Pangolin Hedera Contracts Audit Report

September 2, 2022

by **Verilog Solutions**

This report presents our engineering engagement with the Pangolin dev team on their hedera contracts deployed on the Hedera ecosystem.

Project Name	Pangolin Hedera contracts
Repository Link	https://github.com/pangolindex/hedera-contracts/tree/b3a0dc2c51d72687a5c158517cd4e19899aaaa31
Commit Hash	b3a0dc2c51d72687a5c158517cd4e19899aaaa31
Language	Solidity
Chain	Hedera

▼ About Verilog Solutions

Founded by a group of cryptography researchers and smart contract engineers in North America,

Verilog Solutions elevates the security standards for Web3 ecosystems by being a full-stack Web3 security firm covering smart contract security, consensus security, and operational security for Web3 projects.

Verilog Solutions team works closely with major ecosystems and Web3 projects and applies a quality above quantity approach with a continuous security model. Verilog Solutions onboards the best and most innovative projects and provides the best-in-class advisory services on security needs, including on-chain and off-chain components.

▼ Table of Contents

▼ Service Scope

▼ Service Stages

1. Architecture Consultancy Service
    - Protocol Security & Design Discussion Meeting
        - As a part of the audit service, the Verilog Solutions team worked closely with the Pangolin development team to discuss potential vulnerability and smart contract development best practices in a timely fashion. Verilog Solutions team is very appreciative of establishing an efficient and effective communication channel with the Pangolin team, as new findings were exchanged promptly and fixes were deployed quickly, during the preliminary report stage.
2. Smart Contract Auditing Service
    - The Verilog Solutions team analyzed the entire project using a detailed-oriented approach to capture the fundamental logic and suggested improvements to the existing code. Details can be found under **Findings & Improvement Suggestions.**

▼ Methodology

- Code Assessment
    - We evaluate the overall quality of the code and comments as well as the architecture of the repository.
    - We help the project dev team improve the overall quality of the repository by providing suggestions on refactorization to follow the best practice of Web3

software engineering.

- Code Logic Analysis
  - We dive into the data structures and algorithms in the repository and provide suggestions to improve the data structures and algorithms for the lower time and space complexities.
  - We analyze the hierarchy among multiple modules and the relations among the source code files in the repository and provide suggestions to improve the code architecture with better readability, reusability, and extensibility.
- Access Control Analysis
  - We perform a comprehensive assessment of the special roles of the project, including their authorities and privileges.
  - We provide suggestions regarding the best practice of privilege role management according to the standard operating procedures (SOP).
- Off-Chain Components Analysis
  - We analyze the off-chain modules that are interacting with the on-chain functionalities and provide suggestions according to the SOP.
  - We conduct a comprehensive investigation for potential risks and hacks that may happen on the off-chain components and provide suggestions for patches.

