our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Harvest Flow V2

SECURITY REVIEW

Date: 4 March 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Harvest Flow V2

Harvest flow introduces an innovative approach to lending via exposure to tangible assets that can be monitored in real time. This is achieved by embedding IoT devices in each asset financed through the platform. These devices provide 24/7 location and usage data, ensuring transparency and security for both lenders and borrowers. The devices are linked to the digital assets with the help of NFTs, turning these agreements into verifiable, secure digital assets on the blockchain. This not only simplifies the management of contracts but also opens new avenues for investors to engage with and monitor their investments.

The goal of representing the lending contracts as digital certificates (NFTs) is to also allow users to claim or redeem rights. The objective of the NFT contract is to enhance transparency and convenience by representing TokTok's lending contracts through NFTs.

4. Risk Classification

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

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** - almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** - still relatively likely, although only conditionally possible
- **Low** - requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review was conducted over the span of 5 days by the core Shieldify team. The code is written professionally, and all best practices are considered. However, the security review identified several Medium-severity issues related to minting functionalities, signature reuse, potential interest loss when claiming and a single Low-severity issue regarding the return value of `claimAll()`
.

Shieldify extends its gratitude to Harvest Flow for cooperating with all the questions our team had and strictly following the recommendations provided.

5.1 Protocol Summary

Project Name	**Harvest Flow V2**
Repository	bHARVESTFLOW Ver.2
Type of Project	NFT Lending Protocol
Audit Timeline	5 days
Review Commit Hash	05f0814caa51dcb034dd01f610315d4c6efedce8
Fixes Review Commit Hash	4b49c68e43d3124f626857fd2df6f2c77d48ac03

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
contracts/src/LendingNFT.sol	411
contracts/src/NftFactory.sol	88
Total	**499**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Medium** issues: **3**
- **Low** issues: **1**

ID	Title	Severity	Status
[M-01]	The `mint()` Functions in `NftFactory` Do Not Function Properly	Medium	Fixed
[M-02]	The Signature of `preMint()` Can Be Used in Different NFT Contracts	Medium	Fixed
[M-03]	Users May Lose the Interest They Should Claim When Transferring NFTs	Medium	Fixed
[L-01]	The `claimAll()` Function Does Not Return the Total Amount of Interest Claimed	Low	Fixed

7. Findings

[M-01] The `mint()` Functions in `NftFactory` Do Not Function Properly

Severity

Medium Risk

Description

The `publicMint()` and `preMint()` in `NftFactory.sol` will call the corresponding functions in the NFT contract:

```
function publicMint(address nft, uint256 mintAmount) external returns (
    uint256) {
     uint256 tokenId = LendingNFT(nft).publicMint(mintAmount);
     return tokenId;
}

function preMint(address nft, uint256 mintAmount, uint256 maxMintAmount,
    bytes memory signature)
     external
     returns (uint256)
{
     uint256 tokenId = LendingNFT(nft).preMint(mintAmount, maxMintAmount,
        signature);
     return tokenId;
}
```

In the `LendingNFT.sol` contract, both `publicMint()` and `preMint()` will perform transfers from `msg.sender` through `safeTransferFrom()`.

The issue here is that the `LendingNFT.sol` contract is not approved to transfer payment tokens before calling the minting function through `NftFactory()`, meaning that `safeTransferFrom()` cannot retrieve assets from the `NftFactory.sol` contract. Therefore, calling `NftFactory.publicMint()` / `preMint()` will revert.

Location of Affected Code

File: NftFactory.sol#L59-L76

Impact

The mint functions in `NFTFactory.sol` do not function properly.

Recommendation

In the `publicMint()` and `preMint()` of `NftFactory.sol`, first, obtain assets from `msg.sender`, then approve them to the `LendingNFT.sol` contract, and finally call `LendingNFT.publicMint()` / `preMint()`.

Team Response

Fixed.

[M-02] The Signature of `preMint()` Can Be Used in Different NFT Contracts

Severity

Medium Risk

Description

In `preMint()`, the signature is verified through `_verifyAddressSigner()`, with the parameters of the signature being `msg.sender` and `maxMintAmount`:

```
/// @notice Verify the `signature` is a hash of `msg.sender` and `
   maxMintAmount`, signed by the `signerAddress`.
function _verifyAddressSigner(bytes memory signature, uint256
   maxMintAmount) internal view {
    bytes32 hash = keccak256(abi.encodePacked(msg.sender, maxMintAmount))
        ;
    if (hash.recover(signature) != signerAddress) {
        revert InvalidSignature();
    }
}
```

The problem here lies in the lack of the NFT address in the signature parameters. Since multiple NFTs could exist in this protocol, the signature might be replayed across other NFT contracts.

