Aperture UniV3Automan

Incremental Security Testing and Assessment

July 3, 2023

Prepared for Aperture

Narya.ai

Table of Contents

Summary

Overview

From June 29, 2023, to July 3, 2023, Aperture engaged Narya.ai to evaluate the security of its code changes to the UniV3Automan contract in the GitHub repository: https://github.com/Aperture-Finance/uniswap-v3-automan (commit 0edbc07deca05901d1561a3259a228321250ff8f) and the uni-v3-lib library that hosts the utility code for Uniswap V3: https://github.com/Aperture-Finance/uni-v3-lib (commit b7043e3ab3c578e0705e15dce09374ba4497c96e).

The main changes since the last code review include the following:

- Refactoring of the UniV3Automan contract as well as library contacts, particularly OptimalSwap.
- Refactoring of Uniswap V3 utility code into a separate GitHub repository called uni-v3-lib.
- Added security measures to prevent ERC20 tokens from being whitelisted as a Router.

Project Scope

We reviewed and tested the UniV3Automan contract and its imported library uni-v3-lib. Other security-critical components of UniV3Automan, such as off-chain services and the web front end, are not included in the scope of this security assessment. We recommend a further review of those components.

Summary of Findings

Severity	# of Findings
High	0
Medium	0
Low	0
Informational	0
Gas	0
Total	0

Throughout our review and testing, we did not identify any new security vulnerabilities in the UniV3Automan contract and the uni-v3-lib library.

Summary of Tests

We have implemented 3 property tests and 5 invariants to test the following key logic for the UniV3Automan contracts.

We have identified no issues throughout the testing. All the tests have passed at commit 0edbc07deca05901d1561a3259a228321250ff8f.

Table 1. Tests of UniV3Automan

Test Name	Tested Key Logics	Result
testDrainETH	The minting function will not use the contract's balance of native ether to mint an NFT when the user doesn't send any funds.	Pass
testLeftover	When minting a new NFT, the Automan contract should send back the excess amount of tokens paid by the user.	Pass
testMintAndBurn	For any amount of valid liquidity to add, a new NFT can be minted. When transferred to another account, the new owner should be able to burn the NFT and get back the funds held by it.	Pass
invariantMint	When a mint operation is successful, a new NFT should be given to the user and his amount of tokens should decrease.	Pass
invariantRemove	When a removal operation is successful, the user should receive tokens and have one less NFT.	Pass
invariantRebalance	When a rebalance operation is successful, the user should receive a new NFT as the rebalance operation does not burn the NFT position.	Pass
invariantIncreaseLiquidity	When increasing the liquidity of an NFT position is successful, the user should have the same amount of NFTs, but the concerned NFT position should have more liquidity.	Pass
invariantDecreaseLiquidity	When decreasing the liquidity of an NFT position is successful, the user should have the same amount of NFTs, but the concerned NFT position should have less liquidity.	Pass

Disclaimer

This report should not be used as investment advice.

Narya.ai uses an automatic testing technique to test smart contracts' security properties and business logic rapidly and continuously. However, we do not provide any guarantees on eliminating all possible security issues. The technique has its limitations: for example, it may not generate a random edge case that violates an invariant during the allotted time. Its use is also limited by time and resource constraints.

Unlike time-boxed security assessment, Narya.ai advises continuing to create and update security tests throughout the project’s lifetime. In addition, Narya.ai recommends proceeding with several other independent audits and a public bug bounty program to ensure smart contract security.

Testing Specifications and Results

The test results are against the latest commit [0edbc07deca05901d1561a3259a228321250ff8f]

UniV3Automan

Property Tests

testDrainETH ✅

Tested key logic: The minting function will not use the contract's balance of native ether to mint an NFT when the user doesn't send any funds.

Tested user journey:

- Give the Automan contract an amount of native ether.
- User: approve the maximum amount of tokens for the Automan contract.
- User: attempt to mint an NFT position without sending any funds.

Test checks:

- The minting fails due to the user not sending any funds.
- The funds in the Automan contract are not drained.

testLeftOver ✅

Tested key logic: When minting a new NFT, the Automan contract should send back the excess amount of tokens paid by the user.

Tested user journey:

- Give the user some tokens for adding liquidity and some native ether.
- User: approve the maximum amount of tokens for the Automan contract.
- User: attempt to mint an NFT position by sending 10 native ethers.

