Chain Integrity

The Web 3.0 Ethical Hacking Company

Minted Teddy

Smart Contract Audit Deliverable

Date:	Nov 17, 2021
Version:	2

Table of Contents

1.1 Summary

Project

Name	Minted Teddy
Description	Metaverse, Redeemable Vouchers, NFTs
Platform	Ethereum (ETH)
Codebase	*
Commit	*

Engagement

Delivered	Nov 17, 2021
Methods	Static Analysis, Manual Review, TM, RA.
Consultants	2
Timeline	3 Days (Review Update)

Observations

Total	0	**Status**
Critical	1	Fixed (1/1)
High	0	
Medium	1	Fixed (1/1)
Low	5	Fixed (5/5)
Undetermined	0	

Executive

This document has been prepared for Minted Teddy (Client) to discover and analyze the codebase provided by the team for security vulnerabilities, code correctness, and risk of investment. The codebase has been comprehensively examined using structural analysis, behavioral analysis, and manual review techniques.

Throughout the audit, caution was taken to ensure that the smart contract(s):

- Implements robust functions which are safe from well-known attack vectors.
- The logic and behavior adheres to the associated documentation and code comments.
- Transfer flows are designed in a sustainable way, safeguarded from i.e. infinite loops.
- Does not hide potential back doors and implements sanity checks where suitable.

1.2 Scope

File	Fingerprint (SHA-256 Checksum)
MintedTeddy.sol	40d081b5a87b4e0be598744cfb3b641620ee1f3cfa4ed03d78db338d01abb3a2

1.3 Documentation

The smart contract in scope is documented partially but the naming style makes the remaining code easy to comprehend. Finally, all observations are explicitly based on the information available in the provided codebase and whitepaper.

1.4 Review Notes

The majority of the code optimization recommendations highlighted in the observations list refer to best practices and inefficiencies. The observations of type 'Overpowered Design' can place increased risk on investors as a result of centralized access permissions and data manipulation.

1 critical issue was found that allows the owner to update the metadata of a desired token and influence rarity or the visual representation of the token ID.

The latest version of the codebase has been optimized to follow best practices and some inefficiencies have been fixed as well. The observations of type 'Overpowered Design' now include additional guards to reduce investor risk. The critical issues have been discussed and resolved.

1.5 Recommendations

The codebase in scope should be fixed to reflect the recommendations presented in this document. Optimize the code to lower the gas cost of minting for potential investors. Next, order the smart contract layouts properly and use a consistent styling guide to achieve a high code quality standard.

The latest version of the codebase now reflects the recommendations presented in this document and the smart contract is layered properly for clarity of understanding.

1.6 Disclaimer

It should be noted that this document is not an endorsement of the effectiveness of the smart contracts, rather limited to an assessment of the logic and implementation. This audit should be seen as an informative practice with the intent of raising awareness on the due diligence involved in secure development and make no material statements or guarantees in regards to the operational state of the smart contract(s) post-deployment. Chain Integrity (Consultant) do not undertake responsibility for potential consequences of the deployment or use of the smart contract(s) related to the audit.

For clarity of understanding, observations are arranged from critical to informational. The severity of each issue is evaluated based on the risk of exploitation or other unexpected behavior.

Critical

An issue flagged as critical means that it can affect the smart contract in a way that can cause serious financial implications, catastrophic impact on reputation, or disruption of core functionality.

High

An Issue flagged as high means that it can affect the ability of the smart contract to function in a significant way i.e. lead to broken execution flows or cause financial implicants.

Medium

An Issue flagged as medium means that the risk is relatively small and that the issue can not be exploited to disrupt execution flows or lead to unexpected financial implications.

Undetermined

An issue flagged as undetermined means that the impact of the discovered issue is uncertain and needs to be studied further.

Low

An Issue flagged as informational does not pose an immediate threat to disruption of functionality, however, it should be considered for security best practices or code integrity.

2.2 Observations List

ID	Title	Type	Severity
OBSN-01	Owner Metadata Manipulation	Overpowered Design	︽
OBSN-02	Unsafe Transfer & Mint Implementation	Code Optimization	—
OBSN-03	Duplicate Code	Code Optimization	﹀
OBSN-04	Repetitive Variable Readings	Code Optimization	﹀
OBSN-05	Lazy Control Structure Conditions	Code Optimization	﹀
OBSN-06	Explicit Function Visibility Markings	Code Optimization	﹀
OBSN-07	Inconsistent Styling	Code Optimization	﹀

⬜ Unchanged ▢ Improved 🟩 Fixed

2.3 Observations Review

OBSN-01

Location: MintedTeddy.sol

Explanation:

The owner is capable of updating the metadata of any token IDs minted using a voucher that is not initially in a locked state. The updateMetadata(...) function can also keep the metadata of a token ID in an unlocked state forever by always using a false lock argument.