For example:
1. Alice, as the owner and signer, created two NFTs.
2. She wanted Bob to participate in the `preMint()` of NFT1, so she generated a signature for Bob.
3. Since the signature does not include the NFT address, Bob can use the same signature to participate in the `preMint()` of NFT2 as well, which does not meet Alice's expectations.

Location of Affected Code

File: LendingNFT.sol#L601-L607

Impact

The signature of `preMint()` can be used in different NFT contracts, which may not meet the signer's expectations.

Recommendation

Add the contract address in the signature's parameter.

Team Response

Fixed.

[M-03] Users May Lose the Interest They Should Claim When Transferring NFTs

Severity

Medium Risk

Description

In `LendingNFT`, `claim()` is not called in the NFT's transfer hook.

This means that the contract cannot guarantee that `claim()` and transfer of the NFT occur in the same block. If users want to ensure that claiming and transferring an NFT happen in the same block and that claiming occurs before transferring, they can only achieve this by creating a contract themselves to call it or creating a bundle through Flashbots, which is obviously highly difficult for ordinary users.

Before maturity, interest will increase with the increase of `block.timestamp`.

```
// Proportion of time interval from the beginning of the lending period
   until now to a year, scaled to the 1e18
uint256 proportionOfIntervalTotalScaled = ((Math.min(block.timestamp,
   maturity) - lendingAt) * 1e18) / year;
```

If the transfer of the NFT occurs in a block after the `claim()`, the interest generated during this period cannot be obtained by the previous owner.

For example:
1. Before maturity, Alice wants to transfer her NFT to Bob.
2. Alice first executes the `claim()` and then transfers the NFT.
3. Due to this vulnerability, Alice's two transactions may end up in different blocks, assuming they are separated by one block.
4. Alice cannot receive the interest for that one block, even though it rightfully belongs to her, as she still owned the NFT during this period.
5. Ultimately, the interest in that block, which should have belonged to Alice, ends up belonging to Bob.

Location of Affected Code

File: LendingNFT.sol#L622

Impact

Before maturity, if users want to transfer NFTs, they may lose the interest they are entitled to. To avoid this, users must either create their own contracts or use services like Flashbots, which is challenging for regular users.

Recommendation

Call `claim()` in NFT's transfer hook

Team Response

Fixed.

[L-01] The `claimAll()` Function Does Not Return the Total Amount of Interest Claimed

Severity

Low Risk

Description

The comment for `claimAll()` states that the return value is the `Total amount of interest claimed` , but from the code, it can be seen that if the owners of these tokens are not the same, the return value is only the amount claimed by the last owner, not the total amount claimed throughout the process:

```
/// @return uint256 Total amount of interest claimed
function claimAll(uint256[] memory tokenIds) public whenNotPaused returns
    (uint256) {
    // code
    for (uint256 i; i < tokenIdsLength;) {
        uint256 tokenId = tokenIds.unsafeMemoryAccess(i);
        ...
        address owner = ownerOf(tokenId);
        if (owner != lastOwner) {
            totalClaimed += claimableInterestCurrentOwner;
            payableToken.safeTransfer(lastOwner,
                claimableInterestCurrentOwner);
            claimableInterestCurrentOwner = 0; //@audit reset when owner
                change
        }
```

```
            lastOwner = owner;
            uint256 claimableInterest =
                Math.min(payableToken.balanceOf(address(this)),
                   accruedInterest - claimed[tokenId]);
            claimed[tokenId] += claimableInterest;
            claimableInterestCurrentOwner += claimableInterest;
            emit Claimed(owner, tokenId, claimableInterest);
            unchecked {
                ++i;
            }
        }
        totalClaimed += claimableInterestCurrentOwner;
        payableToken.safeTransfer(lastOwner, claimableInterestCurrentOwner);

        return claimableInterestCurrentOwner;
    }
```

Location of Affected Code

File: LendingNFT.sol#L287-L324

Impact

The `claimAll()` does not return the total amount of interest claimed as the comment states, as well as `redeemAll()` which uses `claimAll()`'s return.

Recommendation

Use the increment of `totalClaimed` as the return value.

Team Response

Fixed.

, our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Thank you!