Titan Legends Security Audit

Report Version 1.0

March 15, 2024

Conducted by:

George Hunter, Independent Security Researcher

Table of Contents

1 About George Hunter

George Hunter is a proficient and reputable independent smart contract security researcher with over 50 solo and team security engagements contributing to the security of numerous smart contract protocols in the past 2 years. Previously held roles include Lead Smart Contract Auditor at Paladin Blockchain Security and Smart Contract Engineer at Nexo. He has audited smart contracts for clients such as LayerZero, Euler, TraderJoe, Maverick, Ambire, and other leading protocols. For security review inquiries, you can reach out to us on Telegram or Twitter at *@georgehntr*.

2 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities **but cannot guarantee their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	Titan Legends
Repository	https://github.com/TitanLegends/Titan-Legends
Commit hash	70ce83292bfe7ddaf146dc46a8db04eda5fb8ccc
Resolution	eecda81180227dab03848ea6e8184d4251ed1741
Methods	Manual review

TitanLegendsMarketplace.sol
TitanLegendsBattlefield.sol
LGNDX/Token.sol

Issues Found

High risk	1
Medium risk	0
Low risk	2
Informational	2

5 Findings

5.1 High

5.1.1 A malicious seller could steal a buyer's entire balance

Severity: *High*

Context: TitanLegendsMarketplace.sol#L66-L87, TitanLegendsMarketplace.sol#L105-L114

Description: A seller of an NFT is able to list it at price X and later edit the price to whatever amount they wish. When purchasing an NFT the buyer only specifies the listing ID they would like to fulfill. The problem is that a malicious seller could front-run the buyer's *buyListing* transaction and eventually edit the price to whatever the buyer's whole balance is. If the buyer has given max approval to the NFT marketplace contract (which is a common scenario) the NFT will be purchased for the whole buyer's balance instead of just the price the buyer intended to purchase the NFT for.

Recommendation: Consider passing the price the buyer is willing to pay for the NFT as an input parameter of *buyListing* and comparing to the one stored on-chain.

Resolution: Resolved. The recommended fix was implemented.

5.2 Low

5.2.1 Tokens may get stuck in the Battlefield contract

Severity: *Low*

Context: TitanLegendsBattlefield.sol

Description: The TitanLegendsBattlefield contract allows users to exchange their TitanLegends NFTs for a predetermined amount of tokens based on the NFT token ID. The amount of LegendX tokens that users will be able to get in exchange for their NFTs should be transferred directly to the contract address.

The problem is that if there is a certain amount of LegendX tokens deposited that is more than the corresponding demand, there is no functionality that would allow the protocol team to get back the tokens, effectively locking them until someone decides to call *claimBounty*. Additionally, if there are more tokens deposited than they can ever be claimed, the extra amount will also be locked forever.

Recommendation: Consider implementing a recovery function which allows the owner of the contract to withdraw a certain amount of LegendX tokens that no one intends to claim.

Resolution: Acknowledged. Client's response: We have intentionally left this function out of the contract the bounty pool is intended to be locked with no function to withdraw any tokens from it unless someone decides to call claim bounty. The issue of additional tokens being locked inside Battlefield contract only applies if someone just sends tokens to the smart contract instead of calling payRansom function.

5.2.2 The balance limit functionality may be exploited by a compromised owner

Severity: *Low*

Context: Token.sol#L89-L91, Token.sol#L57-L72

Description: The LegendX token contract allows the owner of the contract to set a maximum balance limit that would enforce all holders of the tokens to have at most X amount of tokens at the end of every transfer. There is a whitelist mapping *isExcludedFromLimits* that allows the owner to exclude specific address from this constraint.

The problem is that a compromised owner may unset certain addresses like a Uniswap pool and apply a balance limit of the minimal possible value which is equivalent to *totalSupply() / 1000*. Thus, the pool or any (target/victim address) that has >0.1% of the total supply and cannot transfer out enough tokens to fit in the limit, would basically be blocked.

Recommendation: Consider whether this centralization risk is desired. An effective mitigation would be to remove the max balance limit constraint.

