// HALBORN

Pangolin & AVA Labs

Zap Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 5th, 2021 - August 12th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/12/2021	Ataberk Yavuzer
0.5	Document Edits	08/13/2021	Gokberk Gulgun
0.9	Document Edits	08/16/2021	Ferran Celades
1.0	Final Version	08/16/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Ferran Celades	Halborn	Ferran.Celades@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

AVA Labs engaged Halborn to conduct a security assessment on Pangolin's Zap Smart Contract beginning on August 5th, 2021 and ending August 12th, 2021.
The security assessment was scoped to the smart contracts PangolinBridgeMigrationRouter.sol and BridgeToken.sol. An audit of the security risk and implications regarding the changes introduced by the development team at Pangolin prior to its production release shortly following the assessments deadline.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week timeframe for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security engineer is blockchain and smart contracts security expert, with experience in advanced penetration testing, smart contract hacking, and have a deep knowledge in multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that smart contract functions are intended.
- Identify potential security issues with the smart contracts.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart Contract manual code read and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions(solgraph)
- Manual Assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Testnet deployment (hardhat, avash, geth and Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

Both PangolinBridgeMigrationRouter.sol and BridgeToken.sol contracts in the following repository are the test scope.

Repository URL: https://github.com/pangolindex/exchange-contracts
Commit ID: 78134a9df30014b27dc2eb12574ea3050749f7dd

OUT-OF-SCOPE:
Other smart contracts in the repository, external libraries and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	3	4	7

LIKELIHOOD

IMPACT

(HAL-01)				
(HAL-04)				
(HAL-07)	(HAL-05) (HAL-06)	(HAL-02) (HAL-03)		
(HAL-08)				
(HAL-09) (HAL-10) (HAL-11) (HAL-12) (HAL-13) (HAL-14)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL01) - DOS - LP TOKENS NOT MINTED (Out Of Scope)	Medium	-
(HAL02) - INTEGER OVERFLOW	Medium	-
(HAL03) - MISSING ROLE-BASED ACCESS CONTROL	Medium	-
(HAL04) - MISSING ONLY ROUTER MODIFIER (Out Of Scope)	Low	-
(HAL05) - CONFUSION ON THE ADMIN ROLE	Low	-
(HAL06) - LACK OF ADDRESS VALIDATION	Low	-
(HAL07) - DIVISION BY ZERO	Low	-
(HAL08) - MISSING EVENTS EMITTING	Informational	-
(HAL09) - MISSING RE-ENTRANCY PROTECTION	Informational	-
(HAL10) - IMPROPER CHECK EFFECT INTERACTION PATTERN USAGE	Informational	-
(HAL11) - USE OF APPROVE FUNCTION	Informational	-
(HAL12) - USE OF BLOCK.TIMESTAMP	Informational	-
(HAL13) - FLOATING PRAGMA	Informational	-
(HAL14) - IMPROPER IMPLEMENTATION OF CONTRACT ADMIN	Informational	-
STATIC ANALYSIS	-	-
AUTOMATED SECURITY SCAN RESULTS	-	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) DOS - LP TOKENS NOT MINTED (Out Of Scope) - MEDIUM

Description:

Calling sync on the PangolinPair contract before any mint of the pool, by front-running the transaction, it will force liquidity inside the function to be 0, causing a DOS and the INSUFFICIENT_LIQUIDITY_MINTED reverts an error. This is due to reserve0 and reserve1 are equal to the total token balance, causing amount0 and amount1 on mint to be 0. This issue would prevent anyone to mint their LP tokens. Once the attacker wants he can stop front-running the sync function before any mint and get the reserve amount as his LP tokens.

Code Location:

Listing 1: contracts/pangolin-core/PangolinPair.sol (Lines 198)

```
197 // force reserves to match balances
198 function sync() external lock {
199     _update(IERC20(token0).balanceOf(address(this)), IERC20(token1
        ).balanceOf(address(this)), reserve0, reserve1);
200 }
```

Listing 2: contracts/pangolin-core/PangolinPair.sol (Lines 82,83)

```
92 // update reserves and, on the first call per block, price
    accumulators
93 function _update(uint balance0, uint balance1, uint112 _reserve0,
    uint112 _reserve1) private {
94     require(balance0 <= uint112(-1) && balance1 <= uint112(-1), '
        Pangolin: OVERFLOW');
95     uint32 blockTimestamp = uint32(block.timestamp % 2**32);
96     uint32 timeElapsed = blockTimestamp - blockTimestampLast; //
        overflow is desired
97     if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
98         // * never overflows, and + overflow is desired
99         price0CumulativeLast += uint(UQ112x112.encode(_reserve1).
            uqdiv(_reserve0)) * timeElapsed;
```

```
100         price1CumulativeLast += uint(UQ112x112.encode(_reserve0).
                uqdiv(_reserve1)) * timeElapsed;
101     }
102     reserve0 = uint112(balance0);
103     reserve1 = uint112(balance1);
104     blockTimestampLast = blockTimestamp;
105     emit Sync(reserve0, reserve1);
106 }
```

