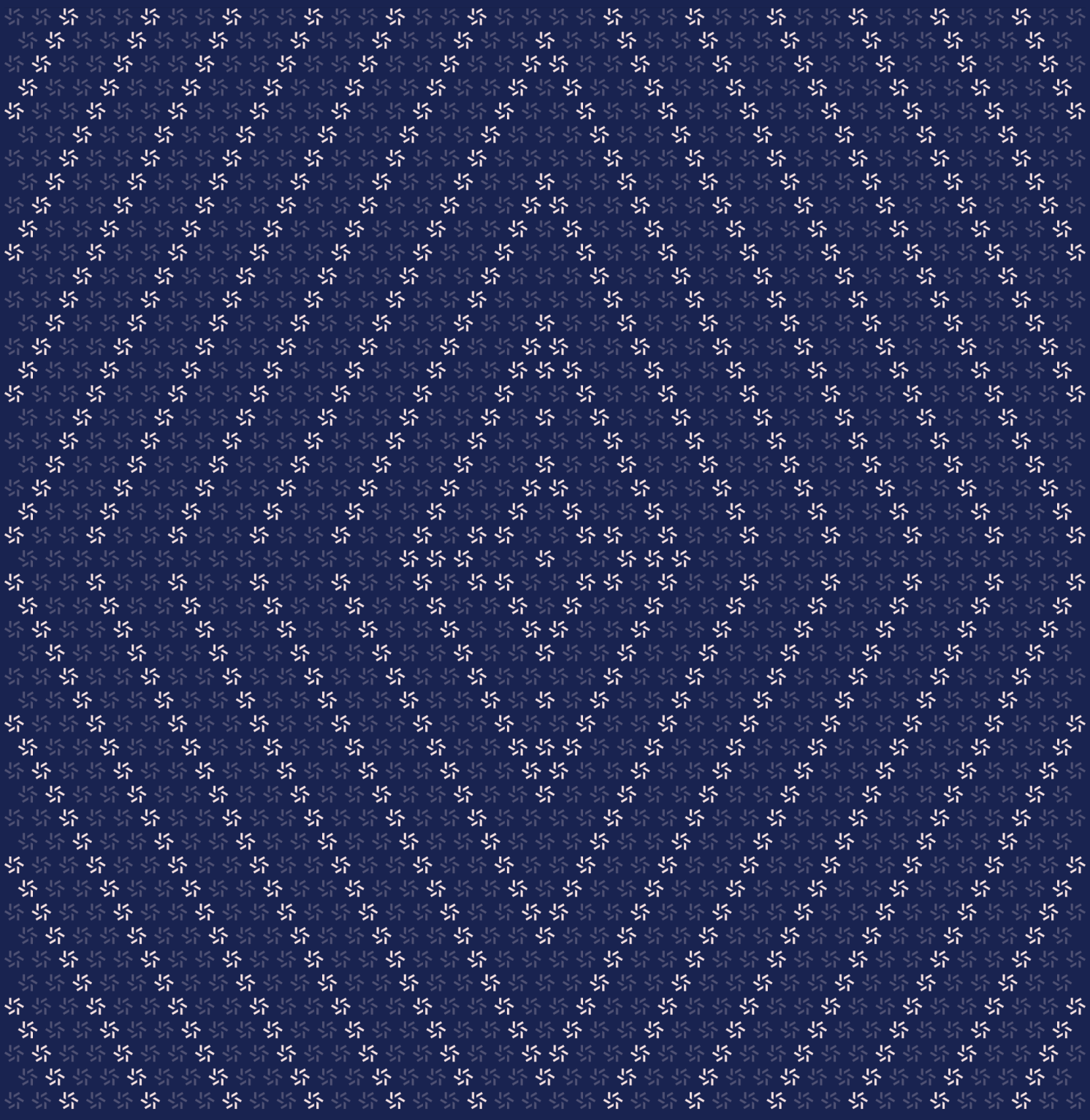
Zellic

Prepared for
Barny Bera
Berachain

Prepared by
Jinseo Kim
Quentin Lemauf
Zellic

February 13, 2026

Honey
Smart Contract Patch Review



Contents

About Zellic **3**

1. Overview **3**

1.1. Executive Summary 4

1.2. Results 4

2. Introduction **4**

2.1. Scope 5

2.2. Disclaimer 6

3. Detailed Findings **6**

3.1. Errors emitted by the `permit` function are changed with upgrade 7

3.2. The `version` function does not affect the version field of EIP-712 8

4. Discussion **9**

4.1. Unnecessary optimization on loop-index–increment overflow check 10

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 Berachain from February 11th to February 12th, 2026. During this engagement, Zellic reviewed Honey's code for security vulnerabilities, design issues, and general weaknesses in security posture.

