CERTIK

Security Assessment

Autonomy Network

Dec 2nd, 2021

Table of Contents

Summary

Overview

- Project Summary
- Audit Summary
- Vulnerability Summary
- Audit Scope

Findings

- AUT-01 : Centralization Risk
- AUT-02 : Initial Token Distribution
- FCK-01 : Centralization Risk
- FCK-02 : Unhandled Return Value
- OCK-01 : Centralization Risk
- POC-01 : Potential Price Manipulation
- RCK-01 : Executor Can Arrange the Transaction Order
- RCK-02 : Discussion About the `isAlive` Logic
- SMC-01 : Risk for Weak Randomness
- SMC-02 : Lack of Access Control
- VLS-01 : Potential Reentrancy Attack
- VLS-02 : Third Party Dependencies

Appendix

Disclaimer

About

Summary

This report has been prepared for Autonomy Network to discover issues and vulnerabilities in the source code of the Autonomy Network project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Autonomy Network
Description	An autonomous network for the user on the blockchain system
Platform	BSC
Language	Solidity
Codebase	https://github.com/Autonomy-Network/autonomy-contracts
Commit	21773d88038a145b59cd3dc35f8120000a620631 f48e6c5e67949d82d499b623ec8cc896a8d126fe

Audit Summary

Delivery Date	Dec 02, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	AUTO, Forwarder, Oracle, PriceOracle, Registry, StakeManager, UniV2LimitsStops

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	5	0	0	4	0	1
Medium	1	0	0	0	0	1
Minor	4	0	0	4	0	0
Informational	2	0	0	0	0	2
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
SCK	abstract/Shared.sol	739b710b8ac7312192e1018d80ead340dbf283b146d8bd81423402e7506033ee
AUT	AUTO.sol	646075e2bf5fae1d1782e9e618043e2af6574246eea2f373a3af852601567704
FCK	Forwarder.sol	36d2b80b0ff746467f83ec046834c5b9d6cf7b174f5f5cda3e71e8ac469ddcd3
MCK	Miner.sol	106b98fe02a0c7311344af2c022f83da1bfe51c73fd6cbc91020d6bd302ecf59
OCK	Oracle.sol	a8f8af4b47c98d48e2860a537dd7d583e452f9cea0aeb3a7c4ea0d1b4e641007
POC	PriceOracle.sol	2257ff7eaf4793d24822d01ddf1da433af60418980a59cef812eaf094bf5e413
RCK	Registry.sol	175a14bf68b5ffde2491808715e0c03c2537408a8aa637ad3816d4cbf9526127
SMC	StakeManager.sol	e7ab36ae730f54b429fad92f29b336624dbd01684924fa458e82b69d295445a1
VLS	uniV2LimitsStops.sol	5c5bf368fbad36d2f162dc9ad40dc18f5eb8e9bdc1d734a8be375a7e5ced010c

Overview

Autonomy Network has created a decentralized automation protocol for "if this then that" functionality, so users can make arbitrary transactions in the future under arbitrary conditions. The main component inside the `Autonomy network` is the `registry` contract and the `stakeManager` contract. `registry` contract will let the user create requests and let the executor execute requests. `stakeManger` contract will assign the executor based on the stake result with `AUTO` token.

The contracts have been deployed at the following address:

Contract	Address
PriceOracle	**0x957Fa92cAc1AD4447B6AEc163af57e7E36537c91**
Oracle	**0x3831ff695ddf9f792F2202a9e3121f3880711d87**
StakeManager	**0xde946E11A1F06F58bA0429dAfAaabE6Ec1C7D498**
Forwarder	**0xcE675B50034a2304B01DC5e53787Ec77BB7965D4**
Forwarder	**0xE390b2436df1fE909628fa5eB8f53d041D7B2c93**
Forwarder	**0x4F54277e6412504EBa0B259A9E4c69Dc7EE4bB9c**
Registry	**0x18d087F8D22D409D3CD366AF00BD7AeF0BF225Db**
Miner	**0x5b0d573E340E9903231de468f2032132639D8b01**
Timelock	**0x9Ce05ad236Ad29B9EF6597633201631c097c3f10**
UniV2LimitsStops	**0xB231B2c7BeA4F767951E79dD5f4973Bc1ADdB189**

External Dependencies

The contract serves as the underlying entity to interact with third-party `UniSwap` protocols (token-swapping) and `ERC1820Registry` contract. The scope of the audit treats third-party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

