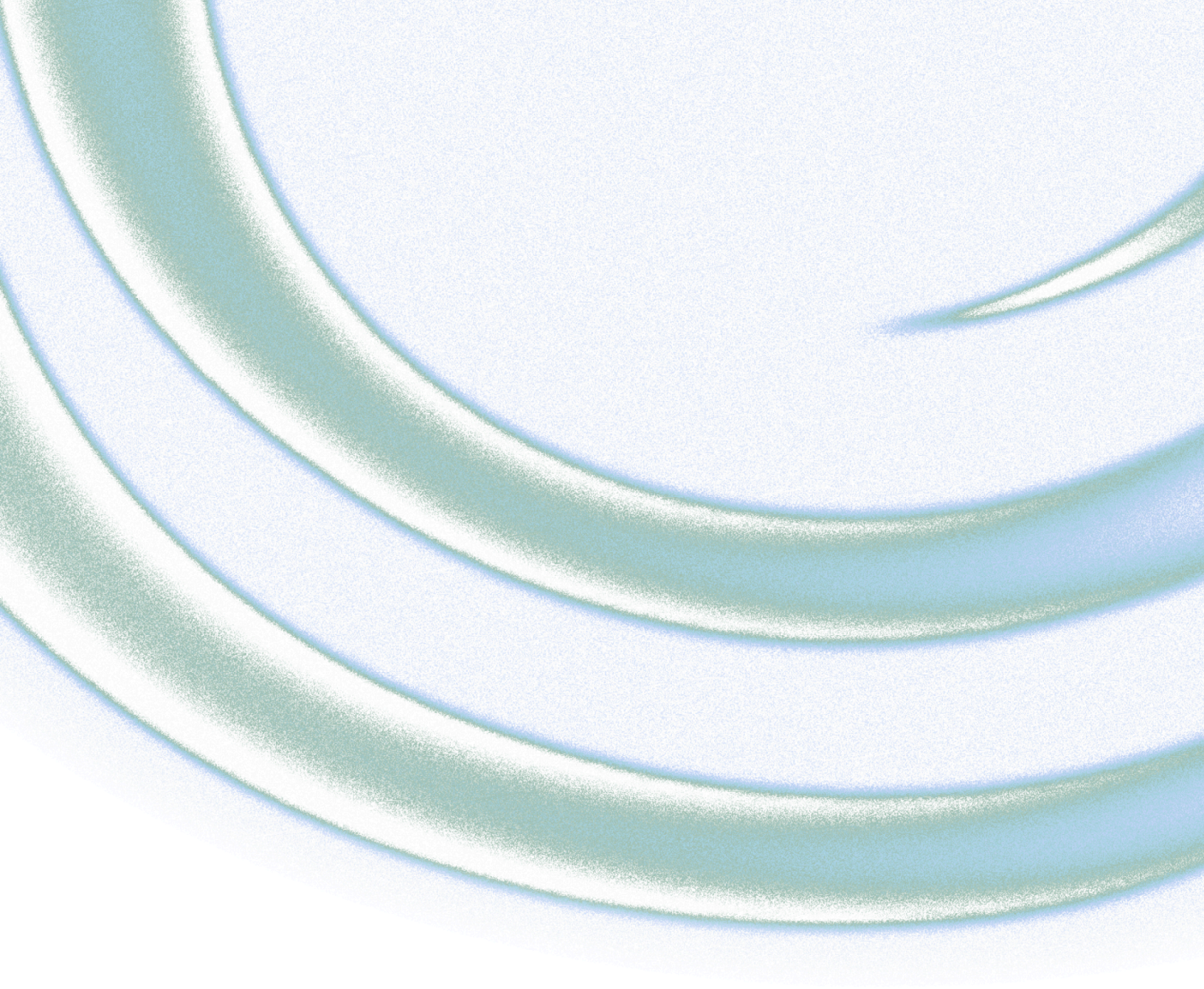


OtterSec

Hello Moon

Security Assessment

Prepared for
Hello Moon

Prepared by
Otter Audits, LLC

Prepared on
September 4th, 2026

Nicholas R. Putra

nicholas@osec.io

Table of Contents

1. Executive Summary 2
1.1. Overview 2
1.2. Key Findings 2
2. Scope 3
3. Findings 4
3.1. Findings Summary 4
3.2. Severity by Component 4
4. Advisories 5
OS-HLM-ADV-00 Ambiguous Pool Detection Misroutes Swap Fees 6
OS-HLM-ADV-01 Compact EIP-2098 Signatures Always Rejected 7
OS-HLM-ADV-02 Partial Swaps Incur Excessive Platform Fees 9
OS-HLM-ADV-03 Inability to Recover Residual Router Funds 10
OS-HLM-ADV-04 Unused Gas Penalty Bypasses Token Settlement 11
5. Suggestions 12
OS-HLM-SUG-00 Lack of Slippage Protection 13
OS-HLM-SUG-01 Absence of Deadline Protection on Router Execution 14
A. Vulnerability Rating Scale 15
B. Procedure 16

© 2026 Otter Audits LLC. All Rights Reserved.

1. Executive Summary

1.1. Overview

Hello Moon engaged OtterSec to conduct a security assessment of the `SignedTokenPaymaster`, `InkLaunchpad`, and `Calamari` programs. This assessment was conducted between September 1st and September 4th, 2026. For information on our auditing methodology, refer to Appendix B.

1.2. Key Findings

We produced 7 findings throughout this audit engagement.

In particular, when both pool assets are recognized as launched tokens, the fee hook may misidentify the newly launched base token as the quote asset, resulting in swap fees being collected in the wrong token (● OS-HLM-ADV-00).

Additionally, the paymaster's decoder accepts 64-byte EIP-2098 signatures, but its selected `tryRecover` overload supports only 65-byte signatures. As a result, all otherwise valid compact-signature quotes are rejected (● OS-HLM-ADV-01). Furthermore, the fee hook charges quote-specified swaps based on the requested amount without adjusting for partial fills. Consequently, the effective fee may exceed 1% of the quote amount actually executed (● OS-HLM-ADV-02).

