hexens × Fungify.

FEB.24

SECURITY REVIEW REPORT FOR FUNGIFY

CONTENTS

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

EXECUTIVE SUMMARY

OVERVIEW

This audit covered a new commit for Fungify, a new lending protocol that builds on Compound to support lending/borrowing of NFTs. This commit introduced the FungifyNFT, NFTMinter and Proxy2Step contracts.

Our security assessment was a full review of the smart contract changes in the commit, spanning a total of 3 days.

During our audit, we have identified 1 high severity vulnerability in the NFTMinter contract. We have also identified various minor vulnerabilities and code optimisations.

Finally, all of our reported issues were fixed or acknowledged by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

SCOPE

The analyzed resources are located on:

https://github.com/fungify-dao/taki-contracts/commit/587471b627224f42857dbb31ebc995197ce1bd90

The issues described in this report were fixed in the following commit:

https://github.com/fungify-dao/taki-contracts/commit/4fd8b80c00cc5fb8f3e957371d9be646624c0c2d

AUDITING DETAILS

STARTED
26.02.2024

DELIVERED
28.02.2024

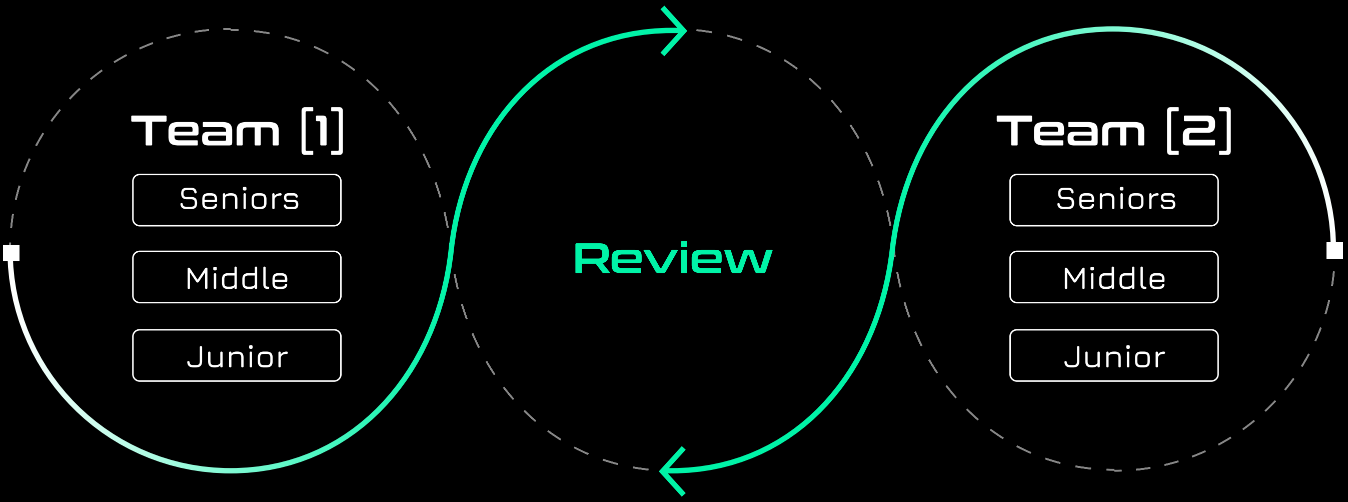
Review Led by

KASPER ZWIJSEN

Head of Smart Contract Audits | Hexens

HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.



SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

Impact	Probability			
	rare	unlikely	likely	very likely
Low/Info	Low/Info	Low/Info	Medium	Medium
Medium	Low/Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

