1, yet targeting the new repository. Lastly, Phase4 ended the cycle by reviewing the fixes applied to the codebase altogether.

Our findings mainly refer to optimizations and Solidity coding standards. Hence, the issues identified pose no threat to the contract deployment's safety.

Files In Scope

ID	Contract	Location
PTNO	PaidToken.sol (Old Version)	contracts/PaidToken.sol
PTN	PaidToken.sol	contracts/PaidToken.sol

Findings


Finding Summary
6%
18%
53%
24%
Major
Medium
Minor
Informational

Findings Table (Previous Repository)

ID	Title	Type	Severity	Resolved
PTNO-01	Recommended Compiler Version	Language Specific	Informational	✓
PTNO-02	Ambiguous Variable	Gas Optimization	Minor	✓
PTNO-03	Redundant Type Casting	Gas Optimization	Informational	✓
PTNO-04	Conditionals Merge	Gas Optimization	Informational	✓
PTNO-05	Absence of the `SafeMath` Library	Mathematical Operations	Major	✓
PTNO-06	Ambiguous Functionality	Volatile Code	Minor	✓

Findings Table (Current Repository)

ID	Title	Type	Severity	Resolved
PTN-01	Ambiguous Comment	Inconsistency	Informational	✓
PTN-02	Division Before Multiplication	Mathematical Operations	Medium	✓
PTN-03	Ambiguous Conditional	Logical Issue	Informational	✓
PTN-04	Inexistent Input Sanitization	Volatile Code	Minor	✓
PTN-05	Redundant Array Look-Up	Gas Optimization	Informational	✓
PTN-06	Redundant Variable Cast	Gas Optimization	Informational	✓
PTN-07	`external` Over `public`	Gas Optimization	Informational	✓
PTN-08	Potential `Ether` Lock	Volatile Code	Medium	✓
PTN-09	`struct` Optimization	Gas Optimization	Informational	!✓
PTN-10	Ambiguous Functionality	Volatile Code	Medium	✓
PTN-11	Ambiguous Functionality	Logical Issue	Minor	✓

PTNO-01: Recommended Compiler Version

Type	Severity	Location
Language Specific	Informational	PaidToken.sol L2

Description:

The latest versions of Solidity fixed some the known security-relevant bugs.

Recommendation:

We advise to either use `v0.6.8` or `v0.6.11`.

Alleviation:

The development team opted to consider our references and used the `v0.6.11` Solidity compiler.

PTNO-02: Ambiguous Variable

Type	Severity	Location
Gas Optimization	Minor	PaidToken.sol L27

Description:

The amount of the tokens minted upon initialization does not match the value of the `totalToken` variable. Also, the `ERC20Upgradeable` contract already introduces the `totalSupply` variable, hence rendering its existance redundant.

Recommendation:

We advise to remove the `totalToken` variable.

Alleviation:

The development team opted to consider our references and removed the linked variable.

PTNO-03: Redundant Type Casting

Type	Severity	Location
Gas Optimization	Informational	PaidToken.sol L151

Description:

The `diff` variable is already declared as a `uint256` variable in L150.

Recommendation:

We advise to remove the variable type cast and directly use the `diff` variable in the linked statement.

Alleviation:

The development team opted to consider our references and removed the reundant type cast.

PTNO-04: Conditionals Merge

Type	Severity	Location
Gas Optimization	Informational	PaidToken.sol L157-L163

Description:

The linked conditionals could be merged into a single one separated with the OR operator, as both return the same value.

Recommendation:

We advise to merge the two conditionals into one via the OR operator.

Alleviation:

The development team opted to consider our references and merged the two conditionals into one `if` statement.

PTNO-05: Absence of the `SafeMath` Library

Type	Severity	Location
Mathematical Operations	Major	PaidToken.sol General

Description:

The contact introduces raw arithmetical operations, which could result in overflows/underflows for Solidity versions lower than `0.8.0`.

Recommendation:

We advise to change the arithmetical operations with `SafeMath`'s function invocations throughout the codebase to prevent potential overflows/underflows.

Alleviation:

The development team opted to consider our references and used the `SafeMath` library for the arithmetical operations throughout the codebase.

PTNO-06: Ambiguous Functionality

Type	Severity	Location
Volatile Code	Minor	PaidToken.sol L217-L220

Description:

The function `transferFrom()` overrides the respective ERC-20 function to check whether a `FrozenWallet` can complete the transaction. Yet, the said wallet will still be able to use the `transfer()` function nontheless.