Listing 3: contracts/pangolin-core/PangolinPair.sol (Lines 114,115)

```
110 function mint(address to) external lock returns (uint liquidity) {
111     (uint112 _reserve0, uint112 _reserve1,) = getReserves(); //
            gas savings
112     uint balance0 = IERC20(token0).balanceOf(address(this));
113     uint balance1 = IERC20(token1).balanceOf(address(this));
114     uint amount0 = balance0.sub(_reserve0);
115     uint amount1 = balance1.sub(_reserve1);
116
117     bool feeOn = _mintFee(_reserve0, _reserve1);
118     uint _totalSupply = totalSupply; // gas savings, must be
            defined here since totalSupply can update in _mintFee
119     if (_totalSupply == 0) {
120         liquidity = Math.sqrt(amount0.mul(amount1)).sub(
                MINIMUM_LIQUIDITY);
121        _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock
               the first MINIMUM_LIQUIDITY tokens
122     } else {
123         liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0
                , amount1.mul(_totalSupply) / _reserve1);
124     }
125     require(liquidity > 0, 'Pangolin:
            INSUFFICIENT_LIQUIDITY_MINTED');
126     _mint(to, liquidity);
127
128     _update(balance0, balance1, _reserve0, _reserve1);
129     if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0
            and reserve1 are up-to-date
130     emit Mint(msg.sender, amount0, amount1);
131 }
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendations:

It is recommended to avoid direct calling of any PangolinPair function. This can be prevented by using a modifier that detects the caller and allow only calls from the PangolinRouter address. The issue is also stated as “MISSING ONLY ROUTER MODIFIER”.

3.2 (HAL-02) INTEGER OVERFLOW - MEDIUM

Description:

An overflow happens when an arithmetic operation reaches the maximum size of a type. For instance in the BridgeToken.sol contract on addSwapToken method the supply is added which may end up overflowing the integer. In computer programming, an integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits -- either larger than the maximum or lower than the minimum re-presentable value.

Code Location:

Listing 4

```
function addSwapToken(address contractAddress, uint256
    supplyIncrement)
    public
{
    require(bridgeRoles.has(msg.sender), "Unauthorized.");
    require(isContract(contractAddress), "Address is not contract
        .");

    // If the swap token is not already supported, add it with the
        total supply of supplyIncrement.
    // Otherwise, increment the current supply.
    if (swapTokens[contractAddress].tokenContract == address(0)) {
        swapTokens[contractAddress] = SwapToken({
            tokenContract: contractAddress,
            supply: supplyIncrement
        });
    } else {
        swapTokens[contractAddress].supply =
            swapTokens[contractAddress].supply +
            supplyIncrement;
    }
    emit AddSwapToken(contractAddress, supplyIncrement);
}
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system or use pragma version bigger than 0.8.0 that adds arithmetic checks automatically.

3.3 (HAL-03) MISSING ROLE-BASED ACCESS CONTROL - MEDIUM

Description:

In smart contracts, implementing a correct Access Control policy is an essential step to maintain security and decentralization for permissions on a token. All the features of the smart contract , such as mint/burn tokens and pause contracts are given by Access Control. For instance, Ownership is the most common form of Access Control. In other words, the owner of a contract (the account that deployed it by default) can do some administrative tasks on it. Nevertheless, other authorization levels are required to follow the principle of least privilege, also known as least authority. Briefly, any process, user or program only can access to the necessary resources or information. Otherwise, the ownership role is useful in a simple system, but more complex projects require the use of more roles by using Role-based access control.

Code Location:

Listing 5: PangolinBridgeMigrationRouter.sol

```
259  function _migrateLiquidity(
260          address liquidityPairFrom,
261          address liquidityPairTo,
262          address to,
263          uint amount
264      ) internal {
265          _arePairsCompatible(liquidityPairFrom, liquidityPairTo);
266          address tokenToMigrate = IPangolinPair(liquidityPairFrom).
                 token0();
267
268          ...
269
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

It is recommended to use role-based access controls on important functions such as `_migrateLiquidity`. In the other way, any user beside contract/-contract admin itself will be able to use these high privileged functions. Also, the `onlyAdmin` modifier could be appended to the `_migrateLiquidity` function to solve the issue.

3.4 (HAL-04) MISSING ONLY ROUTER MODIFIER (Out Of Scope) - LOW

Description:

The PangolinPair contract does not implement any modifier to detect if the functions declared on it were called from the PangolinRouter contract. This allows external functions such as mint and burn to be called without going thought the expected addLiquidity and removeLiquidity. This could cause price miscalculation and minted LP tokens to be owned by a different user than the transferring one. Furthermore, the code states the following comment “this low-level function should be called from a contract which performs important safety checks”. Imagine the following scenario:

- A user manually transfers t1 amount of token1 and t2 amount of token2 to the PangolinPair. This amount is added to the pool balance but no LP tokens are accredited since they are still pending on the pool balance.
- The same user then calls the mint function to receive the corresponding LP tokens of the transferred t1 and t2 amounts on the PangolinPair pool.
- Before the transaction is mined on the chain a malicious user calls the mint function with a controlled address.
- The generated LP token are minted to the attacker instead of the user.

Risk Level:

Likelihood - 1
Impact - 4

Recommendations:

A modifier should be added on the PangolinPair critical functions (the ones using the lock modifier) to make sure those functions are only called

from the exposed router methods. This would prevent unexpected execution flows and logical errors to happen by preventing the execution of the pool methods independently.

Listing 6

```
modifier onlyIsRouter() {
    require(
        msg.sender == swappRouter,
        "ONLY ROUTER ALLOWED"
    );
    _;
}
```

3.5 (HAL-05) CONFUSION ON THE ADMIN ROLE - LOW

Description:

There is a confusion on the access control policy between the PangolinBridgeMigrationRouter.sol and BridgeToken.sol contracts. While PangolinBridgeMigrationRouter.sol contract has multiple admin role, BridgeToken.sol has single admin role. On the other hand, PangolinBridgeMigrationRouter.sol contract can not denote msg.sender and can not add zero address into admin role. However, in the BridgeToken.sol contract, bridge role can be transfer to zero address. After the lockout (migrating admin to zero address), **addSwapToken**, **removeSwapToken**, **addSupportedChainId**, **mint** functions are not accessible.

Code Location:

PangolinBridgeMigrationRouter

Listing 7: PangolinBridgeMigrationRouter.sol

```
37 function addAdmin(address account) external onlyAdmin {
38         require(account != address(0), "
               PangolinBridgeMigrationRouter: Address 0 not allowed");
39         adminRole.add(account);
40     }
```

Listing 8: PangolinBridgeMigrationRouter.sol

```
47     function removeAdmin(address account) external onlyAdmin {
48         require(msg.sender != account, "
               PangolinBridgeMigrationRouter: You can't demote
               yourself");
49         adminRole.remove(account);
50     }
```

BridgeToken.sol

Listing 9: BridgeToken.sol

```
111 function migrateBridgeRole(address newBridgeRoleAddress) public {
112         require(bridgeRoles.has(msg.sender), "Unauthorized.");
113         bridgeRoles.remove(msg.sender);
114         bridgeRoles.add(newBridgeRoleAddress);
115         emit MigrateBridgeRole(newBridgeRoleAddress);
116     }
117
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendations:

It is recommended to review admin roles between the contracts. The admin counts need to be evaluated according to the contact structure. Also, the address validation should be added into BridgeToken.sol contract. This issue is handled on the (HAL03)- LACK OF ADDRESS VALIDATION.

3.6 (HAL-06) LACK OF ADDRESS VALIDATION - LOW

Description:

The BridgeToken.sol and PangolinBridgeMigrationRouter.sol contracts have multiple input fields on their both public and private functions. Some of these inputs are required as address variable. During the test, it seen all of these inputs are protected against using the address(0) as target address. However, providing the contract address itself to these inputs is unintended situation.

The addSwapToken and migrateBridgeRole functions in the BridgeToken.sol contract does not control current contract address. Also, addAdmin , addMigrator, _arePairsCompatible, _migrateToken, calculateChargeBack functions in the PangolinBridgeMigrationRouter.solcontract do not control the current contract address too.

Code Location:

Listing 10: PangolinBridgeMigrationRouter.sol

```
37 function addAdmin(address account) external onlyAdmin {
38         require(account != address(0), "
               PangolinBridgeMigrationRouter: Address 0 not allowed");
39         adminRole.add(account);
40     }
```

Listing 11: PangolinBridgeMigrationRouter.sol

```
66 function addMigrator(address tokenAddress, address migratorAddress
       ) external onlyAdmin {
67         require(tokenAddress != address(0), "
               PangolinBridgeMigrationRouter: tokenAddress 0 not
               supported");
68         require(migratorAddress != address(0), "
               PangolinBridgeMigrationRouter: migratorAddress 0 not
               supported");
```

```
69          uint256 amount = IBridgeToken(migratorAddress).swapSupply(
                tokenAddress);
70
71          ...
```

Listing 12: PangolinBridgeMigrationRouter.sol

```
145 function _arePairsCompatible(address pairA, address pairB)
        internal view {
146         require(pairA != address(0), "
                PangolinBridgeMigrationRouter: liquidityPairFrom
                address 0");
147         require(pairA != address(0), "
                PangolinBridgeMigrationRouter: liquidityPairTo address
                0");
148         require(pairA != pairB, "PangolinBridgeMigrationRouter:
                Cant convert to the same liquidity pairs");
149         require(
150
151         ...
```

Listing 13: PangolinBridgeMigrationRouter.sol

```
164 function _migrateToken(
165         address tokenAddress,
166         uint amount
167     ) internal {
168         require(tokenAddress != address(0), "
                PangolinBridgeMigrationRouter: tokenAddress 0 not
                supported");
169         IBridgeToken(bridgeMigrator[tokenAddress]).swap(
                tokenAddress, amount);
170
171         ...
```

Listing 14: PangolinBridgeMigrationRouter.sol

```
335 function calculateChargeBack(
336         address liquidityPairFrom,
337         address liquidityPairTo,
338         uint amount
339     ) external view returns (uint amount0, uint amount1) {
```

```
340         require(liquidityPairFrom != address(0), "
                PangolinBridgeMigrationRouter: liquidityPairFrom
                address 0 not supported");
341         require(liquidityPairTo != address(0), "
                PangolinBridgeMigrationRouter: liquidityPairTo address
                0 not supported");
342         (uint amountIn0, uint amountIn1) =
                _calculateRescueLiquidity(liquidityPairFrom, amount);
343
344         ...
345
```

Listing 15: BridgeToken.sol

```
111 function migrateBridgeRole(address newBridgeRoleAddress) public {
112         require(bridgeRoles.has(msg.sender), "Unauthorized.");
113         bridgeRoles.remove(msg.sender);
114         bridgeRoles.add(newBridgeRoleAddress);
115         emit MigrateBridgeRole(newBridgeRoleAddress);
116     }
117
```