High

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

Medium

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

Low

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

Informational

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

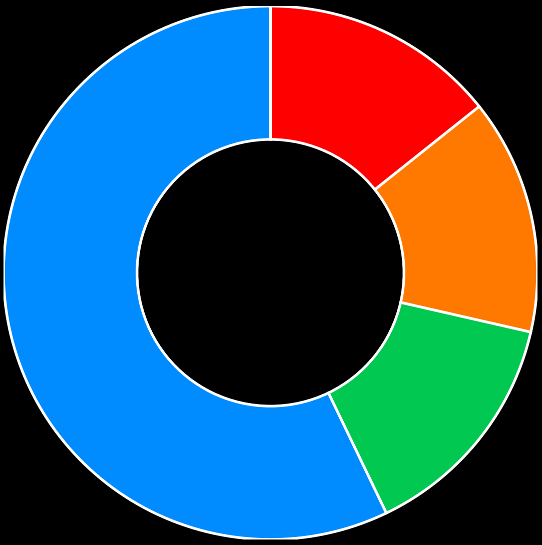
ISSUE SYMBOLIC CODES

Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

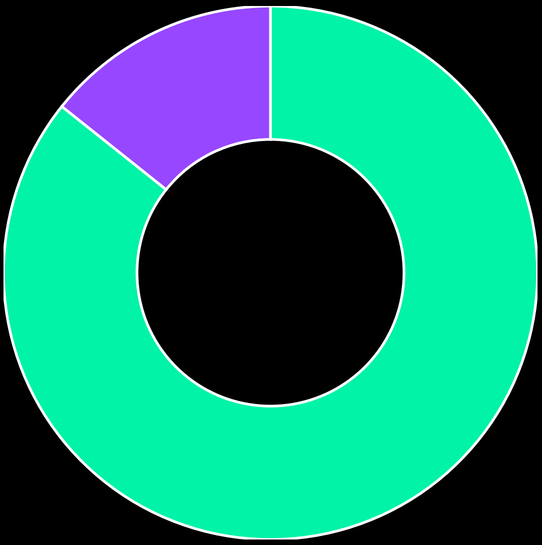
FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	1
Medium	1
Low	1
Informational	4

Total: 7



● High
● Medium
● Low
● Informational



● Fixed
● Acknowledged

WEAKNESSES

This section contains the list of discovered weaknesses.

FNGF-4

NFT MINTER EXCHANGE RATE IS NOT SCALED TO 18 DECIMALS

SEVERITY: High

PATH:

Token/NFTMinter.sol:L25

REMEDIATION:

We recommend to scale the EXCHANGE_RATE to 18 decimals.

For example:

uint256 public constant EXCHANGE_RATE = 420000 * 10 ** 18;

STATUS: Fixed

DESCRIPTION:

The **mintNFT** function allows a user to pay FUNG tokens to mint a Fungify NFT. The amount of FUNG tokens required to mint one NFT is based on the constant **EXCHANGE_RATE** of 420000.

However, the FUNG token has 18 decimals and the exchange rate is not scaled by 18 decimals. This means that even 1 FUNG token would be able to instantly mint the supply cap of 2300, as currently the price is a very tiny amount.

```solidity
uint256 public constant EXCHANGE_RATE = 420000;

function mintNFT(uint256 fungAmount) public {
    if (fungAmount % EXCHANGE_RATE != 0 || fungAmount == 0) {
        revert InvalidDepositAmount();
    }
    uint256 mintCount = fungAmount / EXCHANGE_RATE;
    if (nftContract.totalSupply() + mintCount > mintCurrentCap) {
        revert ExceedsCurrentCap(mintCount);
    }

    fungToken.transferFrom(
        msg.sender,
        address(this),
        fungAmount
    );

    nftContract.mint(msg.sender, mintCount);

    emit NFTsMinted(msg.sender, mintCount);
}
```

FNGF-1

FUNGIFY NFT DOES NOT CHECK OR DELETE ERC4907 USER INFORMATION UPON TRANSFER

SEVERITY: Medium

PATH:

Token/FungifyNFT.sol:L11

REMEDIATION:

We recommend to handle the ERC4907 information correctly upon changes of NFT ownership.

STATUS: Fixed

DESCRIPTION:

The Fungify NFT extends ERC4907A, which allows owners of tokens to set a temporary user of the NFT (i.e. allowing for NFT rentals). ERC4907A uses the mapping **_packedUserInfo** to store this information and the owner can set this using **setUser**.

However, Fungify NFT does not handle this information upon transferring or burning of the NFT, such as by overriding **_beforeTokenTransferor** **_afterTokenTransfer** as is shown in the EIP example implementation: ERC-4907: Rental NFT, an Extension of EIP-721 .

In the example implementation, the user information is deleted upon transfer. It could also be possible to check if the NFT is currently rented out (i.e. timestamp is not expired) and if so, the transfer could be blocked.

```solidity
import "@erc721a/extensions/ERC4907A.sol";

contract FungifyNFT is Upgradeable2Step, ERC721A, ERC721AQueryable, ERC4907A {
  [..]
}
```

FNGF-3

SAFE MINT WITH ONERC721RECEIVED CALLBACK SHOULD BE USED OVER MINT

SEVERITY: Low

PATH:

FungifyNFT.sol:mint():L28-30

REMEDIATION:

Use safeMint instead of mint to check whether the received address supports ERC721:
https://github.com/chiru-labs/ERC721A/blob/b3517b062cfe00d032ff787983a89f3fa790d1ec/contracts/ERC721A.sol#L945

STATUS: Fixed

DESCRIPTION:

When minting NFT the **NFTMinter.sol** calls **nftContract.mint()**, which in turn calls **_mint()** of ERC721A, which is not checking the **onERC721Received** callback on the receiver.