Resolution: Acknowledged. Client's response: The max wallet is in the token contract to prevent a sniper from scooping up a majority of the circulating supply within the first minutes of launch. After the first hour or so when the token goes live, we plan to lift the max wallet and renounce the contract, leaving the token to be completely decentralized.

5.3 Informational

5.3.1 Smart wallets will not be able to interact with the marketplace

Severity: *Informational*

Context: TitanLegendsMarketplace.sol#L54, TitanLegendsMarketplace.sol#L67, TitanLegendsMarketplace.sol#L90, TitanLegendsMarketplace.sol#L106

Description: The following check is applied to every user-facing function in the TitanLegendsMarketplace contract:

```
require(tx.origin == _msgSender(), "Contracts are prohibited");
```

The goal is to prevent the functions from being called by smart contract hence reducing the attack surface. However, these check will also prevent smart wallet and AA users from interacting with the marketplace

Recommendation: Consider whether you'd like to support smart wallet users. If so, consider removing the above check and using re-entrancy guards instead (as well as following the Checks-Effects-Interactions pattern).

Resolution: Resolved. The recommended fix was implemented.

5.3.2 Typographical mistakes, code-style suggestions and non-critical issues

Severity: *Informational*

Description: The contract contains one or more typographical issues, code-style suggestions or non-critical issues. In an effort to keep the report size reasonable, we enumerate these below:

1. Consider using Solidity pragma version *0.8.24* instead of *0.8.22*.
2. Make sure the contracts are compiled with EVM version set to *paris* in order to prevent EVM incompatibility issues between different blockchains such as not supporting the *PUSH0* opcode.
3. Consider importing the OZ library dependency contract instead of using an identical local version of them in the LGNDX directory.
4. The *unicode* keyword in Token.sol seems redundant.
5. Excluding address(this) and address(0) from limits seems unnecessary:

```
_excludeFromLimits(address(this), true);
_excludeFromLimits(address(0), true);
```

6. Consider running *forge fmt .* in to get rid of redundant spacing/tabulation.
7. Consider using *2_888_888_888e18* instead of *28888888880 * (10 ** decimals()) / 10* in Token.sol:

```
updateMaxWalletAmount(28888888880 * (10 ** decimals()) / 10);
_mint(supplyRecipient, 28888888880 * (10 ** decimals()) / 10);
```

8. The *receive()* function in Token.sol is redundant and may lead to users accidentally locking funds in the contract.
9. There is no need to override *beforeTokenTransfer()* in Token.sol.
10. Consider using *IERC721* instead of *IERC721A* as the function interfaces used are the same.
11. Consider using *Ownable2Step* instead of *Ownable* in TitanLegendsMarketplace to prevent the owner from accidentally transferring the ownership of the contract to an incorrect address.
12. *feeStorage* , *titanLegends* and *titanX* in TitanLegendsMarketplace can be marked *immutable* to save gas.
13. *feeStorage* in TitanLegendsMarketplace should never be set to *address(0)*.
14. *addListing* , *buyListing*, *removeListing*, *editListing* can be marked *external*.
15. *buyListing* should not be *payable*.
16. Redundant line - *require(!isListingActive(currentListingId), "Listing already active");*.
17. Do not use *_msgSender()*, but *msg.sender*.
18. *marketplaceFee* should never be set to 0, or the transfer shouldn't execute when the fee is 0.
19. Typo - *MArketplace -> Marketplace*.
20. *tokensPerBountyPoint* could be marked *constant* instead of *immutable*.

21. There is a possible re-entrancy risk in *payRansom* which would allow a user to reenter from line 104 where the NFT is sent back to them. The re-entrancy would allow them to re-enter into either:

- *payRansom* - if they re-enter into *payRansom*, they could withdraw the same NFT multiple times which would only make them pay.
- *claimBounty* - if they re-enter into *claimBounty*, they could lose their NFT since it will be deposited again but then removed from the *_battles* set.
  Consider adding *nonReentrant* modifier guards.

22. The TitanLegends NFT token transfer in *removeListing* should happen after state changes to properly adhere to the Checks-Effects-Interactions pattern.

Recommendation: Consider implementing the aforementioned suggestions.

Resolution: Resolved. The majority of the above recommendations were implemented.