Also, exact-output swaps may leave unused ETH or ERC-20 input stranded in the router due to the lack of a residual refund mechanism, allowing other users to later extract these shared balances (● OS-HLM-ADV-03).

We further highlighted an issue where the paymaster settles the user's token charge before the EntryPoint applies its unused-gas penalty, resulting in the paymaster absorbing the penalty through its native deposit without recovering it from the user (● OS-HLM-ADV-04).

We also suggested adding a slippage check to ensure the buyer does not receive fewer tokens than expected, without the transaction reverting (● OS-HLM-SUG-00). Finally, we advise adding deadline protection for router execution (● OS-HLM-SUG-01).

© 2026 Otter Audits LLC. All Rights Reserved.

2. Scope

`SignedTokenPaymaster` enables ERC-4337 smart accounts to pay transaction fees using allowlisted ERC-20 tokens while covering the actual native gas cost from its EntryPoint ETH deposit. It validates signed quotes, precharges a capped token amount, then settles the final charge after execution and credits any excess as a claimable refund.

`InkLaunchpad` creates fixed-supply ERC-20 tokens and sells most of their supply through a constant-product bonding curve using ETH or supported ERC-20 assets. Once the sale completes, it automatically migrates the collected funds and reserved tokens into a protected Uniswap v4 pool for continued trading.

`Calamari` provides a Uniswap v4 execution layer for swaps and settlements using native ETH, WETH, or Permit2-approved ERC-20 tokens. It combines wrapped-ETH support with a reentrancy-protected router that securely pays the PoolManager.

The source code was delivered to us in Git repositories at:

- `https://github.com/witches-brew/witches-brew-signed-token-paymaster-audit-ink`.
- `https://github.com/witches-brew/InkLaunchpad`.
- `https://github.com/witches-brew/Calamari`.

The audit was performed against the following commits:

- 6b3c87c.
- bf02642.
- dc06734.

© 2026 Otter Audits LLC. All Rights Reserved.

3. Findings

Findings were rated using the criteria in Appendix A.