There is nothing wrong with the implementation and the issue is highlighted for the sake of investor risk. Unlocked metadata leaves a door open for drastic changes to the value of the token ID.

i

OBSN-02

Location: MintedTeddy.sol

Explanation:

Usage of both _mint(...) and _transfer(...) is discouraged because of the associated risk of locking tokens away forever, whenever a recipient is a smart contract with no support for the ERC721 standard.

Recommendation:

Favor _safeMint(...) and _safeTransfer(...) over _mint(...) and _transfer(...) functionality when possible to prevent tokens from being locked away forever as a consequence of no support for the ERC721 standard.

OBSN-03

Location: MintedTeddy.sol

Explanation:

The mintNFT(...) function is executing both _mint(...) and _transfer(...) calls in a duplicative way and a _mint(...) call would be enough to save a considerable amount of gas. The _mint(address to, uint256 tokenId) function assigns the token to the first address param to which is equivalent to the _safeTransfer(...) function where the address param from is address(0) (DEAD ADDRESS) with less gas consumption.

Recommendation:

Remove the _transfer(...) call expression from the mintNFT(...) function, it serves no purpose and will save the community a lot of gas in the end. If required for other reasons, then consider replacing it with safeTransfer(...) with the reason given in former observations.

OBSN-04

Location: MintedTeddy.sol

Explanation:

The current token id is retrieved from storage multiple times during execution of the mintNFT(...) function which has a negative effect on gas consumption. Gas optimization is extremely important for NFTs since the community will be doing the minting and the sensitive nature of the host network.

Recommendation:

Consider assigning the _tokenIds.current() value to a local variable when the value is accessed multiple times during a function execution.

OBSN-05

Location: MintedTeddy.sol

Explanation:

The correctness of the safeParseInt(...) implementation is sound but gas consumption can be further optimized as the function is called only with condition (_b) = 0.

Recommendation:

The function depends on the following control structure conditions: b = 0 or b != 0. However, the function is only running through _b=0, so gas consumption can be optimized by removing the b != 0 condition.

OBSN-06

Location: MintedTeddy.sol

Explanation:

Mark the visibility of the implemented functions explicitly (to ease the process of catching incorrect assumptions concerning operational accessibility and to potentially reduce gas costs). The visibility modifiers of the following functions can be further restricted from public to external because the functions are never accessed internally:

mintNFT(...)
withdraw(...)

Finally, all internal functions declared in MintedTeddy.sol should also be changed to private to align with best practices since they are not used internally in the inheritance graph.

Recommendation:

Change the visibility modifiers of the previously mentioned functions from being public to external for code correctness and to enforce call data on function parameters which will reduce gas costs.

OBSN-07

Location: MintedTeddy.sol

Explanation:

Consider implementing the styling guide outlined in the Solidity documentation (not only in terms of naming conventions but also in regards to order of layout, order of functions, and order of function modifiers). Adhering to a styling guide makes it easier to mentally build an overview of the codebase and the exposed endpoints (attack surface).

Recommendation:

It is not recommended to mutate function params with local variables as seen in the safeParseInt(...) function because it has an impact on code correctness and can result in behavioral inconsistencies. Finally, it is highly recommended to group functions based on type and visibility for clarity of understanding.

No security issues were found during the assessment and the smart contracts in scope pass our auditing process.

Non-fungible tokens are one of the most important components ofthe Metaverse and the Minted Teddy project is an NFT PFP offspring dedicated to shaping the future of sustainable kids toys.

The smart contracts in scope adhere to best practices in terms of security and it is evident that an effort has been made to deliver a codebase of high quality.

The statements made in this report do not constitute legal or investment advice and I should not be held accountable for decisions made based on them.

Appendix

A. Source Code Fingerprints

FILE	FINGERPRINT (SHA-256 Checksum)
MintedTeddy.sol	40d081b5a87b4e0be598744cfb3b641620ee1f3cfa4ed03d78db338d01abb3a2