Recommendation:

We advise to override the `_beforeTokenTransfer()` function instead.

Alleviation:

The development team opted to consider our references and implemented the `_beforeTokenTransfer()` function to override the one from the `ERC-20` standard.

PTN-01: Ambiguous Comment

Type	Severity	Location
Inconsistency	Informational	PaidToken.sol L40, L41

Description:

The in-line comment does not match the `monthlyRate` member of the `VestingType` introduced.

Recommendation:

We advise to either change the integer pushed into the vesting schedule or update the comment.

Alleviation:

The development team opted to consider our references and updated the in-line comments.

PTN-02: Division Before Multiplication

Type	Severity	Location
Mathematical Operations	Medium	PaidToken.sol L64, L65

Description:

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

Recommendation:

We advise to order multiplication before division.

Alleviation:

The development team opted to consider our references, implemented the `mulDiv()` function and applied it to the linked statements.

PTN-03: Ambiguous Conditional

Type	Severity	Location
Logical Issue	Informational	PaidToken.sol L77

Description:

The linked conditional will block the edge case where the mint that will cause the total supply to be equal to the max total supply.

Recommendation:

We advise to include equality on the linked conditional.

Alleviation:

The development team opted to consider our references and included the equality part of the conditional.

PTN-04: Inexistent Input Sanitization

Type	Severity	Location
Volatile Code	Minor	PaidToken.sol L56-L73

Description:

The `addAllocations()` function does not check that the `addresses` and `totalAmounts` input arrays are of equal length.

Recommendation:

We advise to add a `require` statement checking that the two arrays are of equal length.

Alleviation:

The development team opted to consider our references and added the proposed `require` statement.

PTN-05: Redundant Array Look-Up

Type	Severity	Location
Gas Optimization	Informational	PaidToken.sol L61

Description:

The linked loop conditional redundantly performs a query to the `length` member of the `addresses` array at the beginning of each iteration.

Recommendation:

We advise to store the `length` of the array in a local variable outside of the loop in order to save on the overall cost of gas.

Alleviation:

The development team opted to consider our references and changed the linked code segment as proposed.

PTN-06: Redundant Variable Cast

Type	Severity	Location
Gas Optimization	Informational	PaidToken.sol L62

Description:

The linked statement redundantly casts the `addresses` value to type `address`.

Recommendation:

We advise to remove redundant code.

Alleviation:

The development team opted to consider our references and removed the redundant code.

PTN-07: `external` Over `public`

Type	Severity	Location
Gas Optimization	Informational	PaidToken.sol L56, L105

Description:

The linked public functions are never used in the contract.

Recommendation:

We advise to change the attribute of the linked functions to `external`.

Alleviation:

The development team opted to consider our references and changed the visibility of the linked functions to `external`.

PTN-08: Potential `Ether` Lock

Type	Severity	Location
Volatile Code	Medium	PaidToken.sol L56-L73

Description:

The amount of `Ether` sent to the contract via the linked `payable` function will be lost.

Recommendation:

We advise to either implement a `withdraw` function or remove the `payable` attribute.

Alleviation:

The development team opted to consider our references and introduced the `withdraw()` function to the codebase.

PTN-09: `struct` Optimization

Type	Severity	Location
Gas Optimization	Informational	PaidToken.sol L9-L18

Description:

The `FrozenWallet` struct is not tightly packed.

Recommendation:

We advise to change the position of the `scheduled` member right after the `wallet` one, hence striving for a tight 256-bit packing.

Alleviation:

The Paid Networks development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

PTN-10: Ambiguous Functionality

Type	Severity	Location
Volatile Code	Medium	PaidToken.sol L82-L103

Description:

The `addFrozenWallet()` is implemented in a peculiar way. Also, the same function increases the centralization of the system by allowing the `owner` to arbitrarily `mint` tokens.

Recommendation:

We advise to revise the linked function or add descriptive documentation.

Alleviation:

The development team acknowledged this exhibit while commenting that they intend to `mint` the supply right after deployment. Also, the team stated the desire to disable the `minting` mechanism on a future contract upgrade.

PTN-11: Ambiguous Functionality

Type	Severity	Location
Logical Issue	Minor	PaidToken.sol L153-L171

Description:

The `canTransfer()` function returns `true` for every `address` that does not have a frozen wallet (by bypassing the conditionals).

Recommendation:

We advise to adjust the linked function if this is not intended functionality.

Alleviation:

The development team acknowledged this exhibit while commenting that this functionality is indeed intentional.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

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

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

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

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.