Pangolin - MiniChefV2Zapper

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 4th, 2022 - February 9th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/04/2022	Roberto Reigada
0.2	Document Updates	02/09/2022	Roberto Reigada
0.3	Draft Review	02/09/2022	Gabi Urrutia
1.0	Remediation Plan	02/10/2022	Roberto Reigada
1.1	Remediation Plan Review	02/10/2022	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
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Pangolin engaged Halborn to conduct a security audit on their MiniChefV2Zapper smart contract beginning on February 4th, 2022 and ending on February 9th, 2022. The security assessment was scoped to the smart contract provided in the GitHub repository pangolindex/exchange-contracts.

1.2 AUDIT SUMMARY

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

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were addressed by Pangolin team.

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 this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code 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 review 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
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, 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. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were 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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the following smart contract:

- MiniChefV2Zapper.sol

Commit ID: aaca68c25e5226bfbb0d011c5a5a13468e4a3f47

Fixed Commit ID: f2caccfa254cf9cc3a16be797a6624ce86f48f4a

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	2

LIKELIHOOD

IMPACT

🟨	🟨	🟥	🟥	🟪
🟩	🟨	🟨	🟥	🟥
🟩	🟩	🟨	🟨	🟥
⬜	🟩	🟩 (HAL-02)	🟨	🟨 (HAL-01)
⬜ (HAL-03) (HAL-04)	⬜	🟩	🟩	🟨

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - WRONG PERMIT CALL IN ZAPINVIAPERMIT FUNCTION	Medium	SOLVED - 02/10/2022
HAL02 - MISSING ZERO ADDRESS CHECKS	Low	RISK ACCEPTED
HAL03 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 02/10/2022
HAL04 - USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS	Informational	SOLVED - 02/10/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) WRONG PERMIT CALL IN ZAPINVIAPERMIT FUNCTION - MEDIUM

Description:

The function zapInViaPermit() is used to call the zapIn function in just one transaction, without having to have sent previously an initial transaction approving the tokenIn transfer:

Listing 1: MiniChefV2Zapper.sol (Lines 85)

```
77      function zapInViaPermit(
78          address pairAddress,
79          address tokenIn,
80          uint256 tokenInAmount,
81          uint256 tokenAmountOutMin,
82          uint256 deadline,
83          uint8 v, bytes32 r, bytes32 s
84      ) external {
85          IPangolinPair(pairAddress).permit(msg.sender, address(this
                ), tokenInAmount, deadline, v, r, s);
86          zapIn(pairAddress, tokenIn, tokenInAmount,
                tokenAmountOutMin);
87      }
```

The permit, though, is done on the pairAddress token instead of the tokenIn token, which is the token address which should be actually approved, as can be seen in the code below:

Listing 2: MiniChefV2Zapper.sol (Lines 72)

```
68      function zapIn(address pairAddress, address tokenIn, uint256
            tokenInAmount, uint256 tokenAmountOutMin) public {
69          require(tokenInAmount >= minimumAmount, 'Insignificant
                input amount');
70          require(pairAddress != address(0), 'Invalid pair address')
                ;
71
72          TransferHelper.safeTransferFrom(tokenIn, msg.sender,
                address(this), tokenInAmount);
73
74          _swapAndFarm(pairAddress, tokenIn, tokenAmountOutMin, 0,
                false);
75      }
```

With the current implementation, zapInViaPermit() will not work as the permit is wrongly executed.

Risk Level:

Likelihood - 5
Impact - 2

Recommendation:

It is recommended to use tokenIn instead of pairAddress in the permit() call:

Listing 3: MiniChefV2Zapper.sol (Lines 85)

```
77      function zapInViaPermit(
78          address pairAddress,
79          address tokenIn,
80          uint256 tokenInAmount,
81          uint256 tokenAmountOutMin,
82          uint256 deadline,
83          uint8 v, bytes32 r, bytes32 s
84      ) external {
85          IPangolinPair(tokenIn).permit(msg.sender, address(this),
                tokenInAmount, deadline, v, r, s);
86          zapIn(pairAddress, tokenIn, tokenInAmount,
                tokenAmountOutMin);
87      }
```

Remediation Plan:

SOLVED: The Pangolin team replaced pairAddress with tokenIn in the permit call.