Listing 16: BridgeToken.sol

```
123  function addSwapToken(address contractAddress, uint256
        supplyIncrement)
124         public
125     {
126         require(bridgeRoles.has(msg.sender), "Unauthorized.");
127         require(isContract(contractAddress), "Address is not
                contract.");
128
129         // If the swap token is not already supported, add it with
                the total supply of supplyIncrement.
130         // Otherwise, increment the current supply.
131         if (swapTokens[contractAddress].tokenContract == address
                (0)) {
132             swapTokens[contractAddress] = SwapToken({
133                 tokenContract: contractAddress,
134                 supply: supplyIncrement
135             });
136
137             ...
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendations:

It is recommended to implement additional address check to detect is current contract getting used as a target address.

Listing 17: Recommendation

```
require(tokenAddress != address(this), "
    PangolinBridgeMigrationRouter: current contract address is not
    supported");
```

3.7 (HAL-07) DIVISION BY ZERO - LOW

Description:

Calling the burn function with totalSupply as 0 will cause the function to throw a division by zero error.

Code Location:

Listing 18: contracts/pangolin-core/PangolinPain.sol (Lines 143,144,145)

```
133 // this low-level function should be called from a contract which
      performs important safety checks
134 function burn(address to) external lock returns (uint amount0,
      uint amount1) {
135     (uint112 _reserve0, uint112 _reserve1,) = getReserves(); //
          gas savings
136     address _token0 = token0;                                //
          gas savings
137     address _token1 = token1;                                //
          gas savings
138     uint balance0 = IERC20(_token0).balanceOf(address(this));
139     uint balance1 = IERC20(_token1).balanceOf(address(this));
140     uint liquidity = balanceOf[address(this)];
141
142     bool feeOn = _mintFee(_reserve0, _reserve1);
143     uint _totalSupply = totalSupply; // gas savings, must be
          defined here since totalSupply can update in _mintFee
144     amount0 = liquidity.mul(balance0) / _totalSupply; // using
          balances ensures pro-rata distribution
145     amount1 = liquidity.mul(balance1) / _totalSupply; // using
          balances ensures pro-rata distribution
146     require(amount0 > 0 && amount1 > 0, 'Pangolin:
          INSUFFICIENT_LIQUIDITY_BURNED');
147     _burn(address(this), liquidity);
148     _safeTransfer(_token0, to, amount0);
149     _safeTransfer(_token1, to, amount1);
150     balance0 = IERC20(_token0).balanceOf(address(this));
151     balance1 = IERC20(_token1).balanceOf(address(this));
```

```
152
153     _update(balance0, balance1, _reserve0, _reserve1);
154     if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0
            and reserve1 are up-to-date
155     emit Burn(msg.sender, amount0, amount1, to);
156 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

Make sure to validate all operands used during a math operation and inform the user of unappropriated state by reverting the transaction with a custom message.

3.8 (HAL-08) MISSING EVENTS EMITTING - INFORMATIONAL

Description:

It has been observed that critical functionality is missing emitting event for some functions on the PangolinBridgeMigrationRouter.sol contract. These functions should emit events after completing the transactions.

Code Location:

Listing 19: Missing Events

```
addAdmin(address account)
removeAdmin(address account)
addMigrator(address tokenAddress, address migratorAddress)
_addLiquidity(address pairToken, address token0, address token1,
    uint amountIn0, uint amountIn1, address to)
_rescueLiquidity(address liquidityPair, uint amount)
_migrateToken(address tokenAddress, uint amount)
migrateToken(address token, address to, uint amount, uint deadline
    )
migrateLiquidityWithPermit(address liquidityPairFrom, address
    liquidityPairTo, address to, uint amount, uint deadline, uint8
    v, bytes32 r, bytes32 s)
migrateLiquidity(address liquidityPairFrom, address
    liquidityPairTo, address to, uint amount, uint deadline)
_migrateLiquidity(address liquidityPairFrom, address
    liquidityPairTo, address to, uint amount)
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

Consider emitting an event when calling related functions on the list above.

Listing 20: Events

```
1 event addAdmin(address account)
2 event removeAdmin(address account)
3 event addMigrator(address tokenAddress, address migratorAddress)
```

Listing 21: Example Emit (Lines 4)

```
1 function addAdmin(address account) external onlyAdmin {
2         require(account != address(0), "
              PangolinBridgeMigrationRouter: Address 0 not allowed");
3         adminRole.add(account);
4         emit addAdmin(address account)
5     }
```

3.9 (HAL-09) MISSING RE-ENTRANCY PROTECTION - INFORMATIONAL

Description:

To protect against cross-function reentrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdraw function with a recursive call. OpenZeppelin has it's own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against Reentrancy attacks.

Code Location:

Listing 22: PangolinBridgeMigrationRouter.sol (Lines 108)

```
101 function _addLiquidity(
102         address pairToken,
103         address token0,
104         address token1,
105         uint amountIn0,
106         uint amountIn1,
107         address to
108     ) private returns (uint amount0, uint amount1, uint
        liquidityAmount) {
109
110     ...
```

Listing 23: PangolinBridgeMigrationRouter.sol (Lines 134)

```
131 function _rescueLiquidity(
132         address liquidityPair,
133         uint amount
134     ) internal returns (uint amountTokenA, uint amountTokenB) {
```

Listing 24: PangolinBridgeMigrationRouter.sol (Lines 189)

```
184 function migrateToken(
185         address token,
186         address to,
187         uint amount,
188         uint deadline
189     ) external ensure(deadline) {
190
191     ...
```

Listing 25: PangolinBridgeMigrationRouter.sol (Lines 264)

```
259 function _migrateLiquidity(
260         address liquidityPairFrom,
261         address liquidityPairTo,
262         address to,
263         uint amount
264     ) internal {
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

In PangolinBridgeMigrationRouter.sol contract, functions like migrateToken(), migrateLiquidity() and _rescueLiquidity() are missing nonReentrant guard. Use the nonReentrant modifier to avoid introducing future vulnerabilities.