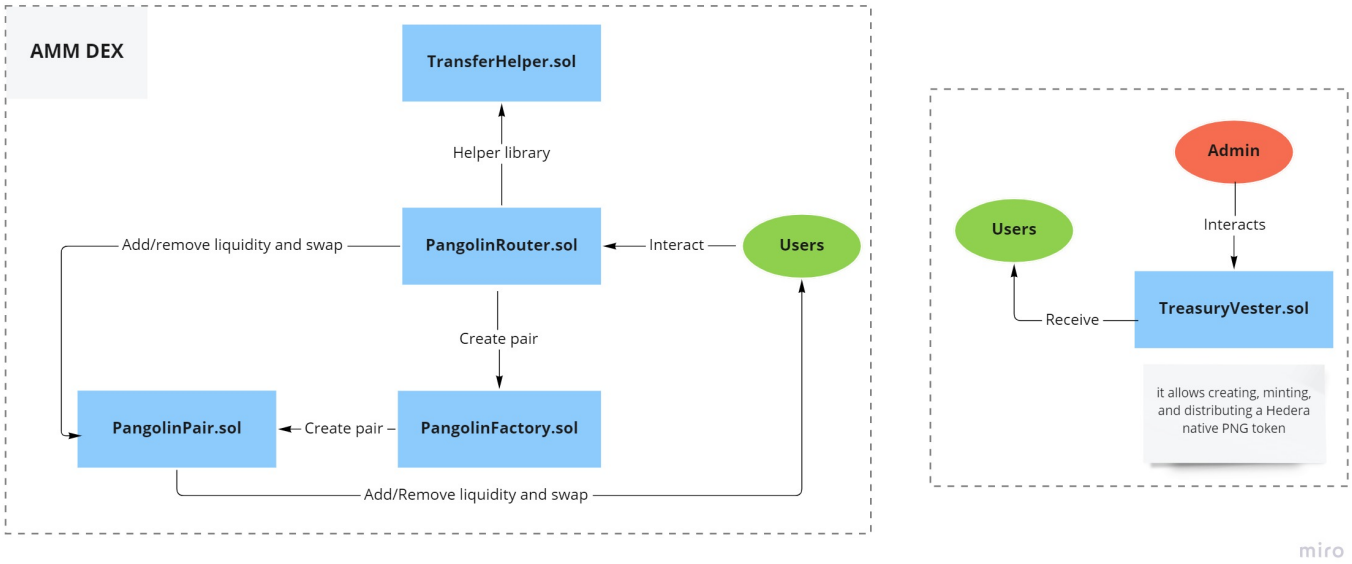
▼ Audit Scope

File
contracts/TreasuryVester.sol
contracts/WHBAR.sol
contracts/pangolin-periphery/PangolinRouter.sol
contracts/pangolin-periphery/interfaces/IPangolinRouter.sol
contracts/pangolin-periphery/interfaces/IBridgeToken.sol
contracts/pangolin-periphery/libraries/PangolinLibrary.sol
contracts/pangolin-periphery/libraries/UniswapV2OracleLibrary.sol
contracts/pangolin-lib/libraries/AddressStringUtil.sol
contracts/pangolin-lib/libraries/Babylonian.sol
contracts/pangolin-lib/libraries/FixedPoint.sol
contracts/pangolin-lib/libraries/FullMath.sol
contracts/pangolin-lib/libraries/PairNamer.sol
contracts/pangolin-core/PangolinERC20.sol
contracts/pangolin-core/PangolinFactory.sol
contracts/pangolin-core/PangolinPair.sol
contracts/pangolin-core/ContractSizeChecker.sol
contracts/pangolin-core/LogicalBurn.sol

File
contracts/pangolin-core/interfaces/IPangolinCallee.sol
contracts/pangolin-core/interfaces/IPangolinERC20.sol
contracts/pangolin-core/interfaces/IPangolinFactory.sol
contracts/pangolin-core/interfaces/IPangolinPair.sol
contracts/pangolin-core/libraries/Math.sol
contracts/pangolin-core/libraries/SafeMath.sol
contracts/pangolin-core/libraries/UQ112x112.sol

▼ Project Summary

Pangolin Hedera contracts contain two parts. The AMM DEX contracts and a treasury vesting contract. The AMM DEX contracts are on Uniswap V2 core contracts with support for Hedera native tokens. Because of some unique features/limitations of the Hedera smart contract service, some important changes and modifications are made to the contract code. The treasury vesting contract distributes Pangolin's Hedera native token PNG based on a 30-month vesting plan.



Pangolin Hedera System Architecture

Pangolin Hedera mainly consists of the following three folders and contracts:

pangolin-core

> contains the logic of interacting with the DEX.

The main contracts of this folder are `PangolinFactory.sol` and `PangolinPair.sol`. They contain the logic to do the following:

- Create a new pair of tokens in the exchange.
- Set the address that will receive the minting fees.
- Get the available reserves of a pair.
- Mint LP tokens.
- Burn LP tokens.
- Swap tokens in the DEX.

pangolin-periphery

> contains the logic for users to interact with the core contracts.

The main contract of this folder is `PangolinRouter.sol` and contains the logic for users to do the following:

- Add/remove liquidity from the specified pair.
- Swap tokens from a pair.
- Query equivalent amount of another asset in a pair.

pangolin-lib

> contains some helper libraries.