We split the findings into **advisories** and **suggestions**. Advisories have an immediate impact and should be remediated as soon as possible. Suggestions do not have an immediate impact but will aid in mitigating future vulnerabilities.

Items are tracked with a stable identifier: `OS-HLM-ADV-*` for advisories and `OS-HLM-SUG-*` for suggestions.

3.1. Findings Summary

Severity	Count
Critical	0
High	0
Medium	5
Low	0
Informational	2

3.2. Severity by Component

Component	Critical	High	Medium	Low	Info	Total
Paymaster	—	—	2	—	—	2
InkLaunchpad	—	—	2	—	1	3
Calamari	—	—	1	—	1	2
Total	0	0	5	0	2	7

© 2026 Otter Audits LLC. All Rights Reserved.

4. Advisories

Here, we present a technical analysis of the 5 vulnerabilities we identified during our audit. These vulnerabilities have **immediate** security implications, and we recommend remediation as soon as possible.

ID	Severity	Status	Description
OS-HLM-ADV-00	Medium	✓ Resolved	Ambiguous Pool Detection Misroutes Swap Fees
OS-HLM-ADV-01	Medium	✓ Resolved	Compact EIP-2098 Signatures Always Rejected
OS-HLM-ADV-02	Medium	✓ Resolved	Partial Swaps Incur Excessive Platform Fees
OS-HLM-ADV-03	Medium	✓ Resolved	Inability to Recover Residual Router Funds
OS-HLM-ADV-04	Medium	✓ Resolved	Unused Gas Penalty Bypasses Token Settlement

© 2026 Otter Audits LLC. All Rights Reserved.

Ambiguous Pool Detection Misroutes Swap Fees

ID	Severity	Status
OS-HLM-ADV-00	Medium	Resolved

Description

`InkFeeHook::_quoteCurrency` infers the quote asset by checking whether either pool currency is a factory-launched token and treating the opposite side as the quote. When a previously launched token is reused as the quote currency, both pool assets satisfy `isLaunchedToken`, making this inference ambiguous and dependent on currency address ordering. If the older quote token is `currency0`, the hook incorrectly identifies the newly launched base token as the quote asset. Consequently, swap fees are charged and accumulated in the base token instead of the configured quote currency.

src/hooks/InkFeeHook.sol

```
215 function _quoteCurrency(PoolKey calldata key) private view returns (Currency quote,
    bool found) {
216     if (factory.isLaunchedToken(Currency.unwrap(key.currency0))) {
217         return (key.currency1, true);
218     }
219     if (factory.isLaunchedToken(Currency.unwrap(key.currency1))) {
220         return (key.currency0, true);
221     }
222     return (Currency.wrap(address(0)), false);
223 }
```

Remediation

Reject a factory launched token in `addQuoteCurrency` and reject a launch whose predicted `CREATE2` address is already an allowlisted quote.

Patch

Resolved in PR#20.

© 2026 Otter Audits LLC. All Rights Reserved.

Compact EIP-2098 Signatures Always Rejected

ID	Severity	Status
OS-HLM-ADV-01	Medium	Resolved

Description

In `SignedTokenPaymaster`, `_decodeQuote` explicitly accepts both 64-byte EIP-2098 and 65-byte ECDSA signatures, implying that either format is valid.

src/SignedTokenPaymaster.sol

```
function _decodeQuote(bytes calldata data) internal pure returns (Quote memory quote) {
    if (data.length != 320 && data.length != 352) revert InvalidPaymasterData();
    (
        [...]
    ) = abi.decode(data, (address, uint192, uint256, uint48, uint48, bytes32,
    bytes));
    if (
        quote.token == address(0)
            || (quote.signature.length != 64 && quote.signature.length != 65)
    ) {
        revert InvalidPaymasterData();
    }
}
```

However, `_validateQuote` passes the raw signature bytes to an OpenZeppelin `ECDSA.tryRecover` overload that accepts only 65-byte signatures, causing every 64-byte compact signature to fail recovery. Consequently, valid EIP-2098 quotes are consistently rejected despite passing the paymaster's initial format validation.

