shieldify

Wawa

SECURITY REVIEW

Date: 15 Aug 2023

CONTENTS

1. About Shieldify

We are Shieldify Security – a company on a mission to make web3 protocols more secure, cost-efficient and user-friendly. Our team boasts extensive experience in the web3 space as both smart contract auditors and developers that have worked on top 100 blockchain projects with multi-million dollars in market capitalization.

Book an audit and learn more about us at shieldify.org or @ShieldifySec

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 Wawa

Wawa is an NFT collection like no other — with NFTs generated based on web3 community members' wallet activity. On-chain identities come to life as unique and hyper-cute characters, with each attribute determined by factors such as trade volume on Uniswap and total gas spent. From top to toe, and even pets, users' web3 histories become the DNA of their pixel-perfect, playable avatars.

The artwork of Wawa is entrusted to your hands to expand its possibilities of expression. They can be freely used as material for derivative works such as films, animations, comics and games. For instance, you could craft a story with Wawa characters as the protagonists, or construct a new video game world using them.

Learn more about Wawa's concept here.

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 or even gas optimization techniques

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. Audit Summary

The audit duration lasted 4 days and a total of 96 hours have been spent by the three auditors – @ShieldifyMartin, @ShieldifyAnon and @ShieldifyGhost. This is the first audit for this specific part of the Phi protocol.

Overall, the codebase is well-written with the implementation of numerous good practices. This is the second audit that Shieldify has prepared for Phi and we are very happy to see that the recommendations, suggested in the first audit have been implemented in this codebase as well.

The report contains findings around the possibility of NFTs with duplicate token URIs and loss of funds as a result of a user's mistake, together with some informational and gas-optimization recommendations. The documentation has been scarce, but this has been balanced out by frequent communication with the Phi Team, who gave us a comprehensive guide over the codebase. The test coverage is very good and comprehensive.

We would also like to thank the Phi team for being very responsive and for providing clarifications and detailed responses to all of our questions. They are an amazing project and Shieldify is happy to be part of it.

5.1 Protocol Summary

Project Name	**Wawa**
Repository	Wawa
Type of Project	NFT collection
Audit Timeline	4 days
Review Commit Hash	5722f44b2115e452238294253f3828702f27323c
Fixes Review Commit Hash	775989cf97ce745823853a92e543ec586409abb1

5.2 Scope

The following smart contracts were in the scope of the audit:

File	nSLOC
src/interfaces/IWawaNFT.sol	4
src/GetWawa.sol	72
src/WawaNFT.sol	100
src/utils/MultiOwner.sol	25
src/types/Wawa.sol	20
Total	**221**

6. Findings Summary

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

- **Critical** and **High** issues: **0**
- **Medium** issues: **2**
- **Low** issues: **1**

ID	Title	Severity
[M-01]	Risk of Having Two `Wawa NFT(Avatar)` NFTs Being Assigned Identical `TokenURI`	Medium
[M-02]	The `getWawa()` Function Does Not Return Excess Fees for `Wawa NFT(Avatar)` Claim	Medium
[L-01]	Use `_safeMint()` When Minting A New `Wawa NFT(Avatar)`	Low

7. Findings

[M-01] Risk of Having Two `Wawa NFT(Avatar)` NFTs Being Assigned Identical `TokenURI`

Severity

Medium Risk

Description

The `SetTokenURI()` function in `WawaNFT.sol` contract is used to assign a `tokenURI` to the `Wawa NFT(Avatar)` that is to be minted, upon calling the `GetWawa.claimWawa()` function which internally calls the `WawaNFT.getWawa()` function. Each `Wawa NFT(Avatar)` must have a unique `tokenURI`, but there is no check for passing an already-used `tokenURI`.

The impact is this break the initial promise of the protocol that each user will get a unique `Wawa NFT(Avatar)`.

The following scenario can happen:

1. Alice wants to claim/mint a new unique `Wawa NFT(Avatar)`.
2. Alice calls the `GetWawa.claimWawa()` function and she gets a new unique `Wawa NFT(Avatar)`.
3. Bob, a malicious user sees the successful transaction of Alice and copies her `tokenURI` and then calls the `GetWawa.claimWawa()` function with Alice's `tokenURI`.
4. Alice and Bob have one `Wawa NFT(Avatar)` token each with the same `metadata`.

Location of Affected Code

File: src/WawaNFT.sol#L94-L97

```
function setTokenURI(uint256 tokenId, string memory tokenURI) public
    virtual onlyOwner {
     allWawa[tokenId].tokenURI = tokenURI;
     emit SetTokenURI(tokenId, tokenURI);
}
```

Recommendation

Implement a check if the `tokenURI` that will be assigned when calling the `setTokenURI()` function has already been used for the minting of another unique `Wawa NFT(Avatar)`. We recommend the implementation of a `mapping` that will indicate if the `tokenURI` that is passed has been used before.

