Zellic

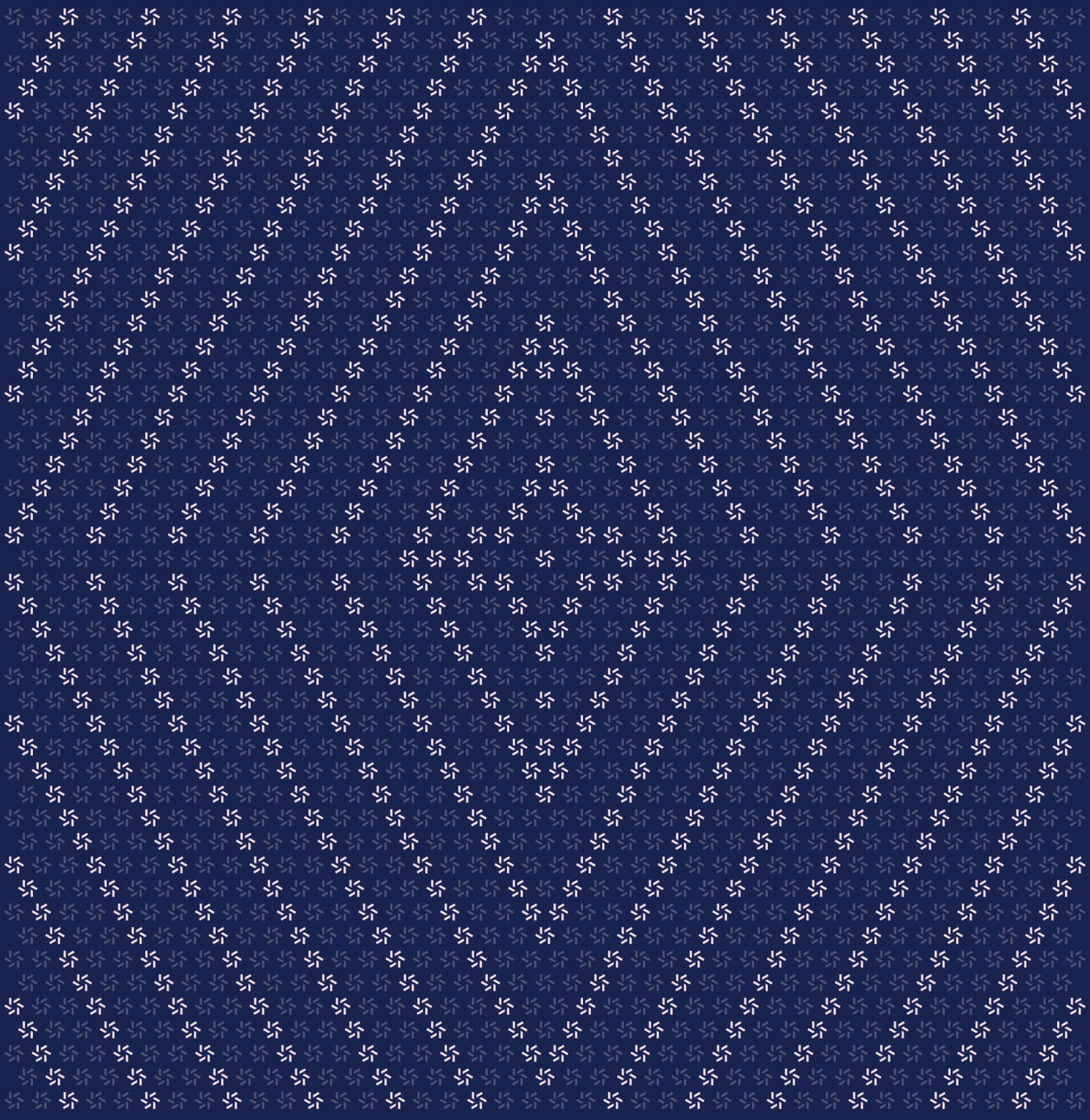
Prepared for
Yuwen Zhang
Taehoon Kim
Succinct

Prepared by
Qingying Jie
Juchang Lee
Quentin Lemauf
Zellic

July 30, 2025

OP Succinct

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 Succinct from July 16th to July 17th, 2025. During this engagement, Zellic reviewed OP Succinct'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:

- Could an unprivileged fallback be exploited to cause a denial of service to the chain?
- Could an unprivileged proposer be allowed to propose an incorrect output root?
- Is there a vulnerability in `OPSuccinctConfig`s that could allow an attacker to see proof of an invalid configuration?
- Could upgrading a contract break its state?

