Zellic

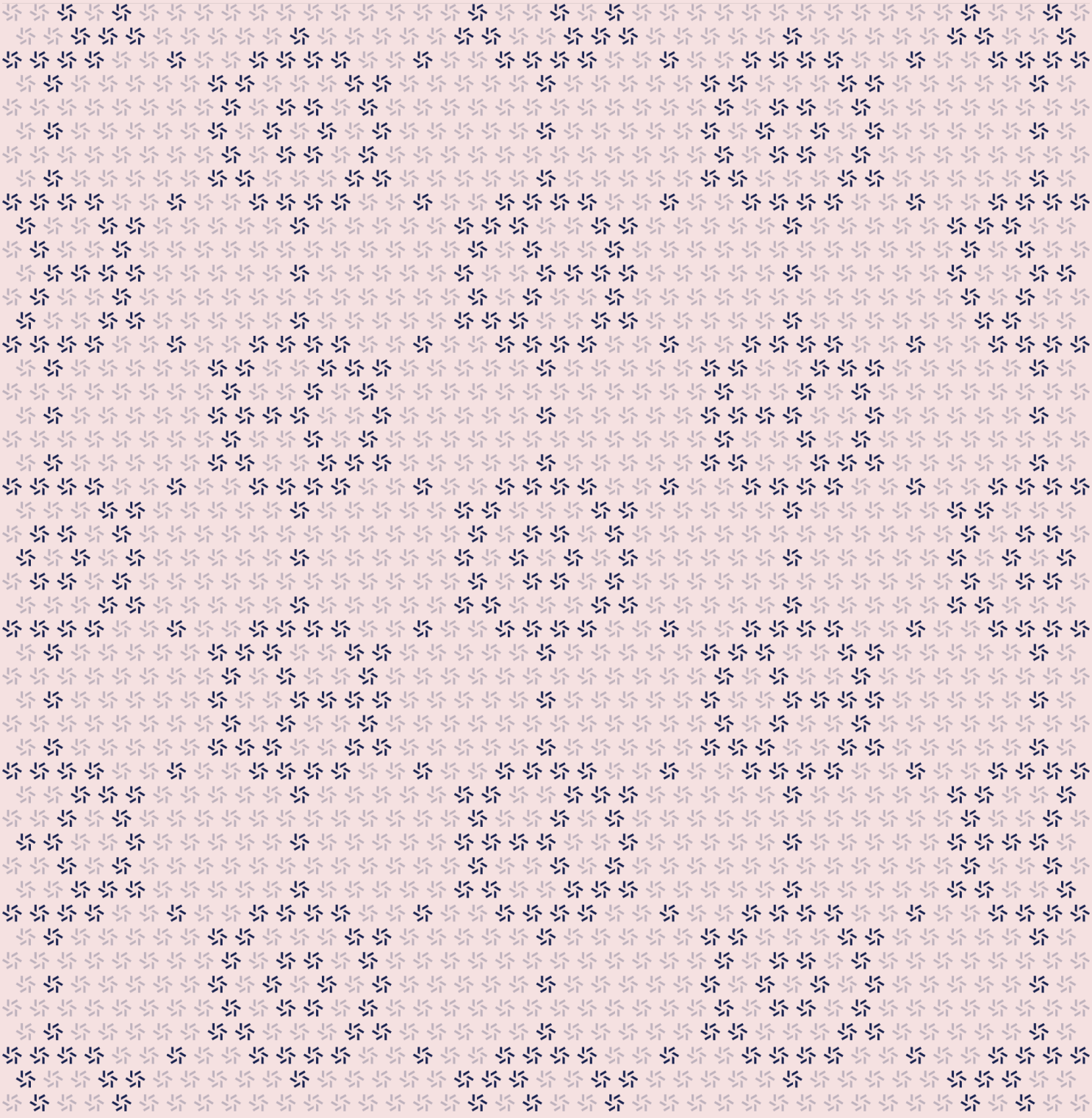
Prepared for
Mason
Awaken Finance

Prepared by
Jasraj Bedi
Kuilin Li
Zellic

May 22, 2024

Awaken Swap

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Awaken Finance from April 29th to May 13th, 2024. During this engagement, Zellic reviewed Awaken Swap's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Are there any ways to break underlying economic assumptions when doing swaps or adding/removing liquidity operations?
- Is the token contract for the LP tokens implemented correctly according to commonsense token mint/transfer/burn semantics?
- Are there any DeFi vulnerabilities or logic vulnerabilities in the reviewed code?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody
- Security guarantees made by the underlying AElf chain

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Awaken Swap contracts, we discovered eight findings. No critical issues were found. Three findings were of high impact, two were of medium impact, one was of low impact, and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Awaken Finance's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	3
Medium	2
Low	1
Informational	2

2. Introduction

2.1. About Awaken Swap

Awaken Finance contributed the following description of Awaken Swap:

> AwakenSwap is a decentralized exchange (DEX) based on the Automated Market Maker (AMM) algorithm. Thriving on aelf chain, AwakenSwap supports swapping between two arbitrary tokens.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect

its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Awaken Swap Contracts

Repository	https://github.com/Awaken-Finance/awaken-swap-contract ↗
Version	awaken-swap-contract: `1d8a152d7805a0fd65f3616a53a72acc19a556ed`
Programs	• AwakenSwapContract • AwakenSwapContract_Admin • AwakenSwapContract_Constants • AwakenSwapContract_Helpers • AwakenSwapContract_LP • AwakenSwapContract_Swap • AwakenSwapContract_Views • AwakenSwapContractReferenceState • AwakenSwapContractState • TokenContract • TokenContract_ACS2 • TokenContract_Actions • TokenContract_Helpers • TokenContract_Views • TokenContractConstants • TokenContractReferenceState • TokenContractState
Type	C#
Platform	AElf

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of three person-weeks. The assessment was conducted over the course of two calendar weeks.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Jasraj Bedi
Co-Founder
jazzy@zellic.io

Kuilin Li
Engineer
kuilin@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

April 29, 2024	Kick-off call
April 29, 2024	Start of primary review period
May 13, 2024	End of primary review period

3. Detailed Findings

3.1. The supply limit limits issue, not supply

Target	TokenContractActions		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

In the state for the token contract, for each token symbol, the `Issued` property tracks the total tokens issued, which does not decrease when tokens are burned. On the other hand, the `Supply` property tracks the total circulating supply, which does decrease when tokens are burned.