There are a few depending injection contracts or addresses in the current project:

- `_AUTO`, `_oracle` in the `stakeManager` contract
- `_oracle`, `_stakeMan`, `_userForwarder`, `_gasForwarder` and `_userGasForwarder` in the `registry` contract
- `registry`, `userVeriForwarder`, `userFeeVeriForwarder` and `WETH_` in the `UniV2LimitsStops` contract

Note that the listed interface contracts below are not within the current codebase:

- `IForwarder.sol`
- `IRegistry.sol`

- `IOracle.sol`
- `IPriceOracle.sol`
- `IStakeManager.sol`
- `IERC777.sol`
- `IERC777Recipient.sol`
- `IERC1820Registry.sol`
- `IUniswapV2Router02.sol`

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

Privileged Functions

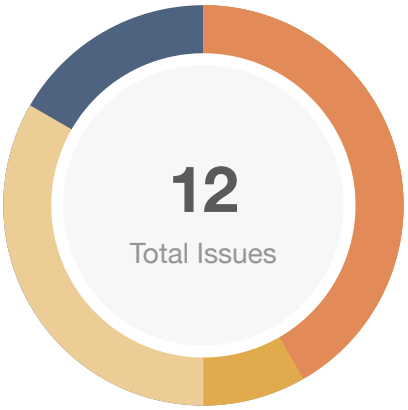
The contract contains the following privileged functions that are restricted by the `onlyOwner` modifier. They are used to modify the contract configurations and address attributes. We grouped these functions below.

The functions below have the `onlyOwner` modifier:

- `AUTO.mint()`
- `Forwarder.setCaller()`
- `Oracle.setPriceOracle()`
- `Oracle.setDefaultPayIsAUTO()`
- `PriceOracle.updateAUTOPerETH()`
- `PriceOracle.updateGasPriceFast()`

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the Timelock contract.

Findings



Critical	0 (0.00%)
Major	5 (41.67%)
Medium	1 (8.33%)
Minor	4 (33.33%)
Informational	2 (16.67%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
AUT-01	Centralization Risk	**Centralization / Privilege**	**Major**	Resolved
AUT-02	Initial Token Distribution	Logical Issue	Minor	Acknowledged
FCK-01	Centralization Risk	**Centralization / Privilege**	**Major**	Acknowledged
FCK-02	Unhandled Return Value	Volatile Code	Minor	Acknowledged
OCK-01	Centralization Risk	**Centralization / Privilege**	**Major**	Acknowledged
POC-01	Potential Price Manipulation	**Centralization / Privilege**	**Major**	Acknowledged
RCK-01	Executor Can Arrange the Transaction Order	Logical Issue	Minor	Acknowledged
RCK-02	Discussion About the `isAlive` Logic	Logical Issue	Informational	Resolved
SMC-01	Risk for Weak Randomness	Logical Issue	Major	Acknowledged
SMC-02	Lack of Access Control	Control Flow	Informational	Resolved
VLS-01	Potential Reentrancy Attack	Volatile Code	Medium	Resolved
VLS-02	Third Party Dependencies	Volatile Code	Minor	Acknowledged

AUT-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	projects/autonomy/contracts/AUTO.sol (cl): 26	Resolved

Description

In the contract `AUTO`, the role `_owner` has the authority over the following function:

- `mint()`: the owner of the contract can mint tokens for an arbitrary address.

Any compromise to the `owner` account may allow the hacker to take advantage of this and cause tokenomic problems to the project.

Recommendation

We advise the client to carefully manage the `_owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

The contract `Registry` created the `AUTO` contract at transaction 0x7dc63a58ed07d6af21481e2ae81fe30b42b28af90933f679b078b300685b15d0 and is the owner of `AUTO`. Therefore, it will not cause any actual problem to the current project.

AUT-02 | Initial Token Distribution

Category	Severity	Location	Status
Logical Issue	Minor	projects/autonomy/contracts/AUTO.sol (cl): 23	Acknowledged

Description

All of the AUTO tokens are sent to a designated address when deploying the contract. This could be a centralization risk as the receiver can distribute AUTO tokens without obtaining the consensus of the community.

Recommendation

We advise the team to be transparent regarding the initial token distribution process.

Alleviation

[Autonomy Network]: It is not necessary for any token creation unless you know the addresses of every single entity in the initial distribution.