Test checks:

- The minting fails due to the amount of ether not corresponding to the expected amount for the minting operation.
- The Automan contract has no amount of native ether.

testMintAndBurn ✅

Tested key logic: For any amount of valid liquidity to add, a new NFT can be minted. When transferred to another account, the new owner should be able to burn the NFT and get back the funds held by it.

Tested user journey:

- Give the user some tokens for adding liquidity.
- User: mint a new NFT position by adding liquidity.
- User: transfer the NFT to user2.
- User2: remove all the liquidity for the NFT.

Test checks:

- The owner of the NFT is User after the mint.
- The owner of the NFT is User2 after the transfer.
- User2 has one less NFT after the burn.
- User2 got back funds from burning the NFT.

Invariant Tests

invariantMint ✅

Tested key logic: When a mint operation is successful, a new NFT should be given to the user and his amount of tokens should decrease.

Invariant setup:

- Owner: deploy UniV3Automan.
- Owner: set the fees configuration and the controllers.

Invariant checks:

- The amount of NFTs increases by 1.
- User tokens balances decreased.

invariantRemove ✅

Tested key logic: When a removal operation is successful, the user should receive tokens and have one less NFT.

Invariant setup:

- Owner: deploy UniV3Automan.
- Owner: set the fees configuration and the controllers.

Invariant checks:

- The amount of NFTs decreases by 1.
- The User's tokens balances increased.

invariantRebalance ✅

Tested key logic: When a rebalance operation is successful, the user should receive a new NFT as the rebalance operation does not burn the NFT position.

Invariant setup:

- Owner: deploy UniV3Automan.
- Owner: set the fees configuration and the controllers.

Invariant checks:

- The amount of NFTs increases by 1.

invariantIncreaseLiquidity ✅

Tested key logic: When increasing the liquidity of an NFT position is successful, the user should have the same amount of NFTs, but the concerned NFT position should have more liquidity.

Invariant setup:

- Owner: deploy UniV3Automan.
- Owner: set the fees configuration and the controllers.

Invariant checks:

- The amount of NFTs stays the same.
- The liquidity for the concerned NFT position should increase.

invariantDecreaseLiquidity ✅

Tested key logic: When decreasing the liquidity of an NFT position is successful, the user should have the same amount of NFTs, but the concerned NFT position should have less liquidity.

Invariant setup:

- Owner: deploy UniV3Automan.
- Owner: set the fees configuration and the controllers.

Invariant checks:

- The amount of NFTs stays the same.
- The liquidity for the concerned NFT position should decrease.

Relevant Past Hacks

We have identified the following publicly known past hacks that are relevant to the UniV3Automan contract which is similar in functionality to DEXs or DEX aggregators in general.

Pre-Approval Token Theft

Threat Model: The attacker leveraged an arbitrary function call to steal pre-approved tokens to the vulnerable contract.

Sushiswap

Link: https://github.com/SunWeb3Sec/DeFiHackLabs/blob/main/src/test/Sushi_Router_exp.sol

Dexible

Link: https://github.com/SunWeb3Sec/DeFiHackLabs/blob/main/src/test/Dexible_exp.sol

Rubic

Link: https://github.com/SunWeb3Sec/DeFiHackLabs/blob/main/src/test/Rubic_exp.sol

CowSwap

Link: https://github.com/SunWeb3Sec/DeFiHackLabs/blob/main/src/test/CowSwap_exp.sol

RabbyWallet

Link: https://github.com/SunWeb3Sec/DeFiHackLabs/blob/main/src/test/RabbyWallet_SwapRouter.exp.sol

SwapX

Link: https://github.com/SunWeb3Sec/DeFiHackLabs/blob/main/src/test/SwapX_exp.sol

Code Review ✅

- The issue was previously reported to Aperture which only concerns itself with Uniswap V3. The whitelist function can only be called by the owner of the UniV3Automan contract and the contract does not hold any funds. Since the Uniswap V3 router will only pull tokens from the sender, only the exact amount of tokens for the trade will be used. For these reasons, we do not believe that the current contract is vulnerable to this bug class.

Fix Log

ID	Title	Severity	Status
N/A*			

* We didn’t find any issues in this incremental review and testing of the UniV3Automan project.

Appendix

Severity Categories

Severity	Description
Gas	Gas optimization.
Informational	The issue does not pose a security risk but is relevant to best security practices.
Low	The issue does not put assets at risk such as functions being inconsistent with specifications or issues with comments.
Medium	The issue puts assets at risk not directly but with a hypothetical attack path, stated assumptions, or external requirements. Or the issue impacts the functionalities or availability of the protocol.
High	The issue directly results in assets being stolen, lost, or compromised.