3.2 (HAL-02) MISSING ZERO ADDRESS CHECKS - LOW

Description:

The constructor of the MiniChefV2Zapper contract is missing address validation. Every address should be validated and checked that is different from zero. This is also considered a best practice.

Code location:

Listing 4: MiniChefV2Zapper.sol (Lines 40-42)

```
35 constructor(address _router, address _miniChefV2, address _WAVAX)
       {
36     // Safety checks to ensure WAVAX token address
37     IWAVAX(_WAVAX).deposit{value: 0}();
38     IWAVAX(_WAVAX).withdraw(0);
39
40     router = IPangolinRouter(_router);
41     miniChefV2 = IMiniChefV2(_miniChefV2);
42     WAVAX = _WAVAX;
43 }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to validate that every address input is different from zero.

Remediation Plan:

RISK ACCEPTED: The Pangolin team accepted this risk.

3.3 (HAL-03) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In the MiniChefV2Zapper contract there is a function marked as public but it is never directly called within the same contract or in any of their descendants:

MiniChefV2Zapper.sol
- estimateSwap() (MiniChefV2Zapper.sol#296-312)

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If the function is not intended to be called internally or by their descendants, it is better to mark it as external to reduce gas costs.

Remediation Plan:

SOLVED: The Pangolin team declared the estimateSwap() function as external
.

3.4 (HAL-04) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS - INFORMATIONAL

Description:

In all the loops, the variable i is incremented using i++. It is known that, in loops, using ++i costs less gas per iteration than i++.

Code Location:

MiniChefV2Zapper.sol

- Line 277: `for (uint256 i; i < tokens.length; i++){`

Proof of Concept:

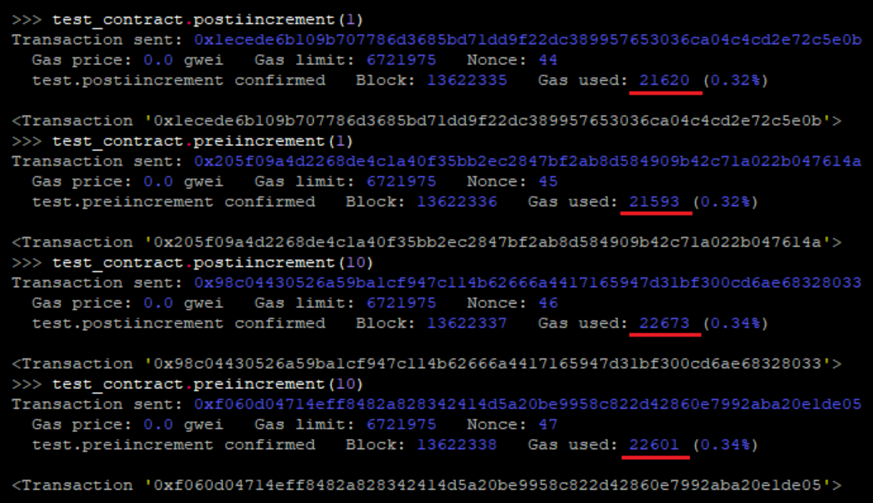
For example, based in the following test contract:

Listing 5: Test.sol

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.9;

contract test {
    function postiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; i++) {
        }
    }
    function preiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; ++i) {
        }
    }
}
```