The `TotalSupply` constant property is, based on its name, supposed to be a limit to the amount of supply that can be outstanding. However, see the `Issue` function:

```
public override Empty Issue(IssueInput input)
{
    // [...]
    tokenInfo.Issued = tokenInfo.Issued.Add(input.Amount);
    tokenInfo.Supply = tokenInfo.Supply.Add(input.Amount);

    Assert(tokenInfo.Issued <= tokenInfo.TotalSupply, "Total supply
    exceeded.");

    // [...]
}
```

Instead of being a limit on total supply, it is implemented as a limit on the total amount issued. This means that burned tokens do not move the contract away from this limit.

Impact

Ultimately, this means that the LP tokens will eventually not be mintable due to the total issued token count hitting the supply cap from normal minting and burning.

Recommendations

Compare `Supply` against `TotalSupply` instead of `Issued`.

Alternatively, if this is meant to be a cap on total issued quantity, the property should be renamed to

`TotalIssued` and logic should be added to increase this constant once the limit is reached.

Remediation

Awaken Finance has chosen to disable enforcement of these limits in `1eeef4bf` ↗.

3.2. Anyone can create tokens before initialization

Target	TokenContractActions		
Category	Coding Mistakes	**Severity**	High
Likelihood	Medium	**Impact**	High

Description

Normally, only a trusted minter whose address is in the `MinterMap` is allowed to call `Create` to create new tokens. However, in the function implementation, this is only checked if the owner is set:

```
public override Empty Create(CreateInput input)
{
    if (State.Owner.Value != null)
    {
        AssertSenderIsMinter();
    }

    // [...]
}
```

This means that if the contract is deployed but not initialized, this check is skipped, and anyone can create tokens with any name.

Impact

Quick users can create tokens with symbols that the swap contract would create as LP tokens by sniping them during the deployment process. If they succeed, they will control the token instead of the swap contract.

Recommendations

Remove the conditional and always check `AssertSenderIsMinter`. This check will always return false if the contract is uninitialized, since the `MinterMap` is empty before initialization.

Remediation

This issue has been acknowledged by Awaken Finance, and a fix was implemented in commit `1eeef4bf` ↗.

3.3. The `AssertContractInitialized` function should check `Initialized`

Target	AwakenSwapContractHelpers		
Category	Business Logic	**Severity**	High
Likelihood	Low	**Impact**	High

Description

The helper function `AssertContractInitialized` is used to ensure that the contract is initialized prior to the execution of some actions:

```
private void AssertContractInitialized()
{
    Assert(State.Admin.Value != null, "Contract not initialized.");
}
```

However, instead of checking `State.Initialized`, it checks whether the admin is null. Although the admin is indeed null when the contract is uninitialized, after initialization, the admin can call `ChangeOwner` to set the admin property to null in order to renounce ownership.

Impact

If the admin renounces ownership by calling `ChangeOwner` to change the admin to null, then `AssertContractInitialized` will erroneously revert.

Recommendations

Compare `Supply` against `TotalSupply` instead of `Issued`.

Alternatively, if this is meant to be a cap on total issued quantity, the property should be renamed to `TotalIssued` and logic should be added to increase this constant once the limit is reached.

Remediation

This issue has been acknowledged by Awaken Finance, and a fix was implemented in commit `1eeef4bf`↗.

3.4. Calling `ChangeOwner` may lock ownership upon user error

Target	AwakenSwapContractAdmin		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

In the swap contract, the current owner can call `ChangeOwner` to change the owner to a new owner; however, the `input` is not checked at all:

```
public override Empty ChangeOwner(Address input)
{
    AssertSenderIsAdmin();
    State.Admin.Value = input;
    return new Empty();
}
```

Impact

If the input contains an incorrect address or the address of a contract incapable of calling `ChangeOwner`, ownership may become locked.

Recommendations

We recommend implementing a two-step admin-transfer method, where the current admin can arbitrarily set a `ProposedAdmin`, and then the named `ProposedAdmin` can call another method to accept ownership.

While this does not completely prevent the possibility of ownership being locked, it decreases the likelihood of human error that would otherwise switch the `State.Admin.Value` to an address incapable of administrating the swap contract.

Remediation

In commit `1eeef4bf` ↗, Awaken Finance improved this function by ensuring that the owner was not accidentally set to null, but they did not implement a two-step admin transfer. We recommend implementing a two-step admin-transfer method to prevent the possibility of ownership being locked, but the risk has been reduced by this change.

3.5. No guard on admin-set fee rate

Target	AwakenSwapContractAdmin		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

In the swap contract, the admin can call `SetFeeRate` to set a new fee rate:

```
public override Empty SetFeeRate(Int64Value input)
{
    AssertSenderIsAdmin();
    State.FeeRate.Value = input.Value;
    return new Empty();
}
```

However, there is no check on the input.

Impact

If the new fee rate is above `FeeRateMax`, then invariants are broken, since the charged fee would be larger than 100%. Similarly, if the fee rate is below zero, then other unintended and unchecked behavior may occur.

Recommendations

This admin function should have a guard on the range of the input.

Remediation

This issue has been acknowledged by Awaken Finance, and a fix was implemented in commit `1eeef4bf` ↗.

3.6. Deadline-enforcement unit is inconsistent

Target	AwakenSwapContractSwap and AwakenSwapContractLP		
Category	Business Logic	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

In various input protobufs, the `Deadline` parameter ensures that a user's transaction, if not executed by the deadline, is canceled. However, the enforcement of the deadline is inconsistent:

```
// AddLiquidity
Assert(input.Deadline >= Context.CurrentBlockTime, "Expired.");

// RemoveLiquidity
Assert(input.Deadline.Seconds >= Context.CurrentBlockTime.Seconds, "Expired");

// SwapExactTokensForTokens
Assert(input.Deadline >= Context.CurrentBlockTime, "Expired");

// SwapTokensForExactTokens
Assert(input.Deadline.Seconds >= Context.CurrentBlockTime.Seconds, "Expired");

// SwapExactTokensForTokensSupportingFeeOnTransferTokens
Assert(input.Deadline.Seconds >= Context.CurrentBlockTime.Seconds, "Expired");
```

Impact

If a request is executed within a second of its deadline, whether it is ultimately executed may depend on the nanoseconds in the timestamps, depending on the exact call. This is not a significant impact, but these should be made consistent.

Recommendations

We recommend selecting either comparing the seconds or comparing the dates (with nanosecond precision) across all the actions.

Remediation

This issue has been acknowledged by Awaken Finance, and a fix was implemented in commit `1eeef4bf`.

3.7. There is no way to set `ExternalInfo` for LP tokens

Target	AwakenSwapContract		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `ExternalInfo` feature of the TokenContract allows tokens to have arbitrary key-value metadata, with one constant key, `awaken_transfer_callback`, having the special behavior that it specifies a callback to be called upon token transfers.

However, the only ways this `ExternalInfo` state can change are during `Create` and `ResetExternalInfo`:

```
public override Empty Create(CreateInput input)
{
    // [...]

    var tokenInfo = new TokenInfo
    {
        // [...]

        ExternalInfo = input.ExternalInfo
    };

    // [...]
}

public override Empty ResetExternalInfo(ResetExternalInfoInput input)
{
    var tokenInfo = State.TokenInfoMap[input.Symbol];
    Assert(tokenInfo.Issuer == Context.Sender, "No permission to reset
    external info.");
    tokenInfo.ExternalInfo = input.ExternalInfo;
    State.TokenInfoMap[input.Symbol] = tokenInfo;
    Context.Fire(new ExternalInfoChanged
    {
        Symbol = input.Symbol,
        ExternalInfo = input.ExternalInfo
    });
    return new Empty();
```

```
}
```

For the TokenContract owned by the AwakenSwapContract, the `Create` call is hardcoded to not specify any `ExternalInfo`:

```
public override Address CreatePair(CreatePairInput input)
{
    // [...]

    State.LPTokenContract.Create.Send(new Token.CreateInput
    {
        Symbol = GetTokenPairSymbol(tokenPair[0], tokenPair[1]),
        Decimals = 8,
        TokenName = $"Awaken {GetTokenPair(tokenPair[0], tokenPair[1])} LP
    Token",
        Issuer = Context.Self,
        IsBurnable = true,
        TotalSupply = long.MaxValue
    });

    // [...]
}
```

And the TokenContract contract never calls `ResetExternalInfo`. Since there is only one issuer of a token, and only the issuer may call `ResetExternalInfo`, this means it is not possible to set the `ExternalInfo` for an LP token.

Impact

For LP tokens created by the AwakenSwapContract, the `ExternalInfo` feature cannot be used.

Recommendations

We recommend either removing this feature, if it is unnecessary, or allowing an admin to be able to call `ResetExternalInfo` on LP tokens through the AwakenSwapContract in order to be able to add this feature to LP tokens in the future.

Remediation

As `ExternalInfo` cannot be set in the current implementation of the contract, Awaken Finance has chosen to defer implementing the fix for this issue.

3.8. Duplicate pair check is skipped in `SwapInternal`

Target	AwakenSwapContractHelpers		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

In SwapExternal, there is a check that `to != pairAddress`:

```
private void SwapInternal(string symbolIn, string symbolOut, long amountIn,
    long amountOut, Address to,
    string channel)
{
    // [...]

    Assert(to != pairAddress, "Invalid account address");
```

The purpose of this check in the Uniswap code that this function was adapted from is to ensure that the address that is receiving the `symbolOut` tokens is not the same as the currently executing contract. This is important because the logic outside the `SwapExternal` call already transferred the tokens to the target, whereas the math inside it assumes that the entire balance above the reserves is part of the swap. So, if the intended destination for the tokens is the pool itself, the destination tokens will be double-counted.

However, in this code, no matter what the pair is, if it is an internal swap pair, `to` will always be the `Context.Self` address. This is because the virtual addresses do not hold the actual tokens — accounting for token ownership happens internally through `State.PoolBalanceMap` only.

Impact

If a swap path contains the same pair twice, this check fails to prevent the destination symbol balance from being double-counted. For example, if the `SwapExactTokensForTokens` in the `Case0Test` in AwakenSwapContractTests is replaced with `Path = {"ELF","TEST","ELF"}`, then this assert fails to detect the problem.

There is no actual impact, since the later check `Error with K` catches the issue — if this check was not there, then this would be a critical issue, because value would be lost through a decreased K.

With careful manipulation of pairs and parameters, it may be possible to bypass fees using this exploit by swapping pairs in a way that this exploit covers swap fees without decreasing K.

Recommendations

Instead of comparing `to` with `pairAddress`, this check should compare `nextPair` with it. This check can be more easily implemented outside `SwapInternal`.

Remediation

This issue has been acknowledged and fixed in the following commits:

- `1eeef4bf` ↗
- `0949d89e` ↗

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Virtual addresses are not actually used

The AElf protocol allows contracts to call other contracts as any of their virtual addresses, in order for a contract to more easily hold tokens on behalf of a separate agent.

The AwakenSwap contract uses virtual addresses to represent pools, but it does not actually hold assets using these virtual addresses, preferring instead to internally record which assets belong to which pool using the state variable `PoolBalanceMap`.

Since they will not be used as callers, it would save contract processing time to not have to hash and calculate the virtual addresses for the pools. Alternatively, to better utilize the underlying protocol, and to create a more user-friendly way of viewing what assets are in what pools, it would be better to use the virtual addresses to store pool assets and actually transfer the assets between virtual addresses when swapping.

4.2. No tests for ACS2 read/write path correctness

There are no tests for the correctness of the ACS2 read/write dependencies. Those resource values being wrong could cause inefficient concurrency, so we recommend adding tests that invoke and check these methods.

4.3. Additional token-sanity checks

We recommend adding some missing checks to the token contract:

- The `DoTransfer` method should check whether `To` is null in case anyone attempts to burn tokens by sending them to the null address, rather than calling `Burn`.
- The `Create` method should check that `Decimals` is nonnegative and not too large. User-interface denial-of-service issues may arise if the number of decimals is not within a reasonable logarithm of the range of the token balances.
- The `AddMinter` and `RemoveMinter` methods should have null checks on the addresses added/removed.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: AwakenSwapContract.cs

Function: `Empty Initialize(InitializeInput input)`

Initializes the contract. Can only be called by deployer.

Inputs

- `input.AwakenTokenContractAddress`
  - **Validation**: No validation.
  - **Impact**: Sets the address of the LP token contract, which should be a deployment of AwakenTokenContract.
- `input.Admin`
  - **Validation**: Defaults to sender if null.
  - **Impact**: Sets the admin address.

Branches and code coverage (including function calls)

Intended branches

- ☐ Initializes contract.

Negative behavior

- ☐ Reverts if contract is already initialized.
- ☐ Reverts if contract is uninitialized but the caller is not the contract author.

Function call analysis

This function makes no external state-mutating calls.

Function: `Address CreatePair(CreatePairInput input)`

Creates a new swap pair. Can be called by anyone.

Inputs

- `input.SymbolPair`
  - **Validation**: Must be a single symbol pair. Symbols may not be the same, and the pair may not already exist.
  - **Impact**: Symbol pair to create a swap on.

Branches and code coverage (including function calls)

Intended branches

- ☑ Creates the first pair, initializing `pairList`.
- ☑ Creates more pairs, adding to `pairList`.

Negative behavior

- ☐ Reverts if contract is not initialized.
- ☐ Reverts if token pair is unparsable or has invalid tokens.
- ☐ Reverts if token pair is two of the same tokens.
- ☐ Reverts if token pair already exists.

Function call analysis

- `CreatePair -> LPTokenContract.Create`
  - **Argument control?**
    - `Symbol`: Normalized token pair. Validity checked by contract.
    - `Decimals`: Constant 8.
    - `TokenName`: Formatted string.
    - `Issuer`: Constant `Context.Self`.
    - `IsBurnable`: Constant `true`.
    - `TotalSupply`: Constant `long.MaxValue`.
  - **Impact**: Creates the new token pair on the underlying AwakenTokenContract, setting the issuer to this swap contract.

5.2. Module: AwakenSwapContract_Admin.cs

Function: `Empty SetFeeRate(Int64Value input)`

Sets the fee rate. Caller must be admin.

Inputs

- `input`

- **Validation**: None.
- **Impact**: New fee rate.

Branches and code coverage (including function calls)

Intended branches

- ☐ Successfully sets fee rate.

Negative behavior

- ☐ Caller is not admin.

Function call analysis

This function makes no external state-mutating calls.

Function: `Empty SetFeeTo(Address input)`

Sets the fee recipient. Caller must be admin.

Inputs

- `input`
  - **Validation**: None.
  - **Impact**: New fee recipient.

Branches and code coverage (including function calls)

Intended branches

- ☐ Successfully sets fee recipient.

Negative behavior

- ☐ Caller is not admin.

Function call analysis

This function makes no external state-mutating calls.

Function: `Empty ChangeOwner(Address input)`

Transfers ownership to a new admin. Caller must be admin.

Inputs

- `input`
  - **Validation**: None.
  - **Impact**: New admin.

Branches and code coverage (including function calls)

Intended branches

- ☐ Successfully transfers admin.

Negative behavior

- ☐ Caller is not admin.

Function call analysis

This function makes no external state-mutating calls.

5.3. Module: AwakenSwapContract_Helper.cs

Function: `long[] AddLiquidity(string tokenA, string tokenB, long amountADesired, long amountBDesired, long amountAMin, long amountBMin)`

Adds liquidity to a token pair.

Inputs

- `tokenA`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Used to add liquidity.
- `tokenB`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Used to add liquidity.
- `amountADesired`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies desired amount of tokenA.
- `amountBDesired`

- **Validation**: Must be greater than zero.
- **Impact**: Specifies desired amount of tokenB.

- `amountAMin`
  - **Validation**: Must be greater than or equal to zero.
  - **Impact**: Ensures minimum amount of tokenA.
- `amountBMin`
  - **Validation**: Must be greater than or equal to zero.
  - **Impact**: Ensures minimum amount of tokenB.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully adds liquidity.

Negative behavior

- ☐ Invalid token symbols.
- ☐ Insufficient token amounts.
- ☐ Pair does not exist.
- ☐ Optimal amounts calculation issues.

Function call analysis

This function interacts with external state (token balances, reserves).

Function: `long[] RemoveLiquidity(string tokenA, string tokenB, long liquidityRemoveAmount, long amountAMin, long amountBMin, Address to)`

Removes liquidity from a token pair.

Inputs

- `tokenA`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Used to remove liquidity.
- `tokenB`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Used to remove liquidity.
- `liquidityRemoveAmount`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies amount of liquidity to remove.

- `amountAMin`
  - **Validation**: Must be greater than or equal to zero.
  - **Impact**: Ensures minimum amount of tokenA to be received.
- `amountBMin`
  - **Validation**: Must be greater than or equal to zero.
  - **Impact**: Ensures minimum amount of tokenB to be received.
- `to`
  - **Validation**: Must be a valid address.
  - **Impact**: Specifies recipient address.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully removes liquidity.

Negative behavior

- ☐ Invalid token symbols.
- ☐ Insufficient liquidity tokens.
- ☐ Pair does not exist.

Function call analysis

This function interacts with external state (token balances, reserves).

Function: `long MintLPToken(string tokenA, string tokenB, long amountA, long amountB, Address account, string channel)`

Mints liquidity pool tokens based on provided token amounts.

Inputs

- `tokenA`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Used in minting LP tokens.
- `tokenB`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Used in minting LP tokens.
- `amountA`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies the amount of tokenA.

- `amountB`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies the amount of tokenB.
- `account`
  - **Validation**: Must be a valid address.
  - **Impact**: Recipient of the minted LP tokens.
- `channel`
  - **Validation**: None.
  - **Impact**: Specifies the channel for the tokens.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully mints LP tokens.

Negative behavior

- ☐ Invalid token symbols.
- ☐ Insufficient token amounts.
- ☐ Pair does not exist.
- ☐ Calculation issues for liquidity amount.

Function call analysis

This function interacts with external state (token balances, reserves).

Function: `long[] PerformBurn(Address to, string tokenA, string tokenB, long liquidityRemoveAmount)`

Burns liquidity pool tokens and returns the underlying tokens.

Inputs

- `to`
  - **Validation**: Must be a valid address.
  - **Impact**: Recipient of the underlying tokens.
- `tokenA`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Token to be returned.
- `tokenB`
  - **Validation**: Must be a valid token symbol.

    - **Impact**: Token to be returned.
- `liquidityRemoveAmount`
    - **Validation**: Must be greater than zero.
    - **Impact**: Specifies the amount of liquidity to burn.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully burns LP tokens and returns underlying tokens.

Negative behavior

- ☐ Invalid token symbols.
- ☐ Insufficient liquidity tokens.
- ☐ Pair does not exist.

Function call analysis

This function interacts with external state (token balances, reserves).

Function: `void Swap(RepeatedField<long> amounts, RepeatedField<string> path, Address lastTo, string channel)`

Performs a token swap along a specified path.

Inputs

- `amounts`
    - **Validation**: Must correspond to the path.
    - **Impact**: Specifies the amounts for the swap.
- `path`
    - **Validation**: Must be a valid path of token symbols.
    - **Impact**: Specifies the path for the swap.
- `lastTo`
    - **Validation**: Must be a valid address.
    - **Impact**: Final recipient of the swap output.
- `channel`
    - **Validation**: None.
    - **Impact**: Specifies the channel for the swap.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully performs token swap.

Negative behavior

- ☐ Invalid token symbols.
- ☐ Invalid swap path.
- ☐ Insufficient token amounts.

Function call analysis

This function interacts with external state (token balances, reserves).

Function: `void SwapSupportingFeeOnTransferTokens(RepeatedField<string> path, Address lastTo, string channel)`

Performs a token swap supporting tokens with transfer fees.

Inputs

- `path`
  - **Validation**: Must be a valid path of token symbols.
  - **Impact**: Specifies the path for the swap.
- `lastTo`
  - **Validation**: Must be a valid address.
  - **Impact**: Final recipient of the swap output.
- `channel`
  - **Validation**: None.
  - **Impact**: Specifies the channel for the swap.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully performs token swap.

Negative behavior

- ☐ Invalid token symbols.
- ☐ Invalid swap path.
- ☐ Insufficient token amounts.

Function call analysis

This function interacts with external state (token balances, reserves).

Function: `void TransferIn(Address pair, Address from, string symbol, long amount)`

Transfers tokens into the contract from a specified address.

Inputs

- `pair`
  - **Validation**: Must be a valid address.
  - **Impact**: Recipient of the transferred tokens.
- `from`
  - **Validation**: Must be a valid address.
  - **Impact**: Source of the transferred tokens.
- `symbol`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Specifies the token to transfer.
- `amount`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies the amount of tokens to transfer.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully transfers tokens in.

Negative behavior

- ☐ Invalid token symbols.
- ☐ Insufficient token balance.
- ☐ Transfer failure.

Function call analysis

This function interacts with external state (token balances).

Function: `void TransferOut(Address pair, Address to, string symbol, long amount)`

Transfers tokens out of the contract to a specified address.

Inputs

- `pair`
  - **Validation**: Must be a valid address.
  - **Impact**: Source of the transferred tokens.
- `to`
  - **Validation**: Must be a valid address.
  - **Impact**: Recipient of the transferred tokens.
- `symbol`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Specifies the token to transfer.
- `amount`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies the amount of tokens to transfer.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully transfers tokens out.

Negative behavior

- ☐ Invalid token symbols.
- ☐ Insufficient token balance.
- ☐ Transfer failure.

Function call analysis

This function interacts with external state (token balances).

5.4. Module: AwakenSwapContract_LP.cs

Function: `AddLiquidityOutput AddLiquidity(AddLiquidityInput input)`

Adds liquidity to a liquidity pool for a given pair of tokens. The function ensures that the contract is initialized, checks the input parameters for validity, and performs necessary transfers and minting of liquidity pool tokens.

Inputs

- `input.SymbolA`
  - **Validation**: Must be an existing symbol for the first token.
  - **Impact**: Ensures that the symbol corresponds to a valid token to be paired.
- `input.SymbolB`
  - **Validation**: Must be an existing symbol for the second token.
  - **Impact**: Ensures that the symbol corresponds to a valid token to be paired.
- `input.AmountADesired`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies the desired amount of the first token to add.
- `input.AmountBDesired`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies the desired amount of the second token to add.
- `input.AmountAMin`
  - **Validation**: Must be greater than or equal to zero.
  - **Impact**: Ensures the minimum amount of the first token to add.
- `input.AmountBMin`
  - **Validation**: Must be greater than or equal to zero.
  - **Impact**: Ensures the minimum amount of the second token to add.
- `input.Deadline`
  - **Validation**: Must be greater than or equal to the current block time.
  - **Impact**: Ensures that the operation is performed within a valid time frame.
- `input.To`
  - **Validation**: Not null.
  - **Impact**: Specifies the recipient of the liquidity tokens.
- `input.Channel`
  - **Validation**: Not explicitly validated in this function.
  - **Impact**: Specifies the channel for the liquidity tokens.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully adds liquidity.

Negative behavior

- ☐ Contract is not initialized.
- ☐ Deadline has passed.
- ☐ Invalid input amounts (`AmountADesired`, `AmountBDesired`, `AmountAMin`, and `AmountBMin`).
- ☐ Token symbols do not exist.

Function call analysis

This function makes no external state-mutating calls.

Function: `RemoveLiquidityOutput RemoveLiquidity(RemoveLiquidityInput input)`

Removes liquidity from a liquidity pool for a given pair of tokens. The function ensures that the contract is initialized, checks the input parameters for validity, and performs the necessary operations to remove liquidity and return the tokens to the user.

Inputs

- `input.SymbolA`
  - **Validation**: Must be an existing symbol for the first token.
  - **Impact**: Ensures that the symbol corresponds to a valid token to be removed from the pool.
- `input.SymbolB`
  - **Validation**: Must be an existing symbol for the second token.
  - **Impact**: Ensures that the symbol corresponds to a valid token to be removed from the pool.
- `input.LiquidityRemove`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies the amount of liquidity to be removed.
- `input.AmountAMin`
  - **Validation**: Must be greater than or equal to zero.
  - **Impact**: Ensures the minimum amount of the first token to be returned.
- `input.AmountBMin`
  - **Validation**: Must be greater than or equal to zero.
  - **Impact**: Ensures the minimum amount of the second token to be returned.
- `input.Deadline`
  - **Validation**: Must be greater than or equal to the current block time.
  - **Impact**: Ensures that the operation is performed within a valid time frame.
- `input.To`
  - **Validation**: None.
  - **Impact**: Specifies the recipient of the returned tokens.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully removes liquidity.

Negative behavior

- ☐ Contract is not initialized.
- ☐ Deadline has passed.
- ☐ Invalid input amounts (`AmountAMin`, `AmountBMin`, and `LiquidityRemove`).
- ☐ Token symbols do not exist.

Function call analysis

This function performs the liquidity withdrawal by performing a transfer of the tokens A and B by invoking the configured `TokenContract`'s transfer function.

5.5. Module: AwakenSwapContract_Swap.cs

Function: `SwapOutput SwapExactTokensForTokens(SwapExactTokensForTokensInput input)`

Swaps an exact amount of input tokens for as many output tokens as possible, subject to a minimum output amount. The function ensures that the contract is initialized, checks the input parameters for validity, and performs the necessary operations to execute the token swap.

Inputs

- `input.AmountIn`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies the exact amount of input tokens to be swapped.
- `input.AmountOutMin`
  - **Validation**: Must be greater than or equal to zero.
  - **Impact**: Ensures that the minimum acceptable amount of output tokens is specified.
- `input.Deadline`
  - **Validation**: Must be greater than or equal to the current block time.
  - **Impact**: Ensures that the operation is performed within a valid time frame.
- `input.Path`
  - **Validation**: Must contain valid token symbols for the swap path.
  - **Impact**: Defines the route for the token swap.
- `input.To`
  - **Validation**: None.
  - **Impact**: Specifies the recipient of the output tokens.
- `input.Channel`
  - **Validation**: Not explicitly validated in this function.
  - **Impact**: Specifies the channel for the swap.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully performs the token swap.

Negative behavior

- ☐ Contract is not initialized.
- ☐ Deadline has passed.
- ☐ Invalid input amounts (`AmountIn` and `AmountOutMin`).
- ☐ Insufficient output amount.
- ☐ Invalid token path.

Function call analysis

This function transfers tokens out via an external call to the configured TokenContract.

Function: `SwapOutput SwapTokensForExactTokens(SwapTokensForExactTokensInput input)`

Swaps tokens to receive an exact amount of output tokens, subject to a maximum input amount. The function ensures that the contract is initialized, checks the input parameters for validity, and performs the necessary operations to execute the token swap.

Inputs

- `input.AmountOut`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies the exact amount of output tokens to be received.
- `input.AmountInMax`
  - **Validation**: Must be greater than zero.
  - **Impact**: Ensures that the maximum acceptable amount of input tokens is specified.
- `input.Deadline`
  - **Validation**: Must be greater than or equal to the current block time.
  - **Impact**: Ensures that the operation is performed within a valid time frame.
- `input.Path`
  - **Validation**: Must contain valid token symbols for the swap path.
  - **Impact**: Defines the route for the token swap.
- `input.To`
  - **Validation**: None.
  - **Impact**: Specifies the recipient of the output tokens.

- `input.Channel`
  - **Validation**: None.
  - **Impact**: Specifies the channel for the swap.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully performs token swap.

Negative behavior

- ☐ Contract is not initialized.
- ☐ Deadline has passed.
- ☐ Invalid input amounts (`AmountOut` and `AmountInMax`).
- ☐ Excessive input amount.
- ☐ Invalid token path.

Function call analysis

This function transfers tokens out via an external call to the configured TokenContract.

Function: `Empty SwapExactTokensForTokensSupportingFeeOnTransferTokens(SwapExactTokensForTokensSupportingFeeOnTransferTokensInput input)`

Swaps an exact amount of input tokens for as many output tokens as possible, supporting tokens that have transfer fees. The function ensures that the contract is initialized, checks the input parameters for validity, performs the necessary operations to execute the token swap, and verifies the swap outcome.

Inputs

- `input.AmountIn`
  - **Validation**: Must be greater than zero.
  - **Impact**: Specifies the exact amount of input tokens to be swapped.
- `input.AmountOutMin`
  - **Validation**: Must be greater than or equal to zero.
  - **Impact**: Ensures that the minimum acceptable amount of output tokens is specified.
- `input.Deadline`
  - **Validation**: Must be greater than or equal to the current block time.
  - **Impact**: Ensures that the operation is performed within a valid time frame.

- `input.Path`
  - **Validation**: Must contain valid token symbols for the swap path.
  - **Impact**: Defines the route for the token swap.
- `input.To`
  - **Validation**: None.
  - **Impact**: Specifies the recipient of the output tokens.
- `input.Channel`
  - **Validation**: Not explicitly validated in this function.
  - **Impact**: Specifies the channel for the swap.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully performs the token swap.

Negative behavior

- ☐ Contract is not initialized.
- ☐ Deadline has passed.
- ☐ Invalid input amounts (`AmountIn` and `AmountOutMin`).
- ☐ Insufficient output amount.
- ☐ Invalid token path.

Function call analysis

This function transfers tokens out via an external call to the configured TokenContract.

Function: `Empty SwapExactTokensForTokensSupportingFeeOnTransferTokensVerify(SwapExactTokensForTokensSupportingFeeOnTransferTokensVerifyInpu input)`

Verifies the outcome of a token swap that supports transfer fees. The function ensures that the contract is initialized, checks the input parameters for validity, verifies the sender, and confirms that the expected amount of output tokens was received.

Inputs

- `input.AmountBefore`
  - **Validation**: Assumed valid as it is derived from the previous function call.
  - **Impact**: Used to calculate the difference in token balance before and after the swap.

- `input.AmountOutMin`
  - **Validation**: Must be greater than or equal to zero.
  - **Impact**: Ensures the minimum acceptable amount of output tokens was received.
- `input.Symbol`
  - **Validation**: Must be an existing symbol.
  - **Impact**: Specifies the token symbol being verified.
- `input.To`
  - **Validation**: None.
  - **Impact**: Specifies the recipient of the output tokens.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully verifies the swap.

Negative behavior

- ☐ Contract is not initialized.
- ☐ Sender is not the contract itself.
- ☐ Insufficient output amount.

Function call analysis

This function makes no external state-mutating calls.

5.6. Module: AwakenSwapContract_Views.cs

Function: `GetReservesOutput GetReserves(GetReservesInput input)`

Fetches the reserve amounts for a given list of token pairs. The function retrieves and validates token pairs and then fetches the reserve amounts and the last block timestamp for each pair.

Inputs

- `input.SymbolPair`
  - **Validation**: Each pair must contain valid token symbols.
  - **Impact**: Defines the list of token pairs for which reserves are fetched.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully fetches reserves for token pairs.

Negative behavior

- ☐ Invalid token pair symbols.

Function call analysis

This function makes no external state-mutating calls.

Function: `GetTotalSupplyOutput GetTotalSupply(StringList input)`

Fetches the total supply for a given list of token pairs. The function retrieves and validates token pairs, fetches the liquidity-pool token information for each pair, and returns the total supply.

Inputs

- `input.Value`
  - **Validation**: Each pair must contain valid token symbols.
  - **Impact**: Defines the list of token pairs for which total supply is fetched.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully fetches total supply for token pairs.

Negative behavior

- ☐ Invalid token pair symbols.

Function call analysis

This function makes no external state-mutating calls.

Function: `Int64Value Quote(QuoteInput input)`

Calculates the amount of token B that would be received for a given amount of token A based on the current reserves in the liquidity pool. The function ensures that the token pair exists and retrieves the reserves to perform the calculation.

Inputs

- `input.SymbolA`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Specifies the input token for the quote calculation.
- `input.SymbolB`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Specifies the output token for the quote calculation.
- `input.AmountA`
  - **Validation**: Must be a nonnegative value.
  - **Impact**: Specifies the amount of token A for which the quote is being calculated.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully calculates the quote for the token pair.

Negative behavior

- ☐ Token pair does not exist.
- ☐ Invalid token symbols.
- ☐ Invalid input amount.

Function call analysis

This function makes no external state-mutating calls.

Function: `Int64Value GetAmountIn(GetAmountInInput input)`

Calculates the amount of input tokens needed to receive a specific amount of output tokens based on the current reserves in the liquidity pool. The function ensures that the token pair exists and retrieves the reserves to perform the calculation.

Inputs

- `input.SymbolIn`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Specifies the input token for the calculation.
- `input.SymbolOut`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Specifies the output token for the calculation.

- `input.AmountOut`
  - **Validation**: Must be a nonnegative value.
  - **Impact**: Specifies the desired amount of output tokens.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully calculates the required input amount for the token pair.

Negative behavior

- ☐ Token pair does not exist.
- ☐ Invalid token symbols.
- ☐ Invalid output amount.

Function call analysis

This function makes no external state-mutating calls.

Function: `Int64Value GetAmountOut(GetAmountOutInput input)`

Calculates the amount of output tokens received for a given amount of input tokens based on the current reserves in the liquidity pool.

Inputs

- `input.SymbolIn`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Specifies the input token for the calculation.
- `input.SymbolOut`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Specifies the output token for the calculation.
- `input.AmountIn`
  - **Validation**: Must be a nonnegative value.
  - **Impact**: Specifies the amount of input tokens.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully calculates the output amount for the token pair.

Negative behavior

- ☐ Token pair does not exist.
- ☐ Invalid token symbols.
- ☐ Invalid input amount.

Function call analysis

This function makes no external state-mutating calls.

Function: `GetAmountsInOutput GetAmountsIn(GetAmountsInInput input)`

Calculates the amounts of input tokens needed to receive a specific amount of output tokens along a specified path.

Inputs

- `input.AmountOut`
  - **Validation**: Must be a nonnegative value.
  - **Impact**: Specifies the desired amount of output tokens.
- `input.Path`
  - **Validation**: Must contain valid token symbols for the swap path.
  - **Impact**: Defines the route for the token swap.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully calculates the required input amounts for the token path.

Negative behavior

- ☐ Invalid token path.
- ☐ Invalid output amount.

Function call analysis

This function makes no external state-mutating calls.

Function: `GetAmountsOutOutput GetAmountsOut(GetAmountsOutInput input)`

Calculates the amounts of output tokens received for a specific amount of input tokens along a specified path.

Inputs

- `input.AmountIn`
  - **Validation**: Must be a nonnegative value.
  - **Impact**: Specifies the amount of input tokens.
- `input.Path`
  - **Validation**: Must contain valid token symbols for the swap path.
  - **Impact**: Defines the route for the token swap.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully calculates the output amounts for the token path.

Negative behavior

- ☐ Invalid token path.
- ☐ Invalid input amount.

Function call analysis

This function makes no external state-mutating calls.

Function: `BigIntValue GetKLast(Address pair)`

Retrieves the last-known constant product (K value) for a given liquidity pair.

Inputs

- `pair`
  - **Validation**: Must be a valid address.
  - **Impact**: Specifies the liquidity pair for which the K value is being retrieved.

Branches and code coverage (including function calls)

Intended branches

☑ Successfully retrieves the K value for the liquidity pair.

Negative behavior

☐ Invalid pair address.

Function call analysis

This function makes no external state-mutating calls.

Function: `Address GetAdmin(Empty input)`

Retrieves the admin address for the contract.

Inputs

- `input`
  - **Validation**: Not applicable (empty input).
  - **Impact**: No impact on the function.

Branches and code coverage (including function calls)

Intended branches

☑ Successfully retrieves the admin address.

Negative behavior

- None.

Function call analysis

This function makes no external state-mutating calls.

Function: `Address GetFeeTo(Empty input)`

Retrieves the address designated to receive fees for the contract.

Inputs

- `input`
  - **Validation**: Not applicable (empty input).

- **Impact**: No impact on the function.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully retrieves the fee-recipient address.

Negative behavior

- None.

Function call analysis

This function makes no external state-mutating calls.

Function: `Int64Value GetFeeRate(Empty input)`

Retrieves the fee rate for the contract.

Inputs

- `input`
  - **Validation**: Not applicable (empty input).
  - **Impact**: No impact on the function.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully retrieves the fee rate.

Negative behavior

- None.

Function call analysis

This function makes no external state-mutating calls.

Function: `Address GetPairAddress(GetPairAddressInput input)`

Retrieves the address of the liquidity pair for the given token symbols.

Inputs

- `input.SymbolA`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Specifies the first token symbol of the pair.
- `input.SymbolB`
  - **Validation**: Must be a valid token symbol.
  - **Impact**: Specifies the second token symbol of the pair.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully retrieves the pair address.

Negative behavior

- ☐ Invalid token symbols.

Function call analysis

This function makes no external state-mutating calls.

5.7. Module: TokenContract.cs

Function: `Empty Initialize(InitializeInput input)`

Initializes the contract. Can only be called by deployer.

Inputs

- `input.Owner`
  - **Validation**: None.
  - **Impact**: Sets the owner of the contract. The owner is also added to the `MinterMap`.

Branches and code coverage (including function calls)

Intended branches

- ☐ Initializes contract.

Negative behavior

- ☐ Reverts if contract is already initialized.
- ☐ Reverts if contract is uninitialized but the caller is not the contract author.

Function call analysis

This function makes no external state-mutating calls.

5.8. Module: TokenContract_ACS2.cs

Function: `ResourceInfo GetResourceInfo(Transaction txn)`

Gets the read and write data dependencies for a given transaction.

Inputs

- `txn`
  - **Validation**: None.
  - **Impact**: Transaction to decode.

Branches and code coverage (including function calls)

Intended branches

- ☐ Transfer data dependency is correct.
- ☐ `TransferFrom` data dependency is correct.
- ☐ Other calls return nonparallelizable.

Negative behavior

- N/A.

Function call analysis

This function makes no external state-mutating calls.

5.9. Module: TokenContract_Actions.cs

Function: `Empty Create(CreateInput input)`

Creates a new token symbol. Should only be callable by a minter in the `MinterMap`.

Inputs

- `input.Symbol`
  - **Validation**: Must be nonempty and not an existing symbol.
  - **Impact**: Token symbol to create.
- `input.TokenName`
  - **Validation**: Must be nonempty.
  - **Impact**: Name of token to create.
- `input.TotalSupply`
  - **Validation**: Must be positive.
  - **Impact**: Total supply of the new token.
- `input.Decimals`
  - **Validation**: None.
  - **Impact**: Decimals of the new token.
- `input.Issuer`
  - **Validation**: None, except it is reset to sender if it is null.
  - **Impact**: Issuer of the token — can call `Issue` on it.
- `input.IsBurnable`
  - **Validation**: None.
  - **Impact**: Whether the token can be burned.
- `input.ExternalInfo`
  - **Validation**: None.
  - **Impact**: Mapping of arbitrary key values for external extensions.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully creates a token.

Negative behavior

- ☐ Caller is not minter, or contract is uninitialized.
- ☐ Symbol is empty or an existing symbol.
- ☐ Name is empty.
- ☐ Total supply is not positive.

Function call analysis

This function makes no external state-mutating calls.

Function: `Empty Issue(IssueInput input)`

Mints new tokens. Can only be called by the issuer of the specific symbol.

Inputs

- `input.Symbol`
  - **Validation**: Must be an existing symbol.
  - **Impact**: Token symbol to mint.
- `input.Amount`
  - **Validation**: Must be positive and not cause the total supply to be exceeded.
  - **Impact**: Amount of token to mint.
- `input.Memo`
  - **Validation**: None.
  - **Impact**: Note to be passed to emitted event.
- `input.To`
  - **Validation**: Not null.
  - **Impact**: Recipient of the new tokens.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully issues tokens.

Negative behavior

- ☐ Caller is not issuer.
- ☐ Amount is negative.
- ☐ Token symbol does not exist.
- ☐ Tokens issued in excess of total supply.

Function call analysis

This function makes no external state-mutating calls.

Function: `Empty Transfer(TransferInput input)`

Transfers tokens from the sender to a specified recipient.

Inputs

- `input.To`
  - **Validation**: Not null and not the sender.
  - **Impact**: Recipient of the tokens.
- `input.Symbol`
  - **Validation**: Must be an existing symbol.
  - **Impact**: Token symbol to transfer.
- `input.Amount`
  - **Validation**: Must not exceed sender's token balance.
  - **Impact**: Amount of token to transfer.
- `input.Memo`
  - **Validation**: None.
  - **Impact**: Note to be passed to emitted event.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully transfers tokens.

Negative behavior

- ☐ Amount is negative.
- ☐ Token symbol does not exist.
- ☐ Recipient is equal to sender.
- ☐ Sender does not have enough tokens to transfer.

Function call analysis

This function may make an arbitrary external call through `ExternalInfo`.

Function: `Empty TransferFrom(TransferFromInput input)`

Transfers tokens from a sender that has allowed the caller to spend their tokens to a specified recipient.

Inputs

- `input.From`
  - **Validation**: Not null.
  - **Impact**: Sender of the tokens.
- `input.To`

- **Validation**: Not null and not the sender.
- **Impact**: Recipient of the tokens.

- `input.Symbol`
  - **Validation**: Must be an existing symbol.
  - **Impact**: Token symbol to transfer.
- `input.Amount`
  - **Validation**: Must not exceed sender's token balance or the spender's allowance from the sender.
  - **Impact**: Amount of token to transfer.
- `input.Memo`
  - **Validation**: None.
  - **Impact**: Note to be passed to emitted event.

Branches and code coverage (including function calls)

Intended branches

- ☑ Successfully transfers tokens.

Negative behavior

- ☐ Amount is negative.
- ☐ Token symbol does not exist.
- ☐ Recipient is equal to the sender.
- ☐ Sender does not have enough tokens to transfer.
- ☐ Caller does not have enough allowance to spend the tokens.

Function call analysis

This function may make an arbitrary external call through `ExternalInfo`.

Function: `Empty Approve(ApproveInput input)`

Increases the allowance for a given spender to spend the sender's tokens.

Inputs

- `input.Spender`
  - **Validation**: Not null.
  - **Impact**: Spender of the tokens.
- `input.Symbol`
  - **Validation**: Must be an existing symbol.

    - **Impact**: Token symbol to change the allowance on.
- `input.Amount`
    - **Validation**: Must be nonnegative.
    - **Impact**: Amount of allowance to add.

Branches and code coverage (including function calls)

Intended branches

- ☐ Successfully increases allowance.

Negative behavior

- ☐ Amount is negative.
- ☐ Token symbol does not exist.

Function call analysis

This function makes no external state-mutating calls.

Function: `Empty UnApprove(UnApproveInput input)`

Decreases the allowance for a given spender to spend the sender's tokens.

Inputs

- `input.Spender`
    - **Validation**: Not null.
    - **Impact**: Spender of the tokens.
- `input.Symbol`
    - **Validation**: Must be an existing symbol.
    - **Impact**: Token symbol to change the allowance on.
- `input.Amount`
    - **Validation**: Must be nonnegative.
    - **Impact**: Amount of allowance to remove.

Branches and code coverage (including function calls)

Intended branches

- ☐ Successfully decreases allowance by the amount.
- ☐ Successfully decreases allowance to zero if amount exceeds existing allowance.

Negative behavior

- ☐ Amount is negative.
- ☐ Token symbol does not exist.

Function call analysis

This function makes no external state-mutating calls.

Function: `Empty Burn(BurnInput input)`

Burns tokens owned by the sender, decreasing the supply.

Inputs

- `input.Symbol`
  - **Validation**: Must be an existing burnable symbol.
  - **Impact**: Token symbol to burn.
- `input.Amount`
  - **Validation**: Must be nonnegative. Sender must have that amount of tokens.
  - **Impact**: Amount of token to burn.

Branches and code coverage (including function calls)

Intended branches

- ☐ Successfully burns tokens.

Negative behavior

- ☐ Amount is negative.
- ☐ Token symbol does not exist.
- ☐ Token is not burnable.
- ☐ Sender does not have enough tokens to burn.

Function call analysis

This function makes no external state-mutating calls.

Function: `Empty ResetExternalInfo(ResetExternalInfoInput input)`

Changes the `ExternalInfo` associated with a symbol. Only callable by the issuer of the symbol.

Inputs

- `input.Symbol`
  - **Validation**: Must be an existing symbol whose issuer is the sender.
  - **Impact**: Token symbol to modify.
- `input.ExternalInfo`
  - **Validation**: None.
  - **Impact**: Mapping of arbitrary key values for external extensions.

Branches and code coverage (including function calls)

Intended branches

- ☐ Successfully changes `ExternalInfo`.

Negative behavior

- ☐ Token symbol does not exist.
- ☐ Caller is not the issuer of the token.

Function call analysis

This function makes no external state-mutating calls.

Function: `Empty AddMinter(Address input)`

Adds an address to the `MinterMap` to allow them to create tokens. Only callable by the owner.

Inputs

- `input`
  - **Validation**: None.
  - **Impact**: Address to add.

Branches and code coverage (including function calls)

Intended branches

- ☐ Successfully adds a minter.

Negative behavior

- ☐ Caller is not owner.

Function call analysis

This function makes no external state-mutating calls.

Function: `Empty RemoveMinter(Address input)`

Removes an address from the `MinterMap`. Only callable by the owner.

Inputs

- `input`
  - **Validation**: None.
  - **Impact**: Address to remove.

Branches and code coverage (including function calls)

Intended branches

- ☐ Successfully removes a minter.

Negative behavior

- ☐ Caller is not owner.

Function call analysis

This function makes no external state-mutating calls.

6. Assessment Results

At the time of our assessment, the reviewed code was deployed to the AElf mainnet.

During our assessment on the scoped Awaken Swap contracts, we discovered eight findings. No critical issues were found. Three findings were of high impact, two were of medium impact, one was of low impact, and the remaining findings were informational in nature.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.