3.10 (HAL-10) IMPROPER CHECK EFFECT INTERACTION PATTERN USAGE - INFORMATIONAL

Description:

In the Smart Contracts, The check effect interaction pattern is used to reduce the attack surface for malicious contracts trying to hijack control flow after an external call. In the PangolinBridgeMigrationRouter.sol, amount0 and amount1 is updated after an external call.

Code Location:

Listing 26: PangolinBridgeMigrationRouter.sol (Lines 118,119)

```
101    function _addLiquidity(
102         address pairToken,
103         address token0,
104         address token1,
105         uint amountIn0,
106         uint amountIn1,
107         address to
108     ) private returns (uint amount0, uint amount1, uint
           liquidityAmount) {
109         (uint112 reserve0, uint112 reserve1,) = IPangolinPair(
               pairToken).getReserves();
110         uint quote0 = amountIn0;
111         uint quote1 = PangolinLibrary.quote(amountIn0, reserve0,
               reserve1);
112         if (quote1 > amountIn1) {
113             quote1 = amountIn1;
114             quote0 = PangolinLibrary.quote(amountIn1, reserve1,
                   reserve0);
115         }
116         TransferHelper.safeTransfer(token0, pairToken, quote0);
117         TransferHelper.safeTransfer(token1, pairToken, quote1);
118         amount0 = amountIn0 - quote0;
119         amount1 = amountIn1 - quote1;
120         liquidityAmount = IPangolinPair(pairToken).mint(to);
```

```
121     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

In the `_addLiquidity` function, `amount0` and `amount1` should be updated before an external call.

3.11 (HAL-11) USE OF APPROVE FUNCTION - INFORMATIONAL

Description:

Due to the implementation of the approve() function in PangolinBridgeMigrationRouter.sol, it's possible for a user to over spend their allowance in the certain conditions. Furthermore,burnFrom on Burnable ERC20 tokens is using the same internal allowance data type which allows both, burning and minting.

Code Location:

Listing 27: PangolinBridgeMigrationRouter.sol

```
37     function _allowToken(address tokenAddress, address
           spenderAddress) internal {
38         IPangolinERC20(tokenAddress).approve(spenderAddress, type(
               uint).max);
39     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

Consider to use that instead of having a direct setter for allowances, decreaseAllowance and increaseAllowance functions should be exposed which decreases and increases allowances for a recipient respectively.

3.12 (HAL-12) USE OF BLOCK.TIMESTAMP - INFORMATIONAL

Description:

The **PangolinBridgeMigrationRouter** contract is using block.timestamp. The global variable block.timestamp does not necessarily hold the current time, and may not be accurate. Miners can influence the value of block.timestamp to perform Maximal Extractable Value (MEV) attacks. There is no guarantee that the value is correct, only that it is higher than the previous block's timestamp.

Code Location:

Listing 28: PangolinBridgeMigrationRouter.sol (Lines 22)

```
21 modifier ensure(uint deadline) {
22         require(deadline >= block.timestamp, '
               PangolinBridgeMigrationRouter: EXPIRED');
23         _;
24     }
```

Recommendations:

Use block.number instead of block.timestamp or now to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years, days and months rather than seconds.

3.13 (HAL-13) FLOATING PRAGMA - INFORMATIONAL

Description:

The PangolinBridgeMigrationRouter.sol contract uses the floating pragma ^0.7.6 and the BridgeToken.sol contract uses ^0.7.0 floating pragma as version. These contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the **pragma** helps to ensure that contracts do not accidentally get deployed using another pragma, for example, either an outdated pragma version that might introduce bugs that affect the contract system negatively or a recently released pragma version which has not been extensively tested.

Reference: ConsenSys Diligence - Lock pragmas

Code Location:

Listing 29: PangolinBridgeMigrationRouter.sol (Lines 1)

```
1 pragma solidity ^0.7.6;
2
3 import "../pangolin-core/interfaces/IPangolinERC20.sol";
4 import "../pangolin-lib/libraries/TransferHelper.sol";
5
6 ...
```

Listing 30: BridgeToken.sol (Lines 2)

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.7.0;
3
4 import "./library/openzeppelin/ERC20Burnable.sol";
5 import "../libraries/Roles.sol";
6
7 ...
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

Lock the pragma version whenever possible and avoid using a floating pragma in the final deployment. The pragma can be locked in the code by removing the caret (^) and by specifying the exact version in the Truffle configuration file truffle-config.js or hardhat.config.js if using the HardHat framework.

3.14 (HAL-14) IMPROPER IMPLEMENTATION OF CONTRACT ADMIN - INFORMATIONAL

Description:

Roles are used on the developed contracts to provide ease of use or to separate the tasks on the contract from each other. It is very important that these roles should be sharply separated from each other during the deployment phase of the contract and assigned to the right accounts.

In some cases, unexpected conditions may occur as a result of incorrect programming of these roles. During the tests, it was seen that the admin role on the PangolinBridgeMigrationRouter.sol contract was given to the person who directly deployed it on the constructor. This situation does not provide flexibility in defining who the admin user is.

Code Location:

Listing 31: PangolinBridgeMigrationRouter.sol (Lines 17)

```
16 constructor() public {
17         adminRole.add(msg.sender);
18     }
19
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

According to OpenZeppelin's AccessControl.sol contract, the part seen in the code example below is more reliable for implementing an administrator role:

Listing 32: AccessControl.sol

```
constructor (address root) public {
    _setupRole(DEFAULT_ADMIN_ROLE, root);
  }
```

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

```
INFO:Detectors:
PangolinBridgeMigrationRouter._migrateToken(address,uint256) (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#164-174) uses a dangerous strict equality:
        - require(bool,string)(IBridgeToken(bridgeMigrator[tokenAddress]).balanceOf(address(this)) == amount,PangolinBridgeMigrationRouter: Didn't yield the correct amount) (contracts/pangolin-periphery/Pa
ngolinBridgeMigrationRouter.sol#170-173)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
PangolinLibrary.getAmountsOut(address,uint256,address[]).i (contracts/pangolin-periphery/libraries/PangolinLibrary.sol#67) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
PangolinBridgeMigrationRouter._allowToken(address,address) (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#83-85) ignores return value by IPangolinERC20(tokenAddress).approve(spenderAddress
,type()(uint256).max) (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#84)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Reentrancy in PangolinBridgeMigrationRouter.addMigrator(address,address) (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#66-76):
        External calls:
        - _allowToken(tokenAddress,migratorAddress) (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#74)
          - IPangolinERC20(tokenAddress).approve(spenderAddress,type()(uint256).max) (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#84)
        State variables written after the call(s):
        - bridgeMigrator[tokenAddress] = migratorAddress (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#75)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['>=0.5.0', '>=0.6.0', '>=0.6.6<0.8.0', '^0.7.0', '^0.7.6']
        - >=0.5.0 (contracts/pangolin-core/interfaces/IPangolinERC20.sol#1)
        - >=0.5.0 (contracts/pangolin-core/interfaces/IPangolinFactory.sol#1)
        - >=0.5.0 (contracts/pangolin-core/interfaces/IPangolinPair.sol#1)
        - >=0.6.0 (contracts/pangolin-lib/libraries/TransferHelper.sol#3)
        - ^0.7.0 (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#1)
        - >=0.5.0 (contracts/pangolin-periphery/interfaces/IBridgeToken.sol#1)
        - >=0.5.0 (contracts/pangolin-periphery/libraries/PangolinLibrary.sol#1)
        - ^0.7.0 (contracts/pangolin-periphery/libraries/Roles.sol#2)
        - >=0.6.6<0.8.0 (contracts/pangolin-periphery/libraries/SafeMath.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
PangolinLibrary.getAmountIn(uint256,uint256,uint256) (contracts/pangolin-periphery/libraries/PangolinLibrary.sol#54-60) is never used and should be removed
PangolinLibrary.getAmountOut(uint256,uint256,uint256) (contracts/pangolin-periphery/libraries/PangolinLibrary.sol#44-51) is never used and should be removed
PangolinLibrary.getAmountsIn(address,uint256,address[]) (contracts/pangolin-periphery/libraries/PangolinLibrary.sol#74-82) is never used and should be removed
PangolinLibrary.getAmountsOut(address,uint256,address[]) (contracts/pangolin-periphery/libraries/PangolinLibrary.sol#63-71) is never used and should be removed
PangolinLibrary.getReserves(address,address,address) (contracts/pangolin-periphery/libraries/PangolinLibrary.sol#30-34) is never used and should be removed
PangolinLibrary.pairFor(address,address,address) (contracts/pangolin-periphery/libraries/PangolinLibrary.sol#19-27) is never used and should be removed
PangolinLibrary.sortTokens(address,address) (contracts/pangolin-periphery/libraries/PangolinLibrary.sol#12-16) is never used and should be removed
SafeMath.add(uint256,uint256) (contracts/pangolin-periphery/libraries/SafeMath.sol#6-8) is never used and should be removed
SafeMath.sub(uint256,uint256) (contracts/pangolin-periphery/libraries/SafeMath.sol#10-12) is never used and should be removed