The main use of the contracts in this folder is to do the following:

- Use helper math functions.
- Use helper transfer functions.

TreasuryVester.sol

> contains the logic of creating, minting, and distributing a Hedera native `PNG` token.

This contract has the following functionality:

- Assign the admin and define the `PNG` token using the Hedera Token Service on deployment.
- Pause and unpause the contract.
- Set the recipients that will be receiving the tokens.
- Distribute the tokens daily.

▼ Use Case Scenario

Pangolin Hedera is an AMM DEX on the Hedera ecosystem. Similar to Uniswap, Pangolin Hedera allows users to exchange cryptocurrencies and earn rewards by providing liquidity. It also allows users to receive a Hedera native PNG token if they are selected for an airdrop.

▼ Findings & Improvement Suggestions

Severity	Total	Acknowledged	Resolved
High	2	2	2
Medium	1	1	0
Low	1	1	0
Informational	3	3	2
Undetermined	1	1	0

▼ High

1. **Possible attacks due to some Hedera unique features/limitations**

Severity	High
File	pangolin-core/*sol; pangolin-periphery/*.sol;
Commit	b3a0dc2c51d72687a5c158517cd4e19899aaaa31;
Status	**Resolved**;

▼ Description

There are some known unique features/limitations for smart contract services on Hedera EVM. Smart contracts that are designed without considering those features/limitations may not work properly and are subject to attacks like DoS attacks.

The following are some of the known unique features/limitations on Hedera and possible issues that can occur when developing the Pangolin DEX contracts on Hedera.

1. **Smart contract state storage limits**

   On Hedera, there are limits on smart contract state storage. As of writing, the state storage limitation is 10MB. New key-value pairs cannot be created once the contract reaches its limits.
   Also, Hedera will soon adopt a rent mechanism, which requires rent payment for the usage of the state storage of smart contracts. Smart contracts will become “removed” without paying rent.

   Currently, Pangolin DEX contracts on hedera are based on Uniswap V2. LP token creations are permissionless and the pair addresses are stored in the PangolinFactory contract. LP tokens can be created with both Hedera native tokens (tokens created via Hedera token services) and normal ERC20 contract tokens. Attackers can execute DoS attacks on Pangolin DEX by massively creating LP token pairs via the PangolinFactory contract with low gas cost. New pair creations will not be possible once the PangolinFactory contract reaches its state storage

limits. When the rent mechanism is adopted on Hedera, the PangolinFactory contract will be forced to pay the rents for the useless pairs created by the attacker otherwise it will become “removed” on Hedera.

2. **Address zero cannot receive Hedera native tokens**

   An account can not receive Hedera native tokens (tokens created by Hedera token service) without associating with the token. Address zero cannot associate with a token because no one controls this address.

   In AMM smart contracts, minimum liquidity needs to be locked. These LP tokens are normally minted to address zero on Ethereum. But on Hedera, Hedera native tokens cannot be sent to address zero. An alternative way could be locking these tokens into a contract.

3. **Limitations token associations per account before HIP-367 is accepted**

   There are limitations on the number of token IDs that can be associated with an account. The current limit is 1000. HIP-367 proposed that the token IDs association per account should be unlimited. So before the HIP-367 is accepted, the limitation should still be considered.

4. **Hedera Contracts only support** `pragma solidity <=0.8.9`

   As of the current development of hedera-services, Hedera contracts only support `pragma solidity <=0.8.9`.

▼ **Exploit Scenario**

1. Mallory creates a massive amount of pairs through `PangolinFactory.createPair()`;
2. `PangolinFactory` contract reaches its state storage limits;
3. New pair contracts can no longer be created via the factory contracts.

▼ **Recommendations**

1. Optimize contracts. Reduce the usage of mappings and avoid storing unnecessary stuff on chain;
2. Make some critical storage write operations permission only to prevent DoS attack;
3. Use a contract to receive the Hedera native tokens that need to be burnt;
4. Do not associate more than 1000 token IDs to an account before HIP-367 is accepted;
5. Only use solidity with version `pragma solidity <=0.8.9`.