[CertiK]: The token usage should be transparent and well documented. Currently, the address **0x3f09e942b0089b8af73ccb9603da8064b6c4b637** holds all the AUTO token 1,000,000,000.

FCK-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/autonomy/contracts/Forwarder.sol (cl): 28	Acknowledged

Description

In the contract `Forwarder`, the role `_owner` has the authority over the following function:

- `setCaller()`: the owner can restrict addresses that can execute `forward`.

Any compromise to the `owner` account may allow the hacker to take advantage of this and register an arbitrary calle in `Forwarder` contract.

Recommendation

We advise the client to carefully manage the `_owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[Autonomy Network]: `Timelock` contract is implemented in the commit f48e6c5e67949d82d499b623ec8cc896a8d126fe.

[CertiK]: The ownership of the following Forwarder contracts have been transferred to the `Timelock` contract **9ce05ad236ad29b9ef6597633201631c097c3f10**. We advise implementing a multi-signature wallet as well to enforc the security of centralization risk.

Here is the `Forwarder` contracts deployment information:

Address	Transaction hash

Address	Transaction hash
0xcE675B50034a2304B01DC5e53787Ec77BB7965D4	0x4b6bb3927406eee9a300b8c81191d27b128e67de62eba22f6c45ff0a0a1f48e
0xE390b2436df1fE909628fa5eB8f53d041D7B2c93	0x1110901d97dee324f98f255b4d9b7a4d5f31c39a7bd0aaa7978126b1396dfba
0x4F54277e6412504EBa0B259A9E4c69Dc7EE4bB9c	0x5d3646aa5af81463bfe51fbd8ca0f56671ae6dfb72d27607d5f54bc3fdeb6bd2

FCK-02 | Unhandled Return Value

Category	Severity	Location	Status
Volatile Code	Minor	projects/autonomy/contracts/Forwarder.sol (cl): 21	Acknowledged

Description

The return value of `address.call{value:}()` is not properly handled. For example,

```
21          (success, returnData) = target.call{value: msg.value}(callData);
```

`address.call{value:}()` is not void-return function. Ignoring the return values of the function might cause some unexpected exceptions, especially if the called functions do not revert automatically on failure.

Recommendation

We recommend checking the return values of the aforementioned functions and handling both success and failure cases based on the business logic.

Example,

```
21          (success, returnData) = target.call{value: msg.value}(callData);
22          require(success, "Forw: forward failed!");
```

Alleviation

[Autonomy Network]: The return value is addressed in the Registry contract.

[CertiK]: The auditors agree this issue will not cause any problem in the `Registry` contract. However, the `Forwarder` contract alone is vulnerable.

OCK-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/autonomy/contracts/Oracle.sol (cl): 37~39, 45~47	Acknowledged

Description

In the contract `Oracle`, the role `_owner` has the authority over the following function:

- `setPriceOracle()`: The owner of the contract can update the price oracle to be an arbitrary one.
- `setDefaultPayIsAUTO()`: The owner of the contract can set `_defaultPayIsAUTO` to be `true`/`false`.

Any compromise to the `_owner` account may allow the hacker to take advantage of this. The attacker can set arbitrary price oracle and set the default payment method for execution.

Recommendation

We advise the client to carefully manage the `_owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[Autonomy Network]: The worst case that an attacker could do here is manipulate the fee charged for paying for execution in AUTO by changing the oracle price of AUTO - so no loss of funds as possible, the attacked can just get cheap execution, that's it. The intention with this is to be able to set it to the AUTO TWAP on Uniswap once there's enough liquidity on Uniswap for the AUTO token. Regardless though, we added Timelock to this and all owner fcns, too.

[CertiK]: The auditors agree the impact might be limited considering the `AUTO` only used for fee charge. However, the team should be aware of the potential risk and ensure the security of the owner's private key.

POC-01 | Potential Price Manipulation

Category	Severity	Location	Status
Centralization / Privilege	**Major**	projects/autonomy/contracts/PriceOracle.sol (cl): 24~26, 32~34	Acknowledged

Description

In the contract `PriceOracle`, the role `_owner` has the authority over the following function:

- `updateAUTOPerETH()`: the owner of the contract can update the exchange rate of AUTO to ETH.
- `updateGasPriceFast()`: the owner of the contract can update gas price to be any value.

Any compromise to the `_owner` account may allow the hacker to take advantage of this and manipulate the price within the project, which could have devastating consequences to the project.

