



# Wormhole NFT Aptos Audit

---

Presented by:

**OtterSec**

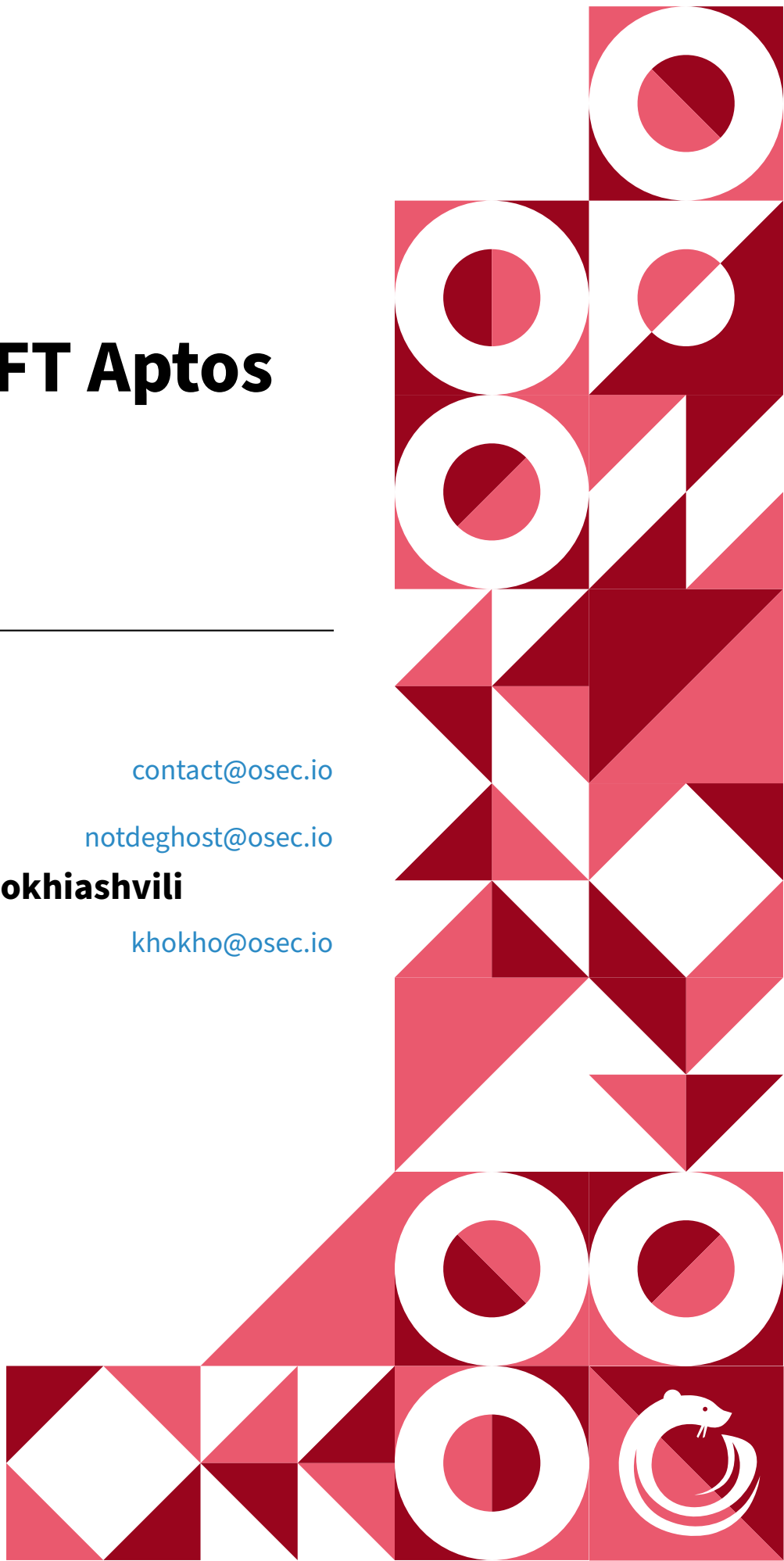
**Robert Chen**

**Aleksandre Khokhiashvili**

[contact@osec.io](mailto:contact@osec.io)