▼ **Results**

Resolved.

As of commit 61b68ef, the following fixes are implemented. Current architecture design makes Pangolin DEX contracts still decentralized without

being prone to Attacks due to the unique features/limitations on Hedera.

1. **Fix on pangolin-core contracts**
    a. Pair token addresses are not stored in the Factory anymore. The pair address is computed when querying the `PangolinFactory.getPair()` function.

    The `extcodesize` is used to check if a pair exists. A middleman contract is added to handle the expected reverts when using the `extcodesize` and `address.size` with an empty address.

    b. A burn contract is created for every 1000 of the pair contract. A single burn contract will associate with 1000 pair tokes. Pairs contracts' minimum liquidity is locked in the associated burn contracts.
    c. Pair contracts can only be created with Hedera native tokens. The pair contract no longer inherits the ERC20 contract. Instead, a Hedera native token created by Hedera token service will be the LP token. In this way, no mappings are needed in the pair contract. (Note: The LP token decimals is 0 now rather than the default 18 anymore)
    d. The initial minimum liquidity is locked in a burn contract rather than address zero.
2. **Fix on pangolin-periphery contracts**
    a. The modified WHBAR contract is used in the router contract. WHBAR contract is no longer an ERC20 token. Instead, a hedera native token created by Hedera token service is associated with the WHABR contract to get rid of mappings. When a user deposits Hbar, the WHBAR token is minted and transferred to the user. When a user withdraws Hbar, the WHBar token is burnt and the HBar is transferred back to the user. Based on the change of the WHABR contract, the router contract is changed accordingly. The `PangolinRouter.removeLiquidityWithPermit()` and `PangolinRouter.removeLiquidityAVAXWithPermitSupportingFeeOnTransferTokens()` are removed from the router contract.

2. **The type of `INITIAL_SUPPLY` should be `uint64`**

Severity	High
File	TreasuryVester.sol#L24;
Commit	b3a0dc2c51d72687a5c158517cd4e19899aaaa31；
Status	**Resolved;**

▼ **Description**

The type of `INITIAL_SUPPLY` should be `uint64`. The value assigned to

`INITIAL_SUPPLY` will cause the overflow error with type `uint32`. The value assigned to `INITIAL_SUPPLY` is `11_500_000 * uint32(10)**8`, which is far bigger than the value type `uint32` can represent. According to the hedera token service document, `uint64` is used for token initial supply. Thus, the type of `INITIAL_SUPPLY` should be `uint64`.

▼ Exploit Scenario

This will cause a revert when `INITIAL_SUPPLY` is used in contract functions.

▼ Recommendations

Change the type of `INITIAL_SUPPLY` to `uint64`.

▼ Results

Resolved in commit 8c6eee, type of `INITIAL_SUPPLY` changed to uint64.

▼ Medium

1. **Lack of two-step process for critical operations**

Severity	Medium
File	contracts/pangolin-core/PangolinFactory.sol#L46;
Commit	b3a0dc2c51d72687a5c158517cd4e19899aaaa31;
Status	**Acknowledged;**

▼ Description

The `feeToSetter` variable can set the new `feeToSetter` and set `feeTo` address, an address that receives all the minting fees. `feeToSetter` can be changed immediately by calling the `setFeeToSetter` function. With the best practice of smart contract programming, the update of a privileged account should be separated into two steps. This is to avoid the situation where a wrong account is mistakenly set but no person is able to recover it.

```
function setFeeToSetter(address _feeToSetter) external override {
    require(msg.sender == feeToSetter, 'Pangolin: FORBIDDEN');
    feeToSetter = _feeToSetter;
}
```

▼ Exploit Scenario

The `feeToSetter` is accidentally set to the wrong address and cannot be set back.

▼ Recommendations

Use a two-step process for change of `feeToSetter`.

Please define another public variable storing pending `feeToSetter`, when we want to update the account of the fee setter, we first save the new address in the new variable. An extra confirmation function is then used to confirm the fee

setter.

