Zellic

MightyNet V2

Smart Contract Security Assessment

July 3, 2023

Prepared for:

Fadzuli Said

Mighty Bear Games

Prepared by:

Nipun Gupta and Yuhang Wu

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for Mighty Bear Games from June 16th to June 20th, 2023. During this engagement, Zellic reviewed MightyNet V2's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Are there any vulnerabilities in the system that could allow players to bypass checks for token burn/mint as part of the import/export feature, potentially leading to their assets ending up in a broken state?
- Can a malicious party exploit the MightyNetShop to deceive the backend into providing items to them?
- How secure are the MightyNetTerminal and MightyNetShop components of the system?

1.2 Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- The administrator's operation to control game items and stop in-game transactions
- Examination of off-chain components
- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide. As a result of scope limitations, we were unable to conduct a comprehensive evaluation of every aspect of the system during this assessment.

1.3 Results

During our assessment on the scoped MightyNet V2 contracts, we discovered no findings.

Zellic recorded its notes and observations from the assessment for Mighty Bear Games's benefit in the Discussion section (3) at the end of the document.

2 Introduction

2.1 About MightyNet V2

MightyNet V2 contains the smart contracts for the MightyNet ecosystem by Mighty Bear Games.

2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review the contracts' external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the code base in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood.

There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of one person-week. The assessment was conducted over the course of one calendar week.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald, Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Nipun Gupta, Engineer
nipun@zellic.io

Yuhang Wu, Engineer
yuhang@zellic.io

2.4 Project Timeline

The key dates of the engagement are detailed below.

June 16, 2023	Start of primary review period
June 20, 2023	End of primary review period

3 Discussion

3.1 Players can front-run to purchase items before the admin changes the game item prices

It is worth noting that there is a function to modify item prices in the contract.

```
function setShopItemPrice(uint256 shopItemId, uint256 price)
        external
        onlyRole(DEFAULT_ADMIN_ROLE)
        isValidShopItemId(shopItemId)
    {
        shopItems[shopItemId].price = price;

        emit ShopItemUpdate(shopItemId, shopItems[shopItemId]);
    }
```

If the admin wants to increase the price of an item, players can front-run to purchase items at a low price. If the game item prices are set incorrectly at the beginning, players might make a substantial profit from it.

Zellic notes that Mighty Bear Games have considered this and will disable purchases before updating prices.

Mighty Bear Games's response:

> The process when updating a shop item's price is to first set it to not enabled (Players will not be able to buy items), update price, set shopItem to be enabled. This will prevent players to be able to front-run purchases ahead of price increases.

4 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the smart contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

4.1 Module: MightyNetShop.sol

Function: `purchaseItem(uint256 shopItemId, uint256 supplyToBuy)`

This is the function used to purchase an item.

Inputs

- `shopItemId`
  - **Control**: Fully controlled.
  - **Constraints**: Should be a valid item ID.
  - **Impact**: An item with this ID would be purchased.
- `supplyToBuy`
  - **Control**: Fully controlled.
  - **Constraints**: Should be enough supplies of the item.
  - **Impact**: This is the number of supplies that would be purchased.

Branches and code coverage (including function calls)

Intended branches

- If the correct amount of `msg.value` is supplied, the total supply of that item would be decreased.
  ☑ Test coverage

Negative behavior

- Should not be able to purchase zero items.
  ☑ Negative test
- Should not be able to purchase more than supply.
  ☑ Negative test

- Should not be able to purchase if item ID does not exist.
  ☑ Negative test
- Should not be able to purchase if item static ID is empty.
  ☑ Negative test
- Should not be able to purchase if price is 0.
  ☑ Negative test
- Should not be able to purchase if item is disabled.
  ☑ Negative test
- Should not be able to purchase if provided `msg.value` is wrong.
  ☑ Negative test

4.2 Module: MightyNetTerminal.sol

Function: `requestReceiveFromGameERC721(address contractAddress, uint256[] tokenIds)`

Receive from game ERC-721.

Inputs

- `contractAddress`
  - **Control**: Fully controlled by user.
  - **Constraints**: Should be registered with type `ERC721_CONTRACT_TYPE`.
  - **Impact**: N/A.
- `tokenIds`
  - **Control**: Fully controlled by user.
  - **Constraints**: N/A.
  - **Impact**: N/A.

Branches and code coverage (including function calls)

Intended branches

- Should pay fee to the `feeVault`.
  ☑ Test coverage

Negative behavior

- Should revert when transferring fee below minimum amount.
  ☑ Negative test
- Should revert if fee vault is not set.
  ☑ Negative test

- Should not be able to receive to game with unregistered token contract.
  ☑ Negative test
- Should not be able to receive to game a differently registered token contract.
  ☑ Negative test
- Should revert on reentry.
  ☐ Negative test

Function: `requestReceiveFromGameERC1155(address contractAddress, uint256[] tokenIds, uint256[] quantities)`

Receive from game ERC-1155.

Inputs

- `contractAddress`
  - **Control**: Fully controlled by user.
  - **Constraints**: Should be registered with type `ERC1155_CONTRACT_TYPE`.
  - **Impact**: N/A.
- `tokenIds`
  - **Control**: Fully controlled by user.
  - **Constraints**: N/A.
  - **Impact**: N/A.

Branches and code coverage (including function calls)

Intended branches

- Should pay fee to the `feeVault`.
  ☑ Test coverage

Negative behavior

- Should revert when transferring fee below minimum amount.
  ☑ Negative test
- Should revert if fee vault is not set.
  ☑ Negative test
- Should not be able to receive to game with unregistered token contract.
  ☑ Negative test
- Should not be able to receive to game a differently registered token contract.
  ☑ Negative test
- Should revert on reentry.
  ☐ Negative test

5 Audit Results

At the time of our audit, the audited code was not deployed to the Ethereum Mainnet.

During our assessment on the scoped MightyNet V2 contracts, we discovered no findings.

5.1 Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the audit version of our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.