Recommendation

We advise the client to carefully manage the `_owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

In addition, we recommend the client consider the following solutions to apply a reliable oracle according to the project's business model.

1. Use multiple reliable on-chain price oracle sources, such as Chainlink and Uniswap.
2. Use Time-Weighted Average Price (TWAP). The TWAP represents the average price of a token over a specified time frame. If an attacker manipulates the price in one block, it will not affect too much on the average price. Here's an example
3. If the business model allows, restrict the function caller to be a non-contract/EOA address.

Alleviation

[Autonomy Network]: The worst case that an attacker could do here is manipulating the fee charged for paying for execution in AUTO by changing the oracle price of AUTO - so no loss of funds as possible, the attacked can just get cheap execution, that’s it. The intention with this is to be able to set it to the AUTO TWAP on Uniswap once there’s enough liquidity on Uniswap for the AUTO token. Regardless though, we added Timelock to this and all owner fcns, too.

[CertiK]: The auditors agree the impact might be limited considering the `AUTO` only used for fee charge. However, the team should be aware of the potential risk and ensure the security of the owner's private key.

RCK-01 | Executor Can Arrange the Transaction Order

Category	Severity	Location	Status
Logical Issue	Minor	projects/autonomy/contracts/Registry.sol (cl): 332, 361	Acknowledged

Description

Per the design of the project, the `StakeManager` contract would designate an executor to execute `Request`s/transactions in a certain period. The requests are executed by calling `executeHashedReq()` or `executeHashedReqUnveri()`.

```
332     function executeHashedReq(
333         uint id,
334         Request calldata r,
335         uint expectedGas
336     )
```

```
361     function executeHashedReqUnveri(
362         uint id,
363         Request calldata r,
364         bytes memory dataPrefix,
365         bytes memory dataSuffix,
366         uint expectedGas
367     )
```

The execution functions above take `Request` as input, which means the executor can choose the `Request`/transaction he wants to execute. Therefore, the executor can determine the order of transactions. The concern is, if the executor is a malicious node, it might lead to some potential risks (e.g. front-running).

We hope to learn more about the logic and ensure this is would not cause any problem to users and the project.

Recommendation

We advise applying a penalty mechanism to punish the malicious nodes.

Alleviation

[Autonomy Network]: Front running is indeed possible, but it's not more possible when using Autonomy compared to doing a regular e.g. swap on Uniswap as a user - they're equivalent in terms of front running

risk. As soon as the tx hits the mempool, it's frontrunnable by anyone no matter what - people just need a fraction of a second from seeing it in the mempool in order to frontrun a trade. As for the executors changing the order of executed txs, this is indeed possible, but the economic incentive for the executors is to execute as many requests as possible as soon as they're executable because the condition that the requests rely on might not be true in 1 or 2 blocks, so they're incentivised to execute instantly, and are also incentivised, if they're checking how the order affects the executor, to order things in a way that allows the maximum amount of requests to be executed at once so they can collect the max amount of fees. Having nonReentrant in executeHashedReq is intended to prevent this kind of behaviour by forcing executors to make a new tx for every execution, so each execution is equally frontrunnable by other mempool observers want the NFT to constantly be checking for some triggers to take actions or update something about itself).

RCK-02 | Discussion About the `isAlive` Logic

Category	Severity	Location	Status
Logical Issue	Informational	projects/autonomy/contracts/Registry.sol (cl): 136	Resolved

Description

Per the implementation of the contract, the `isAlive` field of a request is initialized when creating a new request. However, there is no specific function that can toggle the state `isAlive`, which means the request would stay "alive"/not "alive" forever. In addition, if the `isAlive` field of a request is set as `true`, the request will be able to execute anytime as long as it is not canceled by the request creator.

We hope to check with the team about the design and ensure it is the intended design.

Alleviation

[Autonomy Network]: This is the intended design - it's basically to make repeated calls of the same function more efficient, rather than having a user/contract constantly make a new request at the end of every executed request. Use cases are recurring payments (subscriptions, salaries), autonomous NFTs (aNFTs - where you want the NFT to constantly be checking for some triggers to take actions or update something about itself)

SMC-01 | Risk for Weak Randomness

Category	Severity	Location	Status
Logical Issue	Major	projects/autonomy/contracts/StakeManager.sol (cl): 133	Acknowledged

Description

The `randNum` is obtained by `_oracle.getRandNum()`, which is generated by `blockhash(seed)`.

```
133             randNum = _oracle.getRandNum(epoch - 1);
134             idxOfExecutor = randNum % stakes.length;
135             exec = stakes[idxOfExecutor];
```

```
21      function getRandNum(uint seed) external override view returns (uint) {
22          return uint(blockhash(seed));
23      }
```

Since the values of `epoch - 1` can be queried, the number `randNum` generated can be predicted.