▼ **Results**

Acknowledged.

Pangolin decided to keep this part unchanged.

▼ Low

1. **Incorrect event emission for critical operations**

Severity	Low
File	TreasuryVester.sol#L171;
Commit	b3a0dc2c51d72687a5c158517cd4e19899aaaa31;
Status	**Unresolved;**

▼ **Description**

The `TreasuryVester.TokenVested()` event emits the ideal token vested amount (`vestingAmount`), instead of the actual token vested amount (`actualVestingAmount`). The `actualVestingAmount` is the sum of all recipient's individual vesting amounts, which is calculated with recipients' allocation in the `vestingAmount`. There might be some tiny differences between `vestingAmount` and `actualVestingAmount` because of the precision issues when doing multiplication and division in solidity. Only the `vestingAmount` is emitted through events.

▼ **Exploit Scenario**

N/A.

▼ **Recommendations**

Change `vestingAmount` in `TokenVested()` event to `actualVestingAmount`.

▼ **Results**

Unresolved.

▼ Informational

1. **Inconsistent file name**

Severity	Informational
File	pangolin-periphery/libraries/UniswapV2OracleLibrary.sol;
Commit	b3a0dc2c51d72687a5c158517cd4e19899aaaa31;
Status	**Acknowledged;**

▼ **Description**

The file name of `UniswapV2OracleLibrary.sol` should be `PangolinOracleLibrary.sol`.

▼ **Exploit Scenario**

N/A.

▼ **Recommendations**

Change file name of `UniswapV2OracleLibrary.sol` to `PangolinOracleLibrary.sol`.

▼ **Results**

Acknowledged.

2. **Inconsistent function/variable name**

Severity	Informational
File	*Global;
Commit	b3a0dc2c51d72687a5c158517cd4e19899aaaa31;
Status	**Acknowledged;**

▼ **Description**

A lot of contracts function/variable names still contain or use `avax` rather than `hbar`.

▼ **Exploit Scenario**

N/A.

▼ **Recommendations**

Change `avax` to `hbar`.

▼ **Results**

Acknowledged.

3. **Inconsistent comments and code**

Severity	Informational
File	TreasuryVester.sol#L23;
Commit	b3a0dc2c51d72687a5c158517cd4e19899aaaa31;
Status	**Resolved** in commit 53d0377059745defd1fe3f66e29969197880410e;

▼ **Description**

The number mentioned in the comments does not match the number defined in the code.

```
uint32 private constant MAX_SUPPLY = 230_000_000 * uint32(10)**DECIMALS; // two-hundred-an
uint32 private constant INITIAL_SUPPLY = 11_500_000 * uint32(10)**DECIMALS; // twelve mill
```

▼ **Exploit Scenario**

N/A.

▼ **Recommendations**

Please double-check and verify the number.

▼ **Results**

Resolved in commit 53d0377059745defd1fe3f66e29969197880410e.

Comments are updated.

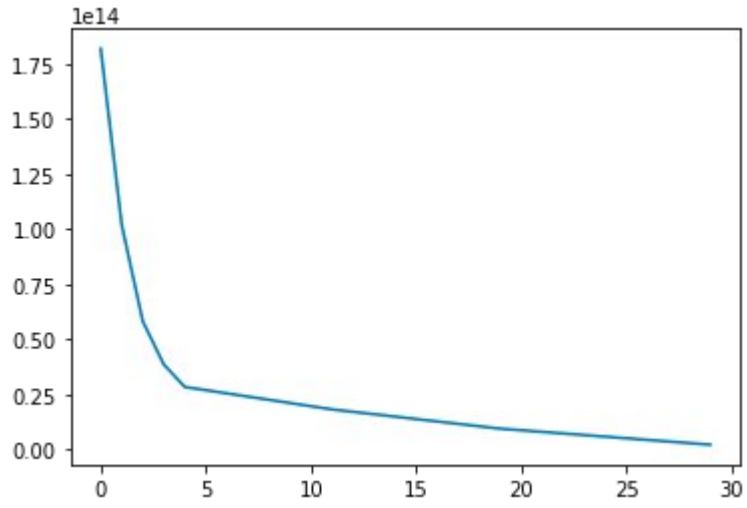
▼ Undetermined

1. **Token vesting rules and issues**

Severity	**Undetermined**
File	TreasuryVester.sol#L41;
Commit	b3a0dc2c51d72687a5c158517cd4e19899aaaa31;
Status	**Acknowledged;**