src/SignedTokenPaymaster.sol

```
function _validateQuote(PackedUserOperation calldata userOp, Quote memory quote)
    internal
    view
    returns (uint256 validationData, bool signatureFailed)
{
    _validateQuoteBounds(quote);
    TokenConfig memory config = tokenConfigs[quote.token];
    if (!config.enabled) revert TokenNotEnabled(quote.token);
    if (quote.unitsPerEth < config.minUnitsPerEth || quote.unitsPerEth >
    config.maxUnitsPerEth)
    {
        revert QuoteRateOutOfBounds(
            quote.unitsPerEth, config.minUnitsPerEth, config.maxUnitsPerEth
        );
```

© 2026 Otter Audits LLC. All Rights Reserved.

```
252         }
253
254         (address recovered, ECDSA.RecoverError recoverError,) =
255             ECDSA.tryRecover(_quoteDigest(userOp, quote), quote.signature);
256         signatureFailed = recoverError != ECDSA.RecoverError.NoError || recovered !=
            quoteSigner;
257         validationData = _packValidationData(signatureFailed, quote.validUntil,
            quote.validAfter);
258     }
```

Remediation

Decode 64-byte signatures into `r` and `vs` and use OpenZeppelin's EIP-2098-compatible `tryRecover` overload.

Patch

Resolved in `87fb063`.

© 2026 Otter Audits LLC. All Rights Reserved.

Partial Swaps Incur Excessive Platform Fees

ID	Severity	Status
OS-HLM-ADV-02	Medium	Resolved

Description

In `InkFeeHook`, for quote-specified swaps, `beforeSwap` calculates the platform fee from `abs(params.amountSpecified)`, representing the requested rather than executed quote amount. If a price limit or insufficient liquidity causes a partial fill, the final quote volume can be smaller while the precomputed fee remains unchanged. Because `InkFeeHook` does not reconcile this fee against the resulting `BalanceDelta`, the effective charge may exceed the intended 1% rate. Traders may therefore be overcharged whenever a quote-specified swap executes only partially.

Remediation

Reject partial fills by comparing the requested specified-side amount against the realised delta in `afterSwap` and reverting on a shortfall.

Patch

Resolved in PR#20.

© 2026 Otter Audits LLC. All Rights Reserved.

Inability to Recover Residual Router Funds

ID	Severity	Status
OS-HLM-ADV-03	Medium	Resolved

Description

Exact-output swaps may consume less than the user's supplied maximum input, but `InkV4Router::execute` provides no mechanism to return the unused balance. Surplus native ETH sent through `msg.value`, or ERC-20 tokens held by the router when it acts as payer, consequently remain pooled in the router without attribution to the original user. Because subsequent executions can use the router itself as the payer, another caller may issue router-funded `SETTLE` and `TAKE` actions to convert those residual assets into a withdrawal.

Remediation

Refund all unused ETH and ERC-20 inputs to the original caller before completing each execute operation.

Patch

Resolved in PR#4.

© 2026 Otter Audits LLC. All Rights Reserved.

Unused Gas Penalty Bypasses Token Settlement

ID	Severity	Status
OS-HLM-ADV-04	Medium	Resolved

Description

`SignedTokenPaymaster::_postOp` calculates the ERC-20 reimbursement using the `actualGasCost` supplied before the `EntryPoint` applies its unused-gas penalty. After `postOp` returns, the `EntryPoint` adds a 10% charge on unused execution gas and deducts the resulting cost from the paymaster's native deposit. Because this later penalty is absent from the token-settlement calculation, the corresponding cost is not recovered from the user. Operations with overallocated gas limits can therefore systematically underpay in tokens while depleting the paymaster's native deposit.

Remediation

Include the `EntryPoint`'s unused-gas penalty in the token precharge and settlement calculation so the paymaster fully recovers its native gas expenditure.

Patch

Resolved in `87fb063`.

© 2026 Otter Audits LLC. All Rights Reserved.

