macro

Titan Node A-1

Security Audit

May 13, 2024
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Titan Node's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team on May 6, 2024.

The purpose of this audit is to review the source code of certain Titan Node Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
Low	1	-	-	1
Code Quality	6	-	-	6

Titan Node was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions with the Titan Node team.
- Available documentation in the repository.

Source Code

The following source code was reviewed during the audit:

- **Repository:** payment-stream
- **Commit Hash:** `bd0dce23217976949b164812c8ccf4db3e58a79d`

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~L-1~~ Some tokens don't revert on failed transfers

TOPIC	STATUS	IMPACT	LIKELIHOOD
ERC20	Fixed	Medium	Low

`PaymentStream` contract uses `IERC20:transfer()` function to send the payment token to the `payee` and/or to the `termReceiver` :

```
function claim() external {
    require(msg.sender == payee, "Not authorized");
    require(claimedAmount < paymentAmount, "All tokens have been claimed");
    require(!isTerminated, "Stream terminated");

    uint256 claimable = getClaimableAmount();

    claimedAmount += claimable;

    IERC20(paymentToken).transfer(payee, claimable);
}
```

Reference: PaymentStream.sol#L62-72

However, some non-compliant ERC20 tokens do not revert on failed transfers. This could cause the function `claim()` to succeed, incrementing the `claimedAmount` but not transferring the tokens, for example, if the tokens are not present in the contract.

Remediations to Consider:

Consider using the **SafeERC20** library to handle potential nonconventional ERC20 tokens or documenting this potential pitfall.

~~Q-1~~ Code consistency

TOPIC	STATUS	QUALITY IMPACT
Protocol Design	Fixed	Low

When deploying and initializing a `PaymentStream` in its constructor, input arguments are being partially verified with sanity checks.

```
constructor(
    address _payee,
    uint256 _duration,
    address _paymentToken,
    uint256 _paymentAmount,
    address[2] memory _termSigners,
    address _termReceiver
) {
    require(_duration > 0 && _paymentToken != address(0) && _paymentAmount > 0, "I
```

Reference: PaymentStream.sol#L28-36

Consider checking all address parameters to keep a consistent behavior across the contract.

~~Q-2~~ Mappings use the same key value can be merged

TOPIC	STATUS	QUALITY IMPACT
Storage Layout	Fixed	Medium

To simplify the storage structure, consider mergin `isTermSigner` and `_hasConfirmed` mappings into one mapping.

```
// Termination
mapping(address => bool) public isTermSigner;
...

mapping(address => bool) public _hasConfirmed;
```

Reference: PaymentStream.sol#L21-26

For example, both mappings share the same key and could be merged with a struct.

```
struct TermSign {
    bool isTermSigner;
    bool hasConfirmed;
}

mapping(address => TermSign) public termSigners;
```

Suggested code snippet.

~~Q-3~~ Declare immutable variables

TOPIC	STATUS	QUALITY IMPACT
Optimization	Fixed	Medium

Immutable variables will set the values passed to the constructor directly into the bytecode, disallowing any future changes for these and optimizing gas, as these values can directly be used as constants and avoid reading and writing storage.

```
contract PaymentStream {
    // Init
    address public payee;   //@audit can be immutable
    uint256 public duration;    //@audit can be immutable

    // Payment
    address public paymentToken;    //@audit can be immutable
    uint256 public paymentAmount;   //@audit can be immutable

    uint256 public startTime;   //@audit can be immutable
    uint256 public endTime;   //@audit can be immutable
    uint256 public claimedAmount;

    // Termination
    mapping(address => bool) public isTermSigner;
    address public termReceiver;   //@audit can be immutable
```

Reference: PaymentStream.sol#L9-22

Consider using the `immutable` keyword for all variables that are only set once in the contract's constructor.

~~Q-4~~ Lack of events

TOPIC	STATUS	QUALITY IMPACT
Events	Fixed	Medium

The current `PaymentStream` implementation lacks events for important and functional operations in general, `claim()` and `terminate()` functions could emit events for easier data retrieval and allow external protocols to index past events efficiently.

~~Q-5~~ Unnecessary use of external party library

TOPIC	STATUS	QUALITY IMPACT
Architecture	Fixed	High

`PRB` library is currently used to calculate on a linear basis the corresponding claimable amounts of the payee, according to the time base variables and the current `block.timestamp` .

```
claimable = (ud(paymentAmount).mul(ud(timeElapsed).div(ud(duration)))).intoUint256() -
```

Reference: PaymentStream.sol#L59

This operation can be implemented with inbuilt solidity language features, avoiding unnecessary casting operations and the use of a third-party library.

~~Q-6~~ Naming convention

TOPIC	STATUS	QUALITY IMPACT
Standards	Fixed	Low

Mapping `_hasConfirmed` starts with an underscore, using the `internal` / `private` naming convention even though it's public.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Titan Node team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.