TransferHelper.safeApprove(address,address,uint256) (contracts/pangolin-lib/libraries/TransferHelper.sol#7-15) is never used and should be removed
TransferHelper.safeTransferAVAX(address,uint256) (contracts/pangolin-lib/libraries/TransferHelper.sol#38-41) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.5.0 (contracts/pangolin-core/interfaces/IPangolinERC20.sol#1) allows old versions
Pragma version>=0.5.0 (contracts/pangolin-core/interfaces/IPangolinFactory.sol#1) allows old versions
Pragma version>=0.5.0 (contracts/pangolin-core/interfaces/IPangolinPair.sol#1) allows old versions
Pragma version>=0.6.0 (contracts/pangolin-lib/libraries/TransferHelper.sol#3) allows old versions
Pragma version>=0.5.0 (contracts/pangolin-periphery/interfaces/IBridgeToken.sol#1) allows old versions
Pragma version>=0.5.0 (contracts/pangolin-periphery/libraries/PangolinLibrary.sol#1) allows old versions
Pragma version^0.7.0 (contracts/pangolin-periphery/libraries/Roles.sol#2) allows old versions
Pragma version>=0.6.6<0.8.0 (contracts/pangolin-periphery/libraries/SafeMath.sol#1) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
INFO:Detectors:
Low level call in TransferHelper.safeApprove(address,address,uint256) (contracts/pangolin-lib/libraries/TransferHelper.sol#7-15):
        - (success,data) = token.call(abi.encodeWithSelector(0x095ea7b3,to,value)) (contracts/pangolin-lib/libraries/TransferHelper.sol#13)
Low level call in TransferHelper.safeTransfer(address,address,uint256) (contracts/pangolin-lib/libraries/TransferHelper.sol#17-25):
        - (success,data) = token.call(abi.encodeWithSelector(0xa9059cbb,to,value)) (contracts/pangolin-lib/libraries/TransferHelper.sol#23)
Low level call in TransferHelper.safeTransferFrom(address,address,address,uint256) (contracts/pangolin-lib/libraries/TransferHelper.sol#27-36):
        - (success,data) = token.call(abi.encodeWithSelector(0x23b872dd,from,to,value)) (contracts/pangolin-lib/libraries/TransferHelper.sol#34)
Low level call in TransferHelper.safeTransferAVAX(address,uint256) (contracts/pangolin-lib/libraries/TransferHelper.sol#38-41):
        - (success) = to.call{value: value}(new bytes(0)) (contracts/pangolin-lib/libraries/TransferHelper.sol#39)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IPangolinERC20.DOMAIN_SEPARATOR() (contracts/pangolin-core/interfaces/IPangolinERC20.sol#18) is not in mixedCase
Function IPangolinERC20.PERMIT_TYPEHASH() (contracts/pangolin-core/interfaces/IPangolinERC20.sol#19) is not in mixedCase
Function IPangolinPair.DOMAIN_SEPARATOR() (contracts/pangolin-core/interfaces/IPangolinPair.sol#18) is not in mixedCase
Function IPangolinPair.PERMIT_TYPEHASH() (contracts/pangolin-core/interfaces/IPangolinPair.sol#19) is not in mixedCase
Function IPangolinPair.MINIMUM_LIQUIDITY() (contracts/pangolin-core/interfaces/IPangolinPair.sol#36) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable PangolinBridgeMigrationRouter._rescueLiquidity(address,uint256).amountTokenA (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#134) is too similar to PangolinBridgeMigrationRouter._r
escueLiquidity(address,uint256).amountTokenB (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#134)
Variable PangolinBridgeMigrationRouter._migrateLiquidity(address,address,address,uint256).amountTokenA (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#284) is too similar to PangolinBridgeM
igrationRouter._migrateLiquidity(address,address,address,uint256).amountTokenB (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#284)
Variable PangolinBridgeMigrationRouter._migrateLiquidity(address,address,address,uint256).amountTokenA (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#284) is too similar to PangolinBridgeM
igrationRouter._rescueLiquidity(address,uint256).amountTokenB (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#134)
Variable PangolinBridgeMigrationRouter._rescueLiquidity(address,uint256).amountTokenA (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#134) is too similar to PangolinBridgeMigrationRouter._m
igrateLiquidity(address,address,address,uint256).amountTokenB (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#284)
Variable PangolinBridgeMigrationRouter._migrateLiquidity(address,address,address,uint256).changeAmount0 (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#298) is too similar to PangolinBridge
MigrationRouter._migrateLiquidity(address,address,address,uint256).changeAmount1 (contracts/pangolin-periphery/PangolinBridgeMigrationRouter.sol#298)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
```

