# Coyote (HOWL)
# Audit Report

**Version Audit: 1.0.0**

**Presented by DexCoyote.com**

**November 5, 2022**

## 01. Introduction

This document includes the results of the audit on the AstroHodler (STAR) Token.

Audit Start Time:

November 3, 2022

Audit End Time:

November 5, 2022

Token's Name:

Coyote

Token's Symbol:

HOWL

Token's Precisions:

18

Token's Address:

0xa2B305CcF61733b10f125ae93671bB6b1793531e

Audited Source File's Address:

https://bscscan.com/token/0xa2B305CcF61733b10f125ae93671bB6b1793531e

The goal of this audit is to review Coyote token issurance function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the AstroHodler team for specified versions. Whenever the code, software, materials, settings, enviroment etc is changed, the comments of this audit will no longer apply.

## — Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

## — Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The auditing process follows a routine series of steps:

1. Code review that includes the following

i. Review of the specifications, sources, and instructions provided to make sure we understand the size, scope, and functionality of the project's source code.

ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.

iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.

2. Testing and automated analysis that includes the following:

i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.

ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.

3. Best practices review, which is a review of the source code to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

## — Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

## — Documentation

For this audit, we used the following source of truth about how the token issurance should work:

https://dexcoyote.com/

This was considered the specification.

## — Comments from Auditor

No vulnerabilities with critical, high, medium or low-severity were found in the above source code.

Additional notice: 0.

## 02. Major functions of audited code

The audited code implements a token issurance function. Here are the details:

Name: Coyote

Symbol: HOWL

Precisions: 18

Max Supply: No upper limit

Flexible Token Supply: the token supply can be increased

Transaction Fee: 0 for now. Can't be changed.

Other functions:

A blacklist: blacklisted addresses cannot transfer tokens
A mint: owner can mint new tokens

## 03. Admin rights

The Admin has the following access rights:

Editing blacklist: the Admin can add/remove addresses to/from the blacklist
Increasing token supply

## 04. Key points in audit

**During the audit we reviewed possible vulnerabilities in token issurance.**

**- Integer Overflow/Underflow**
**We checked all the code sections, which had arithmetic operations and might introduce integer overflow or underflow if no safe libraries were used. All of them used safe libraries.**

**We didn't find issues or risks in these functions or areas at the time of writing.**

**- Access Control**
**We checked each of the functions that could modify a state, especially those functions that could only be accessed by "owner".**

**We didn't find issues or risks in these functions or areas at the time of writing.**

**- Admin Rights**
**We checked whether or not the Admin had potentially risky rights and whether or not the potentially risky rights had been transferred to multi-sig wallets.**

**We didn't find issues or risks in these functions or areas at the time of writing.**

- State Update

We checked some key state variables which should only be set at initialization.

We didn't find issues or risks in these functions or areas at the time of writing.

## 05. Coverage of issues

The issues that the audit covered when conducting the audit include
but are not limited to the following ones:

Re-entrancy Attack
DDos Attack
Integer Overflow
Function Visibility
Logic Vulnerability
Uninitialized Storage Pointer
Arithmetic Precision

Tx.origin
Shadow Variable
Design Vulnerability
Token Issurance
Asset Security
Access Control

## 06. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

## 07. List of issues by severity

A. Critical
- N/A

B. High
- N/A

C. Medium
- N/A

D. Low
- N/A

## 08. List of issues by source file

- N/A

## 09. Issue descriptions

- N/A

## 10. Recommendations to enhance the overall security

**We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.**

**- Consider Using A Safe Library to Transfer and Approve**

The transfer , transferFrom and approve functions don't return bool values. Therefore when each of them is called its caller needs to handle its final state, otherwise the transaction may fail. Consider using safe functions in the TransferHelper library as follows:

```
// helper methods for interacting with ERC20 tokens and sending ETH that do not
consistently return true/false
library TransferHelper {
   function safeApprove(address token, address to, uint value) internal {
      // bytes4(keccak256(bytes('approve(address,uint256)')));
      (bool success, bytes memory data) =
token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
      require(success && (data.length == 0 || abi.decode(data, (bool))),
'TransferHelper: APPROVE_FAILED');
   }

   function safeTransfer(address token, address to, uint value) internal {
      // bytes4(keccak256(bytes('transfer(address,uint256)')));
      (bool success, bytes memory data) =
token.call(abi.encodeWithSelector(0xa9059cbb, to, value));
      require(success && (data.length == 0 || abi.decode(data, (bool))),
'TransferHelper: TRANSFER_FAILED');
   }

   function safeTransferFrom(address token, address from, address to, uint value)
internal {
      // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
      (bool success, bytes memory data) =
token.call(abi.encodeWithSelector(0x23b872dd, from, to, value));
      require(success && (data.length == 0 || abi.decode(data, (bool))),
'TransferHelper: TRANSFER_FROM_FAILED');
   }
}
```