5. Suggestions

Here, we present a discussion of the 2 suggestions we identified during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Severity	Status	Description
OS-HLM-SUG-00	Informational	✓ Resolved	Lack of Slippage Protection
OS-HLM-SUG-01	Informational	✓ Resolved	Absence of Deadline Protection on Router Execution

© 2026 Otter Audits LLC. All Rights Reserved.

Lack of Slippage Protection

ID	Severity	Status
OS-HLM-SUG-00	Informational	✓ Resolved

Description

`InkFactory::launchTokenAndBuy` forwards the initial purchase to `InkCurve::buy` with `minAmountOut` set to zero, preventing the buyer from bounding the received token amount. If the effective curve parameters, fee configuration, or execution result differs from the buyer's expectation, the transaction can still succeed with an unfavorable output.

src/factory/InkFactory.sol

```
185 function launchTokenAndBuy(LaunchTokenArgs memory args, uint256 buyAmount)
186     external
187     payable
188     override
189     nonReentrant
190     whenNotPaused
191 {
192     address newCurve = _launch(args);
193     _buy(newCurve, buyAmount);
194 }
```

Remediation

Add slippage protection to ensure the buyer receives at least the expected minimum token amount.

Patch

Resolved in PR#20.

© 2026 Otter Audits LLC. All Rights Reserved.

Absence of Deadline Protection on Router Execution

ID	Severity	Status
OS-HLM-SUG-01	Informational	✓ Resolved

Description

`InkV4Router::execute` accepts encoded actions without enforcing a transaction deadline, allowing a pending transaction to remain executable indefinitely. Even when individual swaps include slippage limits, delayed execution may occur under stale market conditions or affect actions without equivalent protections.

src/InkV4Router.sol

```
17  function execute(bytes calldata unlockData) external payable isNotLocked {
18      _executeActions(unlockData);
19  }
```

Remediation

Add a deadline-aware overload that reverts when `block.timestamp` exceeds the caller-specified expiry.

Patch

Resolved in PR#4.

© 2026 Otter Audits LLC. All Rights Reserved.

Appendix A.

Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the general findings.

Severity	Description
Critical	Vulnerabilities that immediately result in a loss of availability, consensus divergence, or exploitable memory corruption with minimal preconditions.
High	Vulnerabilities that may result in consensus divergence or denial of service but require specific transaction or network conditions to exploit.
Medium	Vulnerabilities that may result in incorrect behavior or state divergence under specific configurations or edge cases.
Low	Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.
Informational	Best practices to mitigate future security risks. These are classified as general findings.

Resolution status is one of ✓ Resolved, ○ Ack'd, or × To-do.

© 2026 Otter Audits LLC. All Rights Reserved.

Appendix B.

Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

The design review asks whether the programs's architecture and operating assumptions support its intended behavior. We study the inputs it accepts, the decisions it makes, the information shared between components, and the services or participants it trusts. We then consider how it behaves when those inputs are unexpected or conflicting. The goal is to find conditions that can produce an incorrect decision, inconsistent state, or loss of service.

Design problems often arise from interactions rather than one defective function. Two components may each process a value correctly but interpret it differently. A failure may also leave one component ready to continue while another treats the same operation as incomplete. Local checks can look correct in both cases, so we follow the complete operation across component boundaries.

The implementation review asks whether the code matches the design. We trace untrusted data from its entry point to the point where it affects behavior, checking the assumptions made along the way. The review covers validation, error handling, state updates, concurrency, persistence, resource use, and cleanup. Its exact focus depends on the target's language, runtime, and architecture rather than a fixed list of bug classes.

The boundary between design and implementation is not exact. Missing validation may be a local coding error, while the same omission across several components may reflect an unclear rule. We classify a finding by its root cause, but judge its impact from the resulting behavior and the conditions needed to trigger it.

We begin by reading the target as a team and agreeing on its main components, trust boundaries, and expected behavior. Auditors then review different paths through the system and compare notes where those paths meet. We reproduce suspected issues with focused tests or traces when the setup allows. Before reporting a finding, another auditor reviews its trigger, impact, and proposed remediation.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process

© 2026 Otter Audits LLC. All Rights Reserved.