▼ **Description**

The specification of the airdrop is unknown. Vesting schedule will be delayed if not distributed daily by calling function `TreasuryVester.distribute()`.
The ideal token distribution is that 218.5m tokens are distributed continuously for 30 months. To satisfy this, `TreasuryVester.distribute()` has to be called daily. If `TreasuryVester.distribute()` is not called daily, users cannot collect the past uncollected rewards together. Thus, token distribution will be delayed and cannot be finished in 30 months.
The graph shows the ideal token distribution schedule from month 1 to month 30.



PNG token vesting schedule

▼ **Exploit Scenario**

1. Token vesting is initially scheduled for 30 months;
2. `distribute()` is not called for 15 days in total;
3. The vesting schedule is delayed for 15 days.

▼ Recommendations

To be able to distribute tokens within 30 months, The contract should allow users to collect missed vestings if the `TreasuryVester.distribute()` is not called daily.

▼ Results

Acknowledged.

Some delays in the token vesting schedule are accepted and tolerated by the Pangolin team.

▼ Access Control Analysis

The privileged roles and access control for the contracts are the following:

▼ PangolinFactory.sol

- `feeToSetter`: Allows to set the `feeToSetter` and `feeTo` address.
- `feeTo`: It's the address that receives the minting fee.

▼ TreasuryVester.sol

The deployer of the contract can assign the `DEFAULT_ADMIN_ROLE` address. This address has control over the following functionality:

- Pause the contract.
- Unpause the contract.
- Transfer the `PNG` initial supply to the input address.
- Set recipients that will be receiving the token.

▼ Appendix I: Severity Categories

Severity	Description
High	Issues that are highly exploitable security vulnerabilities. It may cause direct loss of funds / permanent freezing of funds. All high severity issues should be resolved.
Medium	Issues that are only exploitable under some conditions or with some privileged access to the system. Users' yields/rewards/information is at risk. All medium severity issues should be resolved unless there is a clear reason not to.
Low	Issues that are low risk. Not fixing those issues will not result in the failure of the system. A fix on low severity issues is recommended but subject to the clients' decisions.
Informational	Issues that pose no risk to the system and are related to the security best practices. Not fixing those issues will not result in the failure of the system. A fix on informational issues or adoption of those security best practices-related suggestions is recommended but subject to clients' decision.

▼ Appendix II: Status Categories

Status	Description
Unresolved	The issue is not acknowledged and not resolved.
Partially Resolved	The issue has been partially resolved.
Acknowledged	The Finding / Suggestion is acknowledged but not fixed / not implemented.
Resolved	The issue has been sufficiently resolved.

▼ Disclaimer

Verilog Solutions receives compensation from one or more clients for performing the smart contract and auditing analysis contained in these reports. The report created is solely for Clients and published with their consent. As such, the scope of our audit is limited to a review of code, and only the code we note as being within the scope of our audit detailed in this report. It is important to note that the Solidity code itself presents unique and unquantifiable risks since the Solidity language itself remains under current development and is subject to unknown risks and flaws. Our sole goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies. Thus, Verilog Solutions in no way claims any guarantee of security or functionality of the technology we agree to analyze.

In addition, Verilog Solutions reports do not provide any indication of the technologies proprietors, business, business model, or legal compliance. As such, reports do not provide investment advice and should not be used to make decisions about investment or involvement with any particular project. Verilog Solutions has the right to distribute the Report through other means, including via Verilog Solutions publications and other distributions. Verilog Solutions makes the reports available to parties other than the Clients (*i.e.*, “third parties”) – on its website in hopes that it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.