[notdeghost@osec.io](mailto:notdeghost@osec.io)

[khokho@osec.io](mailto:khokho@osec.io)



# Contents

<b>01 Executive Summary</b>	<b>2</b>
Overview . . . . .	2
Key Findings . . . . .	2
<b>02 Scope</b>	<b>3</b>
<b>03 Findings</b>	<b>4</b>
<b>04 General Findings</b>	<b>5</b>
OS-WMH-SUG-00   Non UTF-8 Token Name Mutation . . . . .	6
OS-WMH-SUG-01   Semi-Fungible Token Support . . . . .	7
OS-WMH-SUG-02   Misleading URI Constructor Documentation . . . . .	8
 <b>Appendices</b>	
<b>A Vulnerability Rating Scale</b>	<b>9</b>
<b>B Procedure</b>	<b>10</b>

# 01 | Executive Summary

## Overview

Wormhole engaged OtterSec to perform an assessment of the `nft_bridge` program. This assessment was conducted between January 24th and February 8th, 2023. For more information on our auditing methodology, see [Appendix B](#).

Issues were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team to streamline patches and confirm remediation. We delivered final confirmation of the patches February 9th, 2023.

## Key Findings

Over the course of this audit engagement, we produced 3 findings total.

In particular, we found a quirk related to the parsing of non UTF-8 token names, which results in name mutation on roundtrip ([OS-WMH-SUG-00](#)). We also noted a discrepancy in the documentation for URI parsing ([OS-WMH-SUG-02](#)).

Overall, we commend the Wormhole team for their secure implementation around the critical attack surface of the program and for their responsiveness and knowledge demonstrated throughout our audit process.

## 02 | Scope

The source code was delivered to us in a git repository at [github.com/wormhole-foundation/wormhole](https://github.com/wormhole-foundation/wormhole). This audit was performed against commit e30e762.

A brief description of the programs is as follows.

Name	Description
nft_bridge	The nft_bridge contract is used for bridging NFTs between different blockchains and Aptos. NFTs are transferred from one chain to another using a "lockup native & mint wrapped" and "burn wrapped & unlock native" mechanism.

The functionality of the NFT bridge includes:

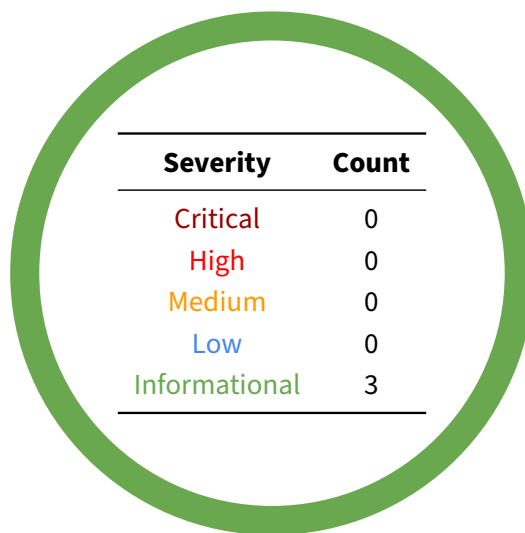
- Locking Aptos NFTs and publishing a VAA that can be used to mint wrapped NFTs on the target chain.
- Receiving transfer VAAs from other chains and creating a wrapped representation for foreign NFTs.
- Burning wrapped NFTs and publishing a transfer VAA to send them back to the source (or possibly a different) chain.
- Receiving transfer VAAs generated by burning wrapped Aptos NFTs on other chains and unlocking native NFTs in response.

As part of this audit, we also provided proofs of concept for each vulnerability to prove exploitability and enable simple regression testing. These patches can be found at [github.com/otter-sec/wormhole-nft-aptos-pocs](https://github.com/otter-sec/wormhole-nft-aptos-pocs).

## 03 | Findings

Overall, we reported 3 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	0
Low	0
Informational	3

## 04 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-WMH-SUG-00	Cross-chain transfer's token name may change on roundtrip
OS-WMH-SUG-01	Only one token of <code>property_version</code> zero can be supported at a time
OS-WMH-SUG-02	The documentation for the URI constructors does not match the implementation

## OS-WMH-SUG-00 | Non UTF-8 Token Name Mutation

### Description

If the incoming `Transfer.name`'s bytes do not represent a valid UTF-8 string, `string32::deserialize` will remove the last few bytes until it becomes valid. This will cause the corresponding wrapped token to be stored with a modified name. Accordingly, when the same wrapped token gets burned and sent back to the original chain, the outgoing `Transfer.name` will be set to the modified name.

*string32.move*

RUST

```
fun take(bytes: vector<u8>, n: u64): String {
    while (vector::length(&bytes) > n) {
        vector::pop_back(&mut bytes);
    };

    let utf8 = string::try_utf8(bytes);
    while (option::is_none(&utf8)) {
        vector::pop_back(&mut bytes);
        utf8 = string::try_utf8(bytes);
    };
    option::extract(&mut utf8)
}
```

The Wormhole [documentation](#) mentions that non UTF-8 strings are expected and removing bad bytes off of them for presentation is accepted behavior. However, the documentation does not seem to provide any commentary on mutating the name field after unwrapping a wrapped token. This could pose a problem if the NFT bridge implementation on the token's source chain uses name for token identification.

### Remediation

Explicitly store the name field as a byte vector to prevent mutation on roundtrip for non utf-8 names.

## OS-WMH-SUG-01 | Semi-Fungible Token Support

### Description

The current NFT Bridge explicitly does not support semi-fungible tokens.

*sources/transfer\_nft.move*

RUST

```
// Instead, we just ensure that the NFT bridge can hold a maximum of  
// 1 copy of this token. That is, even if the token has a supply  
// larger than 1, only 1 of the tokens can be bridged out at any  
// given time, so they effectively behave as NFTs outside of Aptos,  
// even when they're technically fungible on aptos.  
assert!(token::balance_of(@nft_bridge, token_id) == 1,  
↳ E_FUNGIBLE_TOKEN);
```

After discussion with ecosystem teams, it appears that there are some usecases. For example, we encountered one team which was building a ticketing platform where individual tickets would be fungible across sections, but each section would be represented as an individual NFT.

### Remediation

Consider adding explicit support for semi-fungible tokens. For example, Wormhole could implement a nonce to make semi-fungible tokens fully non-fungible.



## OS-WMH-SUG-02 | Misleading URI Constructor Documentation

### Description

The documentation for the `from_string` and `from_bytes` functions states that they will never abort. However, if the input URI is longer than the `MAX_LENGTH` bytes, these functions will in fact abort.

*sources/newtypes/uri.move*

RUST

```
/// Truncates a string to a URI.  
/// Does not abort.  
public fun from_string(s: &String): URI {  
    from_bytes(*string::bytes(s))  
}  
  
/// Truncates a byte vector to a URI.  
/// Does not abort.  
public fun from_bytes(b: vector<u8>): URI {  
    assert!(vector::length(&b) <= MAX_LENGTH, E_URI_TOO_LONG);  
    URI { string: string::utf8(b) }  
}
```

### Remediation

Consider clarifying the intention of these functions.

Either the Wormhole Aptos token bridge does not support sending NFTs with a URI of length greater than 200 or the URI parsing should manually truncate the URIs.

# A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the [General Findings](#) section.

---

<b>Critical</b>	<p>Vulnerabilities that immediately lead to loss of user funds with minimal preconditions</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Misconfigured authority or access control validation</li><li>• Improperly designed economic incentives leading to loss of funds</li></ul>
<b>High</b>	<p>Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Loss of funds requiring specific victim interactions</li><li>• Exploitation involving high capital requirement with respect to payout</li></ul>
<b>Medium</b>	<p>Vulnerabilities that could lead to denial of service scenarios or degraded usability.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Malicious input that causes computational limit exhaustion</li><li>• Forced exceptions in normal user flow</li></ul>
<b>Low</b>	<p>Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Oracle manipulation with large capital requirements and multiple transactions</li></ul>
<b>Informational</b>	<p>Best practices to mitigate future security risks. These are classified as general findings.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Explicit assertion of critical internal invariants</li><li>• Improved input validation</li></ul>

---

## B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.