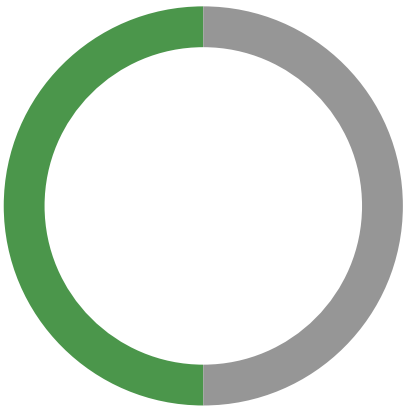
1.2. Results

During our assessment on the scoped Honey contracts, we discovered two findings. No critical issues were found. One finding was of low impact and the other finding was informational in nature.

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

Breakdown of Finding Impacts

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



2. Introduction

We were asked to review patches to Honey, which introduced EIP-2612 and EIP-3009 support and fixed minor issues around the way the contracts handle the dust amount.

2.1. Scope

The engagement involved a review of the following targets:

Honey Contracts

Type	Solidity
Platform	EVM-compatible
Target	contracts-internal
Repository	https://github.com/berachain/contracts-internal ↗
Version	`Diffs in PR #93 up until Git commit fb6098d89719165a4ce2efba7a93f984b4d3906f`

Programs	`EIP2612.sol`
	`EIP3009.sol`
	`Honey.sol`
	`HoneyFactory.sol`
	`HoneyFactoryPythWrapper.sol`
	`VaultAdmin.sol`

Contact Information

The following project manager was associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

The following consultants were engaged to conduct the assessment:

Jinseo Kim
Engineer
jinseo@zellic.io

Quentin Lemauf
Engineer
quentin@zellic.io

2.2. 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.

3. Detailed Findings

3.1. Errors emitted by the `permit` function are changed with upgrade

Target	EIP2612		
Category	Coding Mistakes	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

The Honey contract uses the ERC-20 implementation of Solady. It has the built-in `permit` function, which the changes in the scope of this audit override.

The previous `permit` function emits the `PermitExpired()` or `InvalidPermit()` errors, while the new `permit` function emits the error messages `"EIP2612: permit is expired"` or `"EIP2612: invalid signature"`.

Impact

The errors emitted by the `permit` function would be changed after the upgrade, which may be a breaking change to users, clients, or contracts that utilize the errors from the `permit` function.

Recommendations

Consider matching the errors returned by the new `permit` function to the ones Solady defined.

Remediation

This issue has been acknowledged by Berachain, and a fix was implemented in commit `85f955a9` ↗.

3.2. The `version` function does not affect the version field of EIP-712

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

Description

The Honey contract has the `version` function that returns the version field of the EIP-712 domain separator for the contract:

```
/// @notice Version string for the EIP712 domain separator
/// @return Version string
function version() external pure returns (string memory) {
    return "1";
}
```

However, this function is unused in the logic calculating the EIP-712 domain separator:

```
/// @dev `keccak256("1")`.
/// If you need to use a different version, override `_versionHash`.
bytes32 private constant _DEFAULT_VERSION_HASH =
    0xc89efdaa54c0f20c7adf612882df0950f5a951637e0307cdcb4c672f298b8bc6;

// (...)

/// @dev If you need a different value, override this function.
function _versionHash() internal view virtual returns (bytes32 result) {
    result = _DEFAULT_VERSION_HASH;
}

// (...)

/// @dev Returns the EIP-712 domain separator for the EIP-2612 permit.
function DOMAIN_SEPARATOR() public view virtual returns (bytes32 result) {
    // (...)
    bytes32 versionHash = _versionHash();
    // (...)
}
```

Impact

Changing the `version` function will not change the EIP-712 domain separator accordingly, which is a behavior that may cause confusion.

Recommendations

Consider overriding the `_versionHash` function to make the EIP-712 domain separator be calculated with the intended version.

Remediation

This issue has been acknowledged by Berachain, and a fix was implemented in commit `442096ce` ↗.

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. Unnecessary optimization on loop-index–increment overflow check

The VaultAdmin contract has the `setFeeReceiver` function, which sets the fee receiver for the contract. A fix added logic that migrates the accumulated fees to the new fee receiver.

```
function setFeeReceiver(address _feeReceiver) external {
    // (...)
    for (uint256 i = 0; i < registeredAssets.length;) {
        // (...)
        unchecked {
            ++i;
        }
    }
}
```

Note that Solidity optimizes out the overflow check on the loop index starting with version 0.8.22 ↗. It is unnecessary to wrap the loop-index increment with the `unchecked` block, considering that the code specifies to be compiled with Solidity 0.8.26.

This issue does not affect the business logic, but this suggestion could improve code readability by removing redundant optimizations.