Example:

```
+ mapping(string tokenURI => bool) public createdTokenURI;
+ Error TokenURIAlreadyUsed();

function setTokenURI(uint256 tokenId, string memory tokenURI) public
    virtual onlyOwner {
+ if(createdTokenURI[tokenURI]) revert TokenURIAlreadyUsed();

+ createdTokenURI[TokenURI] = true;
  allWawa[tokenId].tokenURI = tokenURI;

  emit SetTokenURI(tokenId, tokenURI);
}
```

Team Response

Acknowledged and fixed by implementing `mapping` and adding a check if `tokenURI` already exists.

[M-02] The `getWawa()` Function Does Not Return Excess Fees for `Wawa NFT(Avatar)` Claim

Severity

Medium Risk

Description

In the `getWawa()` function, the contract `WawaNFT.sol` requires the users to pay a `fee(price)` to claim/mint new `Wawa NFT(Avatar)` tokens.

However, it does not handle the situation where the user pays more `msg.value` than the required `price` for this `Wawa NFT(Avatar)` token. Currently, if `msg.value > price`, the contract simply accepts the payment without refunding the difference to the user.

The impact is primarily financial, as the current implementation does not return any excess ether `msg.value` paid beyond the required `price = 0.05 ether`. Users who pay more than the required `price` will not get their payment back, resulting in those funds being locked in the `WawaNFT.sol` contract forever.

The following scenario can happen:

1. Alice wants to claim/mint a new unique `Wawa NFT(Avatar)`.
2. Alice calls the `GetWawa.claimWawa()` function which internally calls the `WawaNFT.getWawa()` function and she sends `0.09 ether`, but the price of `Wawa NFT(Avatar)` is `0.05 ether`.
3. The price fee is correctly charged at `0.05 ether` and Alice successfully gets her `Wawa NFT(Avatar)`, but she is not refunded the excess of `0.04 ether`.
4. The excess of `0.04 ether` price fee is locked forever in the `WawaNFT.sol` contract balance.

Location of Affected Code

File: src/WawaNFT.sol#L153-L174

```
function getWawa(
  address to,
  uint256 tokenId,
  string memory tokenURI,
  Faction faction,
  Trait memory trait,
  uint8 petId,
  bytes32 gene
)
  external
  payable
  onlyOwner
  nonReentrant
{
    // check if the function caller is not an zero account address
    if (to == address(0)) revert ZeroAddressNotAllowed();

    // check token is not created
    if (created[tokenId]) revert TokenAlreadyCreated(tokenId);

    // price sent in to buy should be equal to or more than the token's
       price
    if (msg.value < price) revert InsufficientAmountSent();
    .
    .
    .
}
```

Recommendation

To address this vulnerability, we will propose two solutions:

1. Change the check mark for `Wawa NFT(Avatar)` price to `if (msg.value != price)`, if `msg.value` is different from `price = 0.05 ether` the transaction should be reverted:

In this way, you guarantee the user that he will not lose his funds if he accidentally sends a `msg.value` greater than `0.05 ether` because the transaction will revert.

Example:

```
- if (msg.value < price) revert InsufficientAmountSent();
+ if (msg.value != price) revert  heValueDoesNotMatch();
```

2. Return the difference to the user via a call function:

In this way, you will return the difference to the user every time he sends a `msg.value` greater than the `price = 0.05 ether`.

Example:

```
+ uint256 excessPrice = msg.value - price;

+ (bool success, ) = address(to).call{ value: excessPrice }("");
+ if (!success) revert FailedPaymentToUser();
```

Team Response

Acknowledged and fixed by implementing the first suggestion to change the check for the `price` -> `if (msg.value != price)`.

[L-01] Use `_safeMint()` When Minting A New `Wawa NFT(Avatar)`

Severity

Low Risk

Description

The `getWawa()` function in `WawaNFT.sol` contract use the `super._mint()` method of `ERC721` contract which is missing a check if the recipient is a smart contract that can actually handle `ERC721` tokens. If the case is that the recipient can not handle `ERC721` tokens then they will be stuck forever. For this particular problem, the `safe` methods were added to the `ERC721` standard.

Location of Affected Code

File: src/WawaNFT.sol#L186

```
super._mint(to, tokenId);
```

Recommendation

Prefer using `_safeMint()` function `(already included in the imported OpenZeppelin ERC721 contract)` instead of `_mint()` function for `ERC721` tokens.

Example:

```
-   super._mint(to, tokenId);
+   super._safeMint(to, tokenId);
```

Team Response

Acknowledged and fixed by replacing with `_safeMint()` function.

shieldify

Thank you!