We can see the difference in the gas costs:



```
>>> test_contract.postiincrement(1)
Transaction sent: 0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 44
  test.postiincrement confirmed   Block: 13622335   Gas used: 21620 (0.32%)

<Transaction '0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b'>
>>> test_contract.preiincrement(1)
Transaction sent: 0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 45
  test.preiincrement confirmed   Block: 13622336   Gas used: 21593 (0.32%)

<Transaction '0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a'>
>>> test_contract.postiincrement(10)
Transaction sent: 0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 46
  test.postiincrement confirmed   Block: 13622337   Gas used: 22673 (0.34%)

<Transaction '0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033'>
>>> test_contract.preiincrement(10)
Transaction sent: 0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 47
  test.preiincrement confirmed   Block: 13622338   Gas used: 22601 (0.34%)

<Transaction '0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05'>
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use ++i instead of i++ to increment the value of an uint variable inside a loop. This is not applicable outside of loops.

Remediation Plan:

SOLVED: The Pangolin team uses now ++i to increment the i variable inside loops, saving some gas.

AUTOMATED TESTING

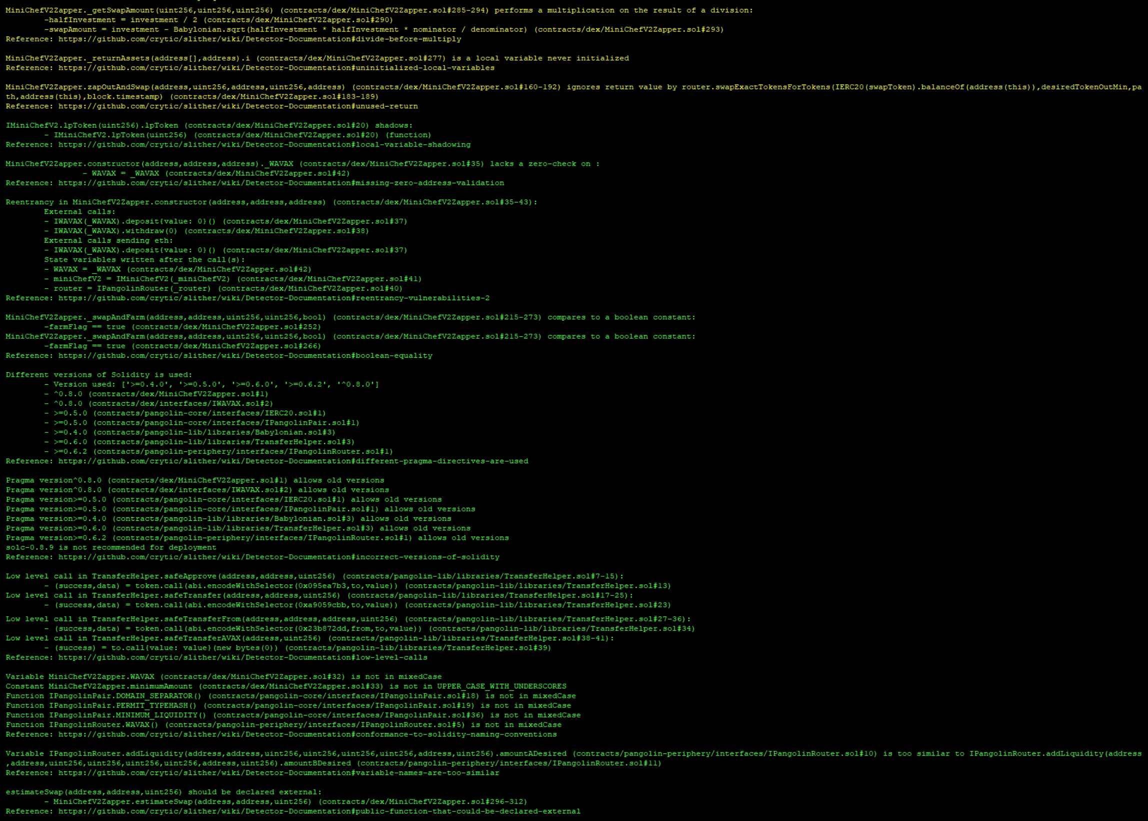
4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. 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, Slither was run on the all-scoped contracts. 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.

Slither results:

MiniChefV2Zapper.sol



```
MiniChefV2Zapper._getSwapAmount(uint256,uint256,uint256) (contracts/dex/MiniChefV2Zapper.sol#285-294) performs a multiplication on the result of a division:
        -halfInvestment = investment / 2 (contracts/dex/MiniChefV2Zapper.sol#290)
        -swapAmount = investment - Babylonian.sqrt(halfInvestment * halfInvestment * nominator / denominator) (contracts/dex/MiniChefV2Zapper.sol#293)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

MiniChefV2Zapper._returnAssets(address[],address).i (contracts/dex/MiniChefV2Zapper.sol#277) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

MiniChefV2Zapper.zapOutAndSwap(address,uint256,address,uint256,address) (contracts/dex/MiniChefV2Zapper.sol#160-192) ignores return value by router.swapExactTokensForTokens(IERC20(swapToken).balanceOf(address(this)),desiredTokenOutMin,pa
th,address(this),block.timestamp) (contracts/dex/MiniChefV2Zapper.sol#183-189)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

IMiniChefV2.lpToken(uint256).lpToken (contracts/dex/MiniChefV2Zapper.sol#20) shadows:
        - IMiniChefV2.lpToken(uint256) (contracts/dex/MiniChefV2Zapper.sol#20) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

