SALUS SECURITY

MAR 2025

CODE SECURITY ASSESSMENT

NATIVE ORG

Overview

Project Summary

- Name: Native org - Native v3
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/Native-org/v3-core/
    - https://github.com/Native-org/v3-periphery/
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Native org - Native v3
Version	v2
Type	Solidity
Dates	Mar 24 2025
Logs	Mar 24 2025; Mar 24 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	1
Total	3

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Uninitialized factory leading to front-running attack	Low	Configuration	Acknowledged
2	Incomplete whitelist implementation in NativeV3Pool	Low	Business Logic	Acknowledged
3	Missing two-step transfer ownership pattern	Informational	Business logic	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Uninitialized factory leading to front-running attack	
Severity: Low	Category: Configuration
Target: - v3-core/contracts/NativeV3PoolDeployer.sol	

Description

The NativeV3PoolDeployer contract's `setFactory` function can be invoked by anyone as long as the factory variable is unset (i.e., still equal to zero).

v3-core/contracts/NativeV3PoolDeployer.sol:L30-L36

```
function setFactory(address _factory) external {
   require(factory == address(0), 'already initialized');

   factory = _factory;

   emit SetFactory(_factory);
}
```

This lack of access restriction enables an attacker to front-run the legitimate initialization by calling `setFactory` before the proper initializer, thereby setting themselves as the factory. Once the factory is set to an attacker-controlled address, the attacker gains exclusive access to the `deploy` function (guarded by the `onlyFactory` modifier), which they can abuse to deploy pool contracts with arbitrary parameters. This vulnerability effectively renders the NativeV3PoolDeployer contract unusable for the intended parties and undermines the protocol's deployment integrity.

Recommendation

Consider deploying the contract with the factory address pre-initialized or using constructor parameters to prevent any window of vulnerability.

Status

This issue has been acknowledged by the team.

2. Incomplete whitelist implementation in NativeV3Pool	
Severity: Low	Category: Business Logic
Target: - v3-core/contracts/NativeV3Pool.sol	

Description

The NativeV3Pool contract declares a public mapping `` `mapping(address => bool) public isWhitelisted;` `` intended for managing a whitelist;

v3-core/contracts/NativeV3Pool.sol:L108-L109

```
/// @notice Mapping to track addresses in the whitelist
mapping(address => bool) public isWhitelisted;
```

However, this mapping is never referenced or enforced in any function. Instead, access control for swaps is solely implemented via a single address check using the `` `swapper` `` variable.

v3-core/contracts/NativeV3Pool.sol:L555-L739

```
/// @inheritdoc INativeV3PoolActions
function swap(
   address recipient,
   bool zeroForOne,
   int256 amountSpecified,
   uint160 sqrtPriceLimitX96,
   bytes calldata data
) external override returns (int256 amount0, int256 amount1) {
   require(amountSpecified != 0, "AS");
   require(swapper == address(0) || swapper == msg.sender, "NWS");
   ...
}
```

As a result, the intended granular whitelist functionality is effectively bypassed.

Recommendation

Integrate the `` `isWhitelisted` `` mapping within the `` `swap` `` logic (or any other relevant functions) to enforce a comprehensive whitelist mechanism.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - v3-core/contracts/NativeV3Factory.sol	

Description

The `` `NativeV3Factory` `` contract uses the `` `setOwner` `` function which is a simple mechanism to transfer the ownership not supporting a two-step transfer ownership pattern. This simpler mechanism can be useful for quick tests, but projects with production concerns are likely to outgrow it. Transferring ownership is a critical operation and this could lead to transferring it to an inaccessible wallet or renouncing the ownership, e.g. mistakenly.

Recommendation

It is recommended to implement a two-step transfer of ownership mechanism where the ownership is transferred and later claimed by a new owner to confirm the whole process and prevent lockout.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covers the files in Native-org/v3-core and Native-org/v3-periphery The repository is built on Uniswap V3, and this audit specifically examines the modifications made in this repository to Uniswap V3 Core at d8b1c63 commit and Uniswap V3 Periphery at 0682387 commit.