The **msg.sender** will mint an NFT when **NFTMinter.mintNFT()** is called. However, if **msg.sender** is a contract address that does not support ERC721, it will not be notified with the **onERC721Received** callback and the NFT could become frozen in the contract, as expected as per the ERC721 docs:

Reference: ERC-721: Non-Fungible Token Standard

```
function mint(address receiver, uint256 quantity) external auth {
    _mint(receiver, quantity);
}
```

FNGF-2

CONSTANT VARIABLES SHOULD BE MARKED AS PRIVATE

SEVERITY: Informational

PATH:

Token/NFTMinter.sol

REMEDIATION:

See description

STATUS: Acknowledged, see commentary

DESCRIPTION:

The **NFTMinter.so**l uses **public** visibility for constant parameters. Setting constants to **private** will save deployment gas. This is because the compiler won't have to create non-payable getter functions for deployment **calldata**, won't need to store the bytes of the values outside of where it's used, and won't add another entry to the method ID table. If necessary, the values can still be read from the verified contract source code.

```
address public constant BURN_ADDRESS = address(0xdead);
uint256 public constant MAX_BURN_FEE = 1000;
uint256 public constant FEE_PERCENT_DENOMINATOR = 10000;
uint256 public constant MINT_TOTAL_CAP = 2380;
uint256 public constant EXCHANGE_RATE = 420000;
```

```diff
-    address public constant BURN_ADDRESS = address(0xdead);
-    uint256 public constant MAX_BURN_FEE = 1000;
-    uint256 public constant FEE_PERCENT_DENOMINATOR = 10000;
-    uint256 public constant MINT_TOTAL_CAP = 2380;
-    uint256 public constant EXCHANGE_RATE = 420000;
+    address private constant BURN_ADDRESS = address(0xdead);
+    uint256 private constant MAX_BURN_FEE = 1000;
+    uint256 private constant FEE_PERCENT_DENOMINATOR = 10000;
+    uint256 private constant MINT_TOTAL_CAP = 2380;
+    uint256 private constant EXCHANGE_RATE = 420000;
```

Commentary from the client:

“ - Not updating. We would rather keep these public despite the increased deployment cost, for user convenience.”

FNGF-5

AVOID USING THE UNDERSCORE PREFIX FOR EXTERNAL FUNCTIONS

SEVERITY: Informational

PATH:

Token/NFTMinter.sol:_setCurrentCap, _setBurnFee (L82-88, L90-96)

REMEDIATION:

See description

STATUS: Fixed

DESCRIPTION:

Based on the guidance provided at Style Guide — Solidity 0.8.25 documentation it is advised to use the underscore prefix for internal or private functions rather than external ones. However, the external functions _setCurrentCap and _setBurnFee do not adhere to this style guide. This deviation could potentially complicate the readability of the codebase.

```
function _setCurrentCap(uint256 _mintCurrentCap) external onlyOwner {
      if (_mintCurrentCap > MINT_TOTAL_CAP) {
          revert CapTooHigh();
      }
      emit CurrentCapUpdated(mintCurrentCap, _mintCurrentCap);
      mintCurrentCap = _mintCurrentCap;
  }

  function _setBurnFee(uint256 _burnFee) external onlyOwner {
      if (burnFee > MAX_BURN_FEE) {
          revert FeeTooHigh();
      }
      emit BurnFeeUpdated(burnFee, _burnFee);
      burnFee = _burnFee;
  }
```

FNGF-7

EXTERNAL FUNCTION PARAMETERS COULD BE MARKED AS CALLDATA

SEVERITY: Informational

REMEDIATION:

Change the mentioned parameter to calldata instead of memory.

STATUS: Fixed

DESCRIPTION:

We have identified the following locations where **memory** parameters of external functions could be marked as **calldata** in favour of saving gas:

1. NFTMinter.sol: uint[] memory nftIds on line 59.
2. FungifyNFT.sol: uint[] memory nftIds on line 32 and 39.

```
function burn(uint[] memory nftIds) external auth {
  [..]
}

function burnIfOwnedByUser(uint[] memory nftIds, address user) external auth
{
  [..]
}

function burnNFT(uint[] memory nftIds) public {
  [..]
}
```

FNGF-6

INCONSISTENCY IN OWNERSHIP VERIFICATION FOR BURN FUNCTIONALITY

SEVERITY: Informational

REMEDIATION:

By removing the first burn function, the contract's behavior becomes more aligned with decentralized principles, ensuring that only rightful owners can burn their NFTs.

STATUS: Fixed

DESCRIPTION:

The FungifyNFT.sol contract exhibits inconsistency in ownership verification within its burn functionality. Specifically, there are two distinct burn functions implemented in the contract:

1. The first burn function allows any authorized address to burn NFTs without verifying if the caller is the rightful owner of the NFTs being burned. This lack of ownership verification could introduce centralization concerns, as it enables authorized addresses to burn NFTs that they do not own.
2. In contrast, the second burn function (burnIfOwnedByUser) verifies whether the NFTs being burned belong to a specific user before allowing the burn operation to proceed. This function provides a level of ownership verification, ensuring that the NFT can only be burned if the correct owner was provided.

This inconsistency in ownership verification mechanisms raises concerns regarding the decentralized nature of the contract and could potentially lead to misuse or exploitation.

```
function burn(uint[] memory nftIds) external auth {
    for(uint i; i < nftIds.length;) {
        _burn(nftIds[i]);
        unchecked { i++; }
    }
}
```

hexens × Fungify.