MiniChefV2Zapper.constructor(address,address,address)._WAVAX (contracts/dex/MiniChefV2Zapper.sol#35) lacks a zero-check on :
                - WAVAX = _WAVAX (contracts/dex/MiniChefV2Zapper.sol#42)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in MiniChefV2Zapper.constructor(address,address,address) (contracts/dex/MiniChefV2Zapper.sol#35-43):
        External calls:
        - IWAVAX(_WAVAX).deposit{value: 0}() (contracts/dex/MiniChefV2Zapper.sol#37)
        - IWAVAX(_WAVAX).withdraw(0) (contracts/dex/MiniChefV2Zapper.sol#38)
        External calls sending eth:
        - IWAVAX(_WAVAX).deposit{value: 0}() (contracts/dex/MiniChefV2Zapper.sol#37)
        State variables written after the call(s):
        - WAVAX = _WAVAX (contracts/dex/MiniChefV2Zapper.sol#42)
        - miniChefV2 = IMiniChefV2(_miniChefV2) (contracts/dex/MiniChefV2Zapper.sol#41)
        - router = IPangolinRouter(_router) (contracts/dex/MiniChefV2Zapper.sol#40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

MiniChefV2Zapper._swapAndFarm(address,address,uint256,uint256,bool) (contracts/dex/MiniChefV2Zapper.sol#215-273) compares to a boolean constant:
        -farmFlag == true (contracts/dex/MiniChefV2Zapper.sol#252)
MiniChefV2Zapper._swapAndFarm(address,address,uint256,uint256,bool) (contracts/dex/MiniChefV2Zapper.sol#215-273) compares to a boolean constant:
        -farmFlag == true (contracts/dex/MiniChefV2Zapper.sol#266)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Different versions of Solidity is used:
        - Version used: ['>=0.4.0', '>=0.5.0', '>=0.6.0', '>=0.6.2', '^0.8.0']
        - ^0.8.0 (contracts/dex/MiniChefV2Zapper.sol#1)
        - ^0.8.0 (contracts/dex/interfaces/IWAVAX.sol#2)
        - >=0.5.0 (contracts/pangolin-core/interfaces/IERC20.sol#1)
        - >=0.5.0 (contracts/pangolin-core/interfaces/IPangolinPair.sol#1)
        - >=0.4.0 (contracts/pangolin-lib/libraries/Babylonian.sol#3)
        - >=0.6.0 (contracts/pangolin-lib/libraries/TransferHelper.sol#3)
        - >=0.6.2 (contracts/pangolin-periphery/interfaces/IPangolinRouter.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version^0.8.0 (contracts/dex/MiniChefV2Zapper.sol#1) allows old versions
Pragma version^0.8.0 (contracts/dex/interfaces/IWAVAX.sol#2) allows old versions
Pragma version>=0.5.0 (contracts/pangolin-core/interfaces/IERC20.sol#1) allows old versions
Pragma version>=0.5.0 (contracts/pangolin-core/interfaces/IPangolinPair.sol#1) allows old versions
Pragma version>=0.4.0 (contracts/pangolin-lib/libraries/Babylonian.sol#3) allows old versions
Pragma version>=0.6.0 (contracts/pangolin-lib/libraries/TransferHelper.sol#3) allows old versions
Pragma version>=0.6.2 (contracts/pangolin-periphery/interfaces/IPangolinRouter.sol#1) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

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

Variable MiniChefV2Zapper.WAVAX (contracts/dex/MiniChefV2Zapper.sol#32) is not in mixedCase
Constant MiniChefV2Zapper.minimumAmount (contracts/dex/MiniChefV2Zapper.sol#33) is not in UPPER_CASE_WITH_UNDERSCORES
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
Function IPangolinRouter.WAVAX() (contracts/pangolin-periphery/interfaces/IPangolinRouter.sol#5) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable IPangolinRouter.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (contracts/pangolin-periphery/interfaces/IPangolinRouter.sol#10) is too similar to IPangolinRouter.addLiquidity(address
,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (contracts/pangolin-periphery/interfaces/IPangolinRouter.sol#11)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

estimateSwap(address,address,uint256) should be declared external:
        - MiniChefV2Zapper.estimateSwap(address,address,uint256) (contracