Zellic

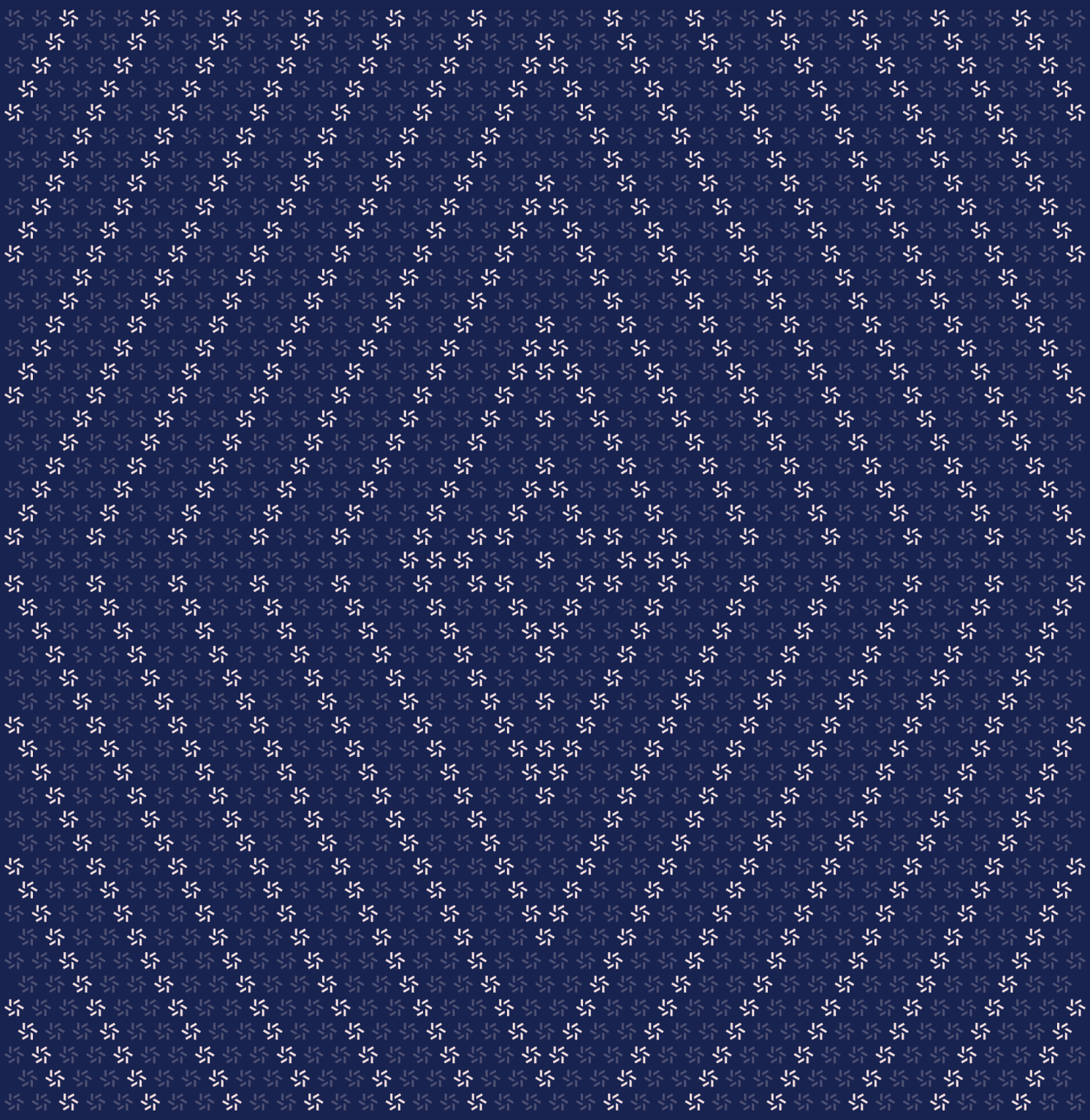
Prepared for
Cameron Fairchild
Latent

Prepared by
Kritsada Dechawattana
Sylvain Pelissier
Zellic

June 9, 2025

Wrapped Staked TAO

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 Latent from June 4th to June 5th, 2025. During this engagement, Zellic reviewed Wrapped Staked TAO'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 of withdrawing more TAO than expected from the share of the staking emissions (e.g., staking first, someone else stakes, and then the first staker can unstake more)?
- Are there reentrancy attacks that could break the contract's functionality?

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

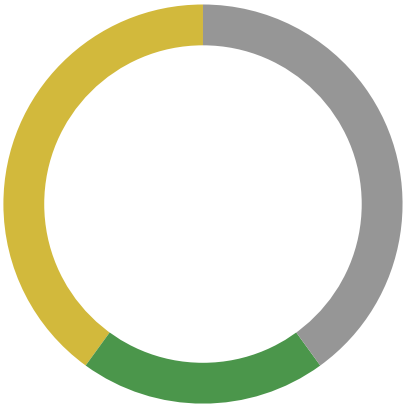
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Wrapped Staked TAO contracts, we discovered five findings. No critical issues were found. Two findings were of medium impact, one was of low impact, and the remaining findings were informational in nature.

Breakdown of Finding Impacts

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



2. Introduction

2.1. About Wrapped Staked TAO

Latent contributed the following description of Wrapped Staked TAO:

> Wrapped Staked TAO (wstTAO) is an ERC20-compatible wrapped version of TAO minted on the native Bittensor EVM. wstTAO is a representation of staked TAO, which means that holders of wstTAO benefit from staking emissions of the TAO being held by the contract. Units of wstTAO effectively represent shares of the underlying staked balance, with new mints requiring a deposit of TAO, which is automatically staked.

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.

2.3. Scope

The engagement involved a review of the following targets:

Wrapped Staked TAO Contracts

Type	Solidity
Platform	EVM-compatible
Target	wsttao
Repository	https://github.com/latent-to/wsttao ↗
Version	`ac3b5034c010d0a04c6fd654713cd4a87a803547`
Programs	`wstTAO`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of four person-days. The assessment was conducted by two consultants over the course of two calendar days.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Kritsada Dechawattana
Engineer
kritsada@zellic.io ↗

Sylvain Pelissier
Engineer
sylvain@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

June 4, 2025	Start of primary review period
June 5, 2025	Kick-off call
June 5, 2025	End of primary review period

3. Detailed Findings

3.1. Lack of test suite

Target	wstTAO		
Category	Code Maturity	**Severity**	High
Likelihood	High	**Impact**	Medium

Description

The project does not include functional testing, only scripts to interact with the contract. The nonfunctional tests verify only the intended branches.

Impact

When building a complex contract ecosystem with multiple moving parts and dependencies, comprehensive testing is essential. This includes testing for both positive and negative scenarios. Positive tests should verify that each function's side effect is as expected, while negative tests should cover every revert, preferably in every logical branch.

The test coverage for this project should cover the whole contract, not just surface-level functions. It is important to test the invariants required for ensuring security and also verify mathematical properties. Additionally, testing cross-chain function calls and transfers is recommended to ensure the desired functionality.

Recommendations

Therefore, we recommend building a rigorous test suite to ensure that the system operates securely and as intended.

Good test coverage has multiple effects.

- It finds bugs and design flaws early (preaudit or prerelease).
- It gives insight into areas for optimization (e.g., gas cost).
- It displays code maturity.
- It bolsters customer trust in your product.
- It improves understanding of how the code functions, integrates, and operates — for developers and auditors alike.
- It increases development velocity long-term.

The last point seems contradictory, given the time investment to create and maintain tests. To expand upon that, tests help developers trust their own changes. It is difficult to know if a code refactor — or even just a small one-line fix — breaks something if there are no tests. This is

especially true for new developers or those returning to the code after a prolonged absence. Tests have your back here. They are an indicator that the existing functionality *most likely* was not broken by your change to the code.

Remediation

This issue has been acknowledged by Latent, and a fix was implemented in PR #2 ↗.

3.2. Lack of minimum for staking

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

Description

Bittensor requires a minimum amount for staking and unstaking of 500,000 RAO (0.0005 TAO) and charges a fee of 50,000 RAO (0.00005 TAO) for both staking and unstaking operations.

A notable issue arises due to the lack of a minimum check in the `stake` function. When the pool has no staking, although a user needs to stake a minimum of 500,000 RAO, they must actually stake 600,000 RAO — the first 50,000 RAO over is for paying the fee when staking and another 50,000 RAO must be reserved for the future unstaking fee. Without this reserve when staking with the minimum, unstaking transactions would fail unless another user stakes more TAO to cover the unstaking fee.

Impact

When the pool has no staking, if the first staker deposits only the minimum amount, they cannot unstake until subsequent stakes are sufficient to cover the unstaking fee.

Recommendations

When the pool has no staking, implement a minimum stake requirement to ensure the first staker can unstake at any time, regardless of whether subsequent stakes occur.

```
function stake(address to) public payable {

  [...]

  uint256 currentStakeRaoDecimals = getCurrentStake(_netuid);
  if(currentStakeRaoDecimals == 0) {
    require(msg.value >= 0.0006 ether, "wstTAO: inefficient staking amount");
    // 600,000 RAO (0.0006 TAO)
  }

  [...]
```

```
}
```

Remediation

This issue has been acknowledged by Latent, and a fix was implemented in commit `fcb3ac5a` ↗.

3.3. Inconsistent staking fee

Target	WrappedStakedTAO		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

Bittensor charges a fee of 50,000 RAO (0.00005 TAO) for both staking and unstaking operations. However, from the Bittensor documentation,

> An existential deposit is the minimum required TAO in a wallet (i.e., in a coldkey). If a wallet balance goes below the existential deposit, then this wallet account is deactivated and the remaining TAO in it is destroyed. This is set to 500 RAO for any Bittensor wallet.

Consequently, after the WrappedStakedTAO contract is deployed, the first staking transaction requires a higher fee of 50,500 RAO (0.0000505 TAO), which is 500 RAO (0.0000005 TAO) more than the regular fee needed to maintain the wallet (the WrappedStakedTAO contract's coldkey).

Impact

This inconsistent fee structure creates unfairness among users, as the first staking transaction incurs higher fees than subsequent ones.

Recommendations

We recommend the protocol admin deposit the minimum required after deploying the WrappedStakedTAO contract to maintain the wallet before allowing users to make stakes. This ensures consistent fee charges for all users.

Remediation

This issue has been acknowledged by Latent, and a fix was implemented in commit `fcb3ac5a` ↗.

3.4. Redundant `console.log` statements from codebase

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

Description

The WrappedStakedTAO contract currently contains several `console.log` statements. While these logging statements can be invaluable during the development phase, they are unnecessary in production. Additionally, leaving them in the production code reduces readability.

Impact

This issue is classified as Informational since it poses no security risk. However, addressing this would improve code maintainability and readability.

Recommendations

Consider removing `console.log` from the contract before deploying to production.

Remediation

This issue has been acknowledged by Latent, and a fix was implemented in commit `aec95ce1` ↗.

3.5. Uninitialized `staking` state

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

Description

WrappedStakedTAO is an upgradable contract, and its states should be initialized within the `initialize` function. However, the `staking` state is assigned within the `constructor`, resulting in an uninitialized state.

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();

    staking = IStaking(ISTAKING_ADDRESS);
}

function initialize(address initialOwner) initializer public {
    __ERC20_init(NAME, SYMBOL);
    __ERC20Burnable_init();
    __ERC20Pausable_init();
    __Ownable_init(initialOwner);
    __ERC20Permit_init(NAME);
    __UUPSUpgradeable_init();

    _hotkey
    = 0x20b0f8ac1d5416d32f5a552f98b570f06e8392ccb803029e04f63fbe0553c954;
    _decimalConversionFactor = 10 ** 9;

    [...]
}
```

Impact

This issue is classified as Informational since the `staking` state is only used to get the selector for building the calling data, which poses no security risk. However, addressing this would improve code maintainability and make onboarding easier for new developers.

Recommendations

Since the `staking` state is only used to get the selector for building the calling data, it could be removed from the WrappedStakedTAO contract in favor of using `IStaking.*.selector` to build the calling data instead. However, if the decision is made to keep the `staking` state, it should be initialized within the `initialize` function for proper state initialization.

Remediation

This issue has been acknowledged by Latent, and a fix was implemented in commit `676015c1` ↗.

4. 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.

4.1. Module: wstTAO.sol

Function: `stake(address to)`

The function takes as input the value in TAO to stake, calls the staking precompile, and mints wstTAO to the address parameter. The rate of minted wstTAO is the current stake divided by the number of wstTAO tokens.

Inputs

- `to`
  - **Validation**: No validation.
  - **Impact**: The address to which the wstTAO is minted.
- `value`
  - **Validation**: The stake value is increased after calling the precompile.
  - **Impact**: The value to stake.

Branches and code coverage (including function calls)

Intended branches

- ☐ A given value is correctly staked, and the balances are correctly updated.

Negative behavior

- ☐ A stake with a too small value is rejected.

Function: `unstake(uint256 amountEvm)`

The function takes as input the amount of wstTAO to unstake, calls the unstaking precompile, burns the corresponding wstTAO tokens, and transfers TAOs back to the user. The amount returns the number of wstTAO tokens divided by the amount staked.

Inputs

- `amountEvm`
  - **Validation**: The user balance must be greater than the amount.
  - **Impact**: The amount of wstTAO to unstake.

Branches and code coverage (including function calls)

Intended branches

- ☐ A user with large enough balance can unstake correctly, and the balances are updated accordingly.

Negative behavior

- ☐ A user with a balance smaller than the requested amount cannot unstake.
- ☐ A user cannot unstake with an incorrect hotkey.

5. Assessment Results

During our assessment on the scoped Wrapped Staked TAO contracts, we discovered five findings. No critical issues were found. Two findings were of medium impact, one was of low impact, and the remaining findings were informational in nature.

This audit assumes that the smart contracts function correctly only within the context of the Bittensor EVM protocol version active at the time of this assessment. The review does not account for interactions with future Bittensor EVM upgrades, modifications, or integrations beyond the existing deployment at the time of this assessment. Any changes to the Bittensor EVM protocol's logic after the time of this assessment may introduce new risks that are outside the scope of this assessment.

We suggest the Latent team maintain a structured plan to gradually update the smart contract logic in alignment with changes in Bittensor EVM behavior. For each new Bittensor EVM version, the updated contracts should undergo a dedicated security audit to ensure continued compatibility and safety.

Also, while the codebase is not large, the lack of test coverage suggests that additional preparation would be beneficial before mainnet deployment.

Key areas for enhancement

- Documentation is somewhat limited, with missing NatSpec and comments that, if added, would improve readability.
- Tests are currently nonfunctional and focus on single actions. The project could benefit from more multistep testing to verify correctness of sequential operations.

Recommended next steps

- Deploy to testnet for several weeks to validate intended behavior under real conditions.
- Expand test coverage by implementing multi-operation test scenarios.
- Consider improving code documentation to support long-term maintenance.

While the foundation is promising, implementing these recommendations would provide greater assurance for mainnet deployment.

5.1. 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.