In this case, an attacker can predict the value of `randNum` and adjust `stakes.length` (by calling `stake` and `unstake`). Therefore, the attacker can manipulate the value of `idxOfExecutor` and further manipulate the `exec`, which is the executor address.

Recommendation

Consider obtaining the random number based on a third-part random service such as chainlink.

Alleviation

[Autonomy Network]: The problem with hardcoding ChainLink in is that they're not on every EVM chain we want to deploy on. I agree that it's a risk, but it's just not worth the cost to an attacker of going through PoW all over again just to get a block hash that makes them the executor.

SMC-02 | Lack of Access Control

Category	Severity	Location	Status
Control Flow	Informational	projects/autonomy/contracts/StakeManager.sol (cl): 50~54	Resolved

Description

The external facing function `setAUTO()` allows anyone to update the significant state `_AUTO` as long as nobody has executed `setAUTO()`. This is potentially risky because an attacker is possible to execute `setAUTO()` before anyone by front-running.

Recommendation

We advise the client to add neccessary access control for `setAUTO()` function and ensure the `AUTO` token is correctly set.

Alleviation

[Autonomy Network]: We were not intended to add an owner to the StakeManager because of the optics. The worst case of someone frontrunning this is that we just have to redeploy, which doesn't affect users.

[CertiK]: The auditors confirmed that the StakeManager contract was correctly deployed and initialized at address **0xde946E11A1F06F58bA0429dAfAaabE6Ec1C7D498**. It will not cause any problem to the current project.

VLS-01 | Potential Reentrancy Attack

Category	Severity	Location	Status
Volatile Code	Medium	projects/autonomy/contracts/uniV2LimitsStops.sol (cl): 133~136	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects.

```
134             registry.transfer(feeAmount);
135             tradeInput -= feeAmount;
```

If the attacker can control the `registry` contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

For example,

```
134             tradeInput -= feeAmount;
135             registry.transfer(feeAmount);
```

Alleviation

[**Autonomy Network**]: Reentrancy is already prevented with the userFeeVerified modifier which forces the caller to be a forwarder, which forces the caller to be the Registry, and all execution functions have nonreentrant modifiers on them.

[**CertiK**]: The auditors confirmed the `register` contract in `UniV2LimitsStops` is **0x18d087f8d22d409d3cd366af00bd7aef0bf225db** and does not contain a malicious fallback function.

VLS-02 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Minor	projects/autonomy/contracts/uniV2LimitsStops.sol (cl)	Acknowledged

Description

The contract is serving as the underlying entity to interact with third-party DEX (Decentralized Exchange) protocols. For example,

```
217     function ethToTokenStopLoss(
218         uint maxGasPrice,
219         IUniswapV2Router02 uni,
220         uint amountOutMin,
221         uint amountOutMax,
222         address[] calldata path,
223         address to,
224         uint deadline
225     ) external payable gasPriceCheck(maxGasPrice) {
226         uint[] memory amounts = uni.swapExactETHForTokens{value: msg.value}
(amountOutMin, path, to, deadline);
227         require(amounts[amounts.length-1] <= amountOutMax, "LimitsStops: price too
high");
228     }
```

The function `ethToTokenStopLoss()` takes `UniswapV2Router02` as the input, which is a third-party DEX protocol.

The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts.

Recommendation

We understand that the business logic of `UniV2LimitsStops` requires interaction with third-part DEX protocol (i.e., Uniswap). We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Autonomy Network]: The input `IUniswapV2Router02 uni` is something that's chosen by the UI of the DEX that's integrating Autonomy. It's not possible for us to know what input 3rd party DEXes are using,

since Autonomy is a B2B tool. I don't think this should be marked as an issue with us, in the same way that ChainLink isn't expected to monitor what dapps use its on-chain data for. Even if we were to monitor it all, we wouldn't be able to take action without introducing more centralization like blacklisting certain addresses.

[CertiK]: For security, it is recommended to sanitize the input and ensure the input will not cause any risk to the project or the user.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.