```
INFO:Detectors:
Reentrancy in BridgeToken.swap(address,uint256) (contracts/pangolin-periphery/test/BridgeToken.sol#185-209):
        External calls:
        - swapToken.burnFrom(msg.sender,amount) (contracts/pangolin-periphery/test/BridgeToken.sol#203)
        State variables written after the call(s):
        - _mint(msg.sender,amount) (contracts/pangolin-periphery/test/BridgeToken.sol#206)
                - _balances[account] = _balances[account].add(amount) (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#234)
        - _mint(msg.sender,amount) (contracts/pangolin-periphery/test/BridgeToken.sol#206)
                - _totalSupply = _totalSupply.add(amount) (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#233)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in BridgeToken.swap(address,uint256) (contracts/pangolin-periphery/test/BridgeToken.sol#185-209):
        External calls:
        - swapToken.burnFrom(msg.sender,amount) (contracts/pangolin-periphery/test/BridgeToken.sol#203)
        Event emitted after the call(s):
        - Swap(token,amount) (contracts/pangolin-periphery/test/BridgeToken.sol#208)
        - Transfer(address(0),account,amount) (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#235)
                - _mint(msg.sender,amount) (contracts/pangolin-periphery/test/BridgeToken.sol#206)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
BridgeToken.addSupportedChainId(uint256) (contracts/pangolin-periphery/test/BridgeToken.sol#75-92) uses assembly
        - INLINE ASM (contracts/pangolin-periphery/test/BridgeToken.sol#80-82)
BridgeToken.isContract(address) (contracts/pangolin-periphery/test/BridgeToken.sol#216-222) uses assembly
        - INLINE ASM (contracts/pangolin-periphery/test/BridgeToken.sol#218-220)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
BridgeToken.addSupportedChainId(uint256) (contracts/pangolin-periphery/test/BridgeToken.sol#75-92) compares to a boolean constant:
        -chainIds[chainId] == true (contracts/pangolin-periphery/test/BridgeToken.sol#86)
BridgeToken.unwrap(uint256,uint256) (contracts/pangolin-periphery/test/BridgeToken.sol#100-105) compares to a boolean constant:
        -require(bool,string)(chainIds[chainId] == true,Chain ID not supported.) (contracts/pangolin-periphery/test/BridgeToken.sol#102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['>=0.6.0<0.8.0', '^0.7.0']
        - ^0.7.0 (contracts/pangolin-periphery/libraries/Roles.sol#2)
        - ^0.7.0 (contracts/pangolin-periphery/test/BridgeToken.sol#2)
        - >=0.6.0<0.8.0 (contracts/pangolin-periphery/test/library/openzeppelin/Context.sol#3)
        - >=0.6.0<0.8.0 (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#3)
        - >=0.6.0<0.8.0 (contracts/pangolin-periphery/test/library/openzeppelin/ERC20Burnable.sol#3)
        - >=0.6.0<0.8.0 (contracts/pangolin-periphery/test/library/openzeppelin/IERC20.sol#3)
        - >=0.6.0<0.8.0 (contracts/pangolin-periphery/test/library/openzeppelin/OpenZeppelinSafeMath.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

```
INFO:Detectors:
Pragma version^0.7.0 (contracts/pangolin-periphery/libraries/Roles.sol#2) allows old versions
Pragma version^0.7.0 (contracts/pangolin-periphery/test/BridgeToken.sol#2) allows old versions
Pragma version>=0.6.0<0.8.0 (contracts/pangolin-periphery/test/library/openzeppelin/Context.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/pangolin-periphery/test/library/openzeppelin/ERC20Burnable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/pangolin-periphery/test/library/openzeppelin/IERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/pangolin-periphery/test/library/openzeppelin/OpenZeppelinSafeMath.sol#3) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Redundant expression "this (contracts/pangolin-periphery/test/library/openzeppelin/Context.sol#21)" inContext (contracts/pangolin-periphery/test/library/openzeppelin/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
decimals() should be declared external:
        - BridgeToken.decimals() (contracts/pangolin-periphery/test/BridgeToken.sol#43-45)
        - ERC20.decimals() (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#89-91)
mint(address,uint256,address,uint256,bytes32) should be declared external:
        - BridgeToken.mint(address,uint256,address,uint256,bytes32) (contracts/pangolin-periphery/test/BridgeToken.sol#56-69)
addSupportedChainId(uint256) should be declared external:
        - BridgeToken.addSupportedChainId(uint256) (contracts/pangolin-periphery/test/BridgeToken.sol#75-92)
unwrap(uint256,uint256) should be declared external:
        - BridgeToken.unwrap(uint256,uint256) (contracts/pangolin-periphery/test/BridgeToken.sol#100-105)
migrateBridgeRole(address) should be declared external:
        - BridgeToken.migrateBridgeRole(address) (contracts/pangolin-periphery/test/BridgeToken.sol#111-116)
addSwapToken(address,uint256) should be declared external:
        - BridgeToken.addSwapToken(address,uint256) (contracts/pangolin-periphery/test/BridgeToken.sol#123-142)
removeSwapToken(address,uint256) should be declared external:
        - BridgeToken.removeSwapToken(address,uint256) (contracts/pangolin-periphery/test/BridgeToken.sol#149-169)
swapSupply(address) should be declared external:
        - BridgeToken.swapSupply(address) (contracts/pangolin-periphery/test/BridgeToken.sol#176-178)
swap(address,uint256) should be declared external:
        - BridgeToken.swap(address,uint256) (contracts/pangolin-periphery/test/BridgeToken.sol#185-209)
name() should be declared external:
        - ERC20.name() (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#64-66)
symbol() should be declared external:
        - ERC20.symbol() (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#72-74)
totalSupply() should be declared external:
        - ERC20.totalSupply() (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#96-98)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#103-105)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#115-118)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#134-137)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#152-156)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#170-173)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (contracts/pangolin-periphery/test/library/openzeppelin/ERC20.sol#189-192)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (contracts/pangolin-periphery/test/library/openzeppelin/ERC20Burnable.sol#22-24)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (contracts/pangolin-periphery/test/library/openzeppelin/ERC20Burnable.sol#37-42)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers

to locate any vulnerabilities. Only security-related findings are shown below.

Results:

PangolinBridgeMigrationRouter.sol

Report for pangolin-periphery/PangolinBridgeMigrationRouter.sol
https://dashboard.mythx.io/#/console/analyses/c8b7e455-35a1-49c5-90de-2fe1fdd4f9dd

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
9	(SWC-123) Requirement Violation	Low	Requirement violation.
22	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.
75	(SWC-107) Reentrancy	Medium	Write to persistent state following external call
84	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.
84	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
320	(SWC-123) Requirement Violation	Low	Requirement violation.

BridgeToken.sol

Report for test/BridgeToken.sol
https://dashboard.mythx.io/#/console/analyses/b5a4af92-7989-4968-b837-9cf247bd733d

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
20	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
101	(SWC-115) Authorization through tx.origin	Low	Use of "tx.origin" as a part of authorization control.
101	(SWC-115) Authorization through tx.origin	Low	Use of tx.origin as a part of authorization control.

All relevant findings were founded in the manual code review.

THANK YOU FOR CHOOSING

// HALBORN