HashEx

Autonomy Network

smart contracts audit report

Prepared for:

autonomynetwork.io

Authors: HashEx audit team

September 2021

Contents

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (https://hashex.org).

Introduction

HashEx was commissioned by Apeswap Finance to perform an audit of Autonomy's smart contracts. The audit was conducted between August 31 and September 08, 2021.

The code located in private GitHub repositories @autonomy-network/autonomy-contracts @autonomy-network/uniV2-limits-stops was audited after the 4d7dcd1 and 6ac5be5 commits. Repositories contain tests for audited contracts, documentation was provided via team's website and Medium. It must be noted that the Miner.sol contract was not in the scope of this audit.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts.
- Formally check the logic behind given smart contracts.

Information in this report should be used to understand the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

Update. The Autonomy team has responded to this report. Individual responses were added after each item in the section. The updated contracts are deployed to the Binance Smart Chain (BSC):
0x957Fa92cAc1AD4447B6AEc163af57e7E36537c91 PriceOracle,
0x3831ff695ddf9f792F2202a9e3121f3880711d87 Oracle,
0xde946E11A1F06F58bA0429dAfAaabE6Ec1C7D498 StakeManager,
0xcE675B50034a2304B01DC5e53787Ec77BB7965D4 Forwarder (UserForwarder),
0xE390b2436df1fE909628fa5eB8f53d041D7B2c93 Forwarder (GasForwarder),
0x4F54277e6412504EBa0B259A9E4c69Dc7EE4bB9c Forwarder (UserGasForwarder),
0x18d087F8D22D409D3CD366AF00BD7AeF0BF225Db Registry,
0x151394FBa85A10E7A669f07818aC408b9abb8e09 UniV2LimitsStops.

Ownership of UniV2LimitsStops, PriceOracle, Oracle and 3 Forwarders was transferred to the Timelock contract with 48 hours of minimum delay:
0x9Ce05ad236Ad29B9EF6597633201631c097c3f10.

AUTO token contract hasn't been deployed, but it's address was set to 0x62d68c1dfD5e570D258C40bE204871877F5eE6a0 in deployed Registry and StakeManager contracts.

Contracts overview

`AUTO.sol`

ERC777 [1] token with minting under the owner's control.

`Forwarder.sol`

Proxy contract with restricted access.

`Registry.sol`

Registry of the requests, translates executions to Forwarder contracts.

`Oracle.sol & PriceOracle.sol`

Simple price oracle to be operated by the owner.

`StakeManager.sol`

Staking contract for ERC777 token.

`uniV2LimitsStops.sol`

Limit orders and stop losses contract to be called via Forwarder proxies.

Found issues

ID	Title	Severity	Response
01	`uniV2LimitsStops: stop losses aren’t guaranteed`	High	Responded
02	`uniV2LimitsStops: malicious owner can set parameters via setDefaultFeeInfo()`	High	Fixed
03	`Forwarder: owner regulates access`	High	Fixed
04	`StakeManager: unstake() indexation problem`	Medium	Fixed
05	`Registry: execution can be sabotaged`	Medium	Responded
06	`Oracle: subjected to mining attack on random`	Medium	Acknowledged
07	`uniV2LimitsStops: approveUnapproved() to arbitrary address`	Medium	Responded
08	`Inconsistent comments & typos`	Low	Fixed
09	`General recommendations`	Low	Acknowledged

#01 `uniV2LimitsStops: stop losses aren’t guaranteed` High

Request and execution model should not be used to Stop Losses orders, because users expect unconditional execution after firing the trigger condition. However, executors may sabotage their epochs or the user’s order may have contradictory parameters (i.g. `amountOutMax < amountOutMin`) and therefore the order may leave in the mempool until cancellation.

Recommendation: Stop Losses risks should be described in the documentation explicitly.

Response from Autonomy team: this is intended and the user can decide whether the stop loss acts as a true stop loss by choosing amountOutMin = 1 wei. This will be displayed to the user in the integrating UI. Although it's worth noting that when DEXes use Autonomy without us doing the integration for them (i.e. most in the long run), we can't control this. It's a UI issue rather than a contract issue.

#02 uniV2LimitsStops: malicious owner can set parameters via setDefaultFeeInfo() High

`setDefaultFeeInfo()` function in [L742](L742) of the `UniV2LimitsStops` contract allows the owner to change the default Uniswap Router and path to fee. Malicious owner can set the siphoning router or break functions with fee by setting the wrong path.

Recommendation: restrict the owner with Timelock or renounce ownership completely.

Response from Autonomy team: owner in UniV2LimitsStops can brick the contract, but can't be used to steal funds because it only impacts the fee and the Registry requires it gets sent the correct fee. Theoretically it should be a lower severity than an upgradable contract since the latter could actually be used to steal funds. It was suggested to add a 12h delay to this fcn which is fair enough, so we'll do that.

Update: the ownership was transferred to the Timelock contract with minimum delay of 48 hours.

#03 Forwarder: owner regulates access High

`setCaller()` function in [L28](L28) of the Forwarder contract allows the owner granting or denying access to the proxy. Malicious owner can deny calls from the Registry and effectively brick it.

Recommendation: deploy the Forwarders during the construction of the Registry or restrict the owner with Timelock.

Response from Autonomy team: the intent with the Owner in the Forwarder is that, incase we redeploy the Registry in the future for the token launch, dapps that aren't upgradable won't need to go through a lengthy upgrade process because they need to change their hardcoded Forwarder address they rely on. We'd only set it once, then change the owner to the zero address. But adding a 12h delay is also reasonable so we'll do that too.

Update: the ownership was transferred to the Timelock contract with minimum delay of 48 hours.

#04 StakeManager: unstake() indexation problem Medium

`unstake()` function in [L191](L191) of the StakeManager contract may be reverted without specific error if it’s called with the last element of the `_stakes[]` not in the first place of the input `idxs[]` parameter. First pop`()` will reduce the array length and the transaction will be reverted.

Update: the issue was fixed.

#05 Registry: execution can be sabotaged Medium

The PoS model can be sabotaged by lazy executors. Possibly ineffective rewards multipliers in the `Miner` contract may push executors to add new requests and execute only them.

Response from Autonomy team: it's no different than what already exists with producing blocks in PoW or PoS in BTC/ETH etc - they can produce a block but not fill it with any txs, but they don't do that because they profit from adding as many txs as possible, which is the same incentive model here. The intent is to add slashing for this behaviour in the future, but it's not implemented yet, I don't think it's necessary for the mentioned reason.

#06 Oracle: subjected to mining attack on random Medium

`getRandNum()` function in L21 of the Oracle contract uses blockhash of the previous block to update the executor for the epoch. Possible mining attack limits the maximum price of the reward token if any would be set. Users and stakers should be warned about possible risks.

#07 uniV2LimitsStops: approveUnapproved() to arbitrary address Medium

`approveUnapproved()` function in L730 of the `UniV2LimitsStops` contract can be used to approve `type(uint256).max` amount of arbitrary token to an arbitrary address.

Response from Autonomy team: UniV2LimitsStops should never actually hold any tokens after each swap is executed and the tokens sent to the user.

#08 Inconsistent comments & typos Low

`Miner` contract contains inconsistent comment in L27-28 and all the mentions of ‘referral’.

`Registry`: typo in L31, 256, 289 and all the mentions of ‘referrer’.

`uniV2LimitsStops`: typo in L20.

Update: the issues were fixed.

#09 General recommendations Low

Forwarder contract lacks the specific events.

Conclusion

3 high severity issues were found. The contracts are highly dependent on the owner's account. Users using the project have to trust the owner and that the owner's account is properly secured.

Audit includes recommendations on the code improving and preventing potential attacks.

Update. The Autonomy team has responded to this report. Individual responses were added after each item in the section. The updated contracts are deployed to the Binance Smart Chain (BSC):

`0x957Fa92cAc1AD4447B6AEc163af57e7E36537c91`	PriceOracle,
`0x3831ff695ddf9f792F2202a9e3121f3880711d87`	Oracle,
`0xde946E11A1F06F58bA0429dAfAaabE6Ec1C7D498`	StakeManager,
`0xcE675B50034a2304B01DC5e53787Ec77BB7965D4`	Forwarder (UserForwarder),
`0xE390b2436df1fE909628fa5eB8f53d041D7B2c93`	Forwarder (GasForwarder),
`0x4F54277e6412504EBa0B259A9E4c69Dc7EE4bB9c`	Forwarder (UserGasForwarder),
`0x18d087F8D22D409D3CD366AF00BD7AeF0BF225Db`	Registry,
`0x151394FBa85A10E7A669f07818aC408b9abb8e09`	UniV2LimitsStops.

Ownership of UniV2LimitsStops, PriceOracle, Oracle and 3 Forwarder contracts were transferred to the Timelock contract with 48 hours of minimum delay:
`0x9Ce05ad236Ad29B9EF6597633201631c097c3f10`.

AUTO token contract hasn't been deployed, but it's address was set to `0x62d68c1dfD5e570D258C40bE204871877F5eE6a0` in deployed Registry and StakeManager contracts.

References

1. EIP-777: Token Standard

Appendix A. Issues' severity classification

We consider an issue critical, if it may cause unlimited losses or break the workflow of the contract and could be easily triggered.

High severity issues may lead to limited losses or break interaction with users or other contracts under very specific conditions.

Medium severity issues do not cause the full loss of functionality but break the contract logic.

Low severity issues are typically nonoptimal code, unused variables, errors in messages. Usually, these issues do not need immediate reactions.

Appendix B. List of examined issue types

Business logic overview

Functionality checks

Following best practices

Access control and authorization

Reentrancy attacks

Front-run attacks

DoS with (unexpected) revert

DoS with block gas limit

Transaction-ordering dependence

ERC/BEP and other standards violation

Unchecked math

Implicit visibility levels

Excessive gas usage

Timestamp dependence

Forcibly sending ether to a contract

Weak sources of randomness

Shadowing state variables

Usage of deprecated code