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 OP Succinct contracts, we discovered two findings, both of which were medium impact.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Succinct in the Discussion section (4. ↗).

Breakdown of Finding Impacts

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

2. Introduction

2.1. About OP Succinct

Succinct contributed the following description of OP Succinct:

> OP Succinct is a lightweight upgrade to the OP Stack that enables ZK-based finality. Specifically, it upgrades a single on-chain contract and the op-proposer component.

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:

OP Succinct Contracts

Type	Solidity
Platform	EVM-compatible
Target	Only changes between 46482d3f...530df863
Repository	https://github.com/succinctlabs/op-succinct ↗
Version	`530df8630562292a9629303528d51307df954aec`

Programs	`validity/OPSuccinctDisputeGame.sol`
	`validity/OPSuccinctL2OutputOracle.sol`
	`fp/AccessManager.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of two person-days. The assessment was conducted by three 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:

Qingying Jie
Engineer
qingying@zellic.io

Juchang Lee
Engineer
lee@zellic.io

Quentin Lemauf
Engineer
quentin@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

July 16, 2025	Start of primary review period
July 17, 2025	Kick-off call
July 17, 2025	End of primary review period

3. Detailed Findings

3.1. Anyone could submit a malicious output via the optimistic function `proposeL2Output`

Target	OPSuccinctL2OutputOracle		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

When optimistic mode is enabled, the function `proposeL2Output` without output verification will be used. If the current `block.timestamp` has exceeded the `fallbackTimeout` since the last proposal, then anyone can use this function to propose an output root.

```
function proposeL2Output(bytes32 _outputRoot, uint256 _l2BlockNumber,
    bytes32 _l1BlockHash, uint256 _l1BlockNumber)
    external
    payable
    whenOptimistic
{
    // The proposer must be explicitly approved, or the zero address must be
    approved (permissionless proposing),
    // or the fallback timeout has been exceeded allowing anyone to propose.
    require(
        approvedProposers[msg.sender] || approvedProposers[address(0)]
            || (block.timestamp - lastProposalTimestamp() > fallbackTimeout),
        "L2OutputOracle: only approved proposers can propose new outputs"
    );

    // [...]
}
```

Impact

A malicious user can submit an invalid output root when the function `proposeL2Output` falls back to permissionless mode. Although the challenger can later delete output roots that have not yet been finalized through the function `deleteL2Outputs`, if not deleted within a limited time, the invalid output root will no longer be removable.

Recommendations

According to the documentation ↗, permissionless mode is generally not enabled when optimistic mode is enabled:

> **Warning**: If you had permissionless proving enabled in non-optimistic (OP Succinct) mode, ensure that `address(0)` is set to `false` in the `approvedProposers` mapping before enabling optimistic mode. If `address(0)` remains approved, any account will be able to submit outputs without verification in optimistic mode.

However, given that the fallback scenario is intended to ensure business continuity, consider verifying the output when falling back to permissionless mode.

Remediation

Succinct has acknowledged this issue, and a fix was implemented in PR #575 ↗. Succinct has indicated that they plan to merge it into the production branch.

3.2. Creation of contract OPSuccinctDisputeGame could be blocked by front-running the output proposal

Target	OPSuccinctDisputeGame, OPSuccinctL2OutputOracle		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

When deploying the contract OPSuccinctDisputeGame, its function `initialize` will call the function `proposeL2Output` of the contract OPSuccinctL2OutputOracle to propose an output root.

```
// DisputeGameFactory
function create(
    GameType _gameType,
    Claim _rootClaim,
    bytes calldata _extraData
)
    external
    payable
    returns (IDisputeGame proxy_)
{
    // [...]
    proxy_ = IDisputeGame(address(impl).clone(abi.encodePacked(msg.sender,
    _rootClaim, parentHash, _extraData)));
    proxy_.initialize{ value: msg.value }();

    // [...]
}

// OPSuccinctDisputeGame
function initialize() external payable {
    // [...]
    OPSuccinctL2OutputOracle oracle
    = OPSuccinctL2OutputOracle(L2_OUTPUT_ORACLE);
    oracle.proposeL2Output(
        configName(), rootClaim().raw(), l2BlockNumber(), l1BlockNumber(),
    proof(), proverAddress()
    );

    this.resolve();
```

```
}
```

However, after one proposal, a certain interval must pass before another proposal can be made. Meanwhile, others can directly propose through the function `proposeL2Output`.

```
function proposeL2Output(
    bytes32 _configName,
    bytes32 _outputRoot,
    uint256 _l2BlockNumber,
    uint256 _l1BlockNumber,
    bytes memory _proof,
    address _proverAddress
) external whenNotOptimistic {
    // [...]
    require(
        approvedProposers[tx.origin] || approvedProposers[address(0)]
            || (block.timestamp - lastProposalTimestamp() > fallbackTimeout),
        "L2OutputOracle: only approved proposers can propose new outputs"
    );

    require(
        _l2BlockNumber >= nextBlockNumber(),
        "L2OutputOracle: block number must be greater than or equal to next
    expected block number"
    );

    // [...]
}
```

Impact

A malicious proposer can front-run the deployment transaction of the contract OPSuccinctDisputeGame by proposing an output root directly through the contract OPSuccinctL2OutputOracle, causing the contract OPSuccinctDisputeGame to fail to deploy.

Recommendations

Consider refactoring contracts OPSuccinctDisputeGame and OPSuccinctL2OutputOracle.

Remediation

Succinct has acknowledged this issue, and a fix was implemented in PR #575 ↗. Succinct has indicated that they plan to merge it into the production branch.

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. No function to update `verifier`

OP Succinct removed the function `updateVerifier` and event `VerifierUpdated` from the contract OPSuccinctL2OutputOracle.

```
event VerifierUpdated(address indexed oldVerifier,
    address indexed newVerifier);

function updateVerifier(address _verifier) external onlyOwner {
    emit VerifierUpdated(verifier, _verifier);
    verifier = _verifier;
}
```

Thus, there is no function to update the state variable `verifier`, which stored the address of the SP1Verifier contract. Consider adding this function and event back.

This issue has been acknowledged by Succinct, and a fix was implemented in PR #575 ↗.

5. Diff Scope

The purpose of this section is to document the exact diffs of the codebase that were considered in scope for this audit.

We focused on the changes made between commit 46482d3f ↗ and commit 530df863 ↗ of the codebase. The following sections describe notable changes made in the diff.

5.1. Multiconfiguration was introduced

This update significantly improves flexibility by moving to a system where multiple attestation configurations can be managed by name.

- The `OpSuccinctConfig` structure was introduced. The Succinct team has added an `OpSuccinctConfig` structure that ties together the `aggregationVkey`, `rangeVkeyCommitment`, and `rollupConfigHash` needed for attestation verification.
- The `opSuccinctConfigs` mapping was added. The `opSuccinctConfigs` mapping was introduced to store an `OpSuccinctConfig` with a configuration name in Bytes32 format as the key. This allows support for multiple versions of attestation systems or key rotation.
- Owner-management functionality was added. Owners can add or delete attestation settings via the `addOpSuccinctConfig` and `deleteOpSuccinctConfig` functions. This forms a centralized management point.
- The `proposeL2Output` function signature was changed. The `proposeL2Output` function, which proposes L2 output, now takes `_configName` as the first argument and validates the SP1 attestation with the appropriate verification key for that setting.

5.2. A fallback mechanism was added in case the proposer is disabled

To ensure liveness of the network, a fallback feature has been added in case an authorized proposer does not submit L2 output root in time.

- The `fallbackTimeout` state variable was added. The `fallbackTimeout` variable is set via the initialize function. This variable defines the maximum time, in seconds, to wait for a response from a proposer.
- The proposal logic was updated. The permission-checking logic within the `proposeL2Output` function has been modified. Anyone can now propose a new L2 output, even if they are not an approved proposer, as long as `fallbackTimeout` has elapsed since the last proposal. This prevents service interruptions due to censorship or disabling of a proposer.

6. Assessment Results

During our assessment on the scoped OP Succinct contracts, we discovered two findings, both of which were medium impact.

After all findings are resolved and tests are updated, including regression coverage, the code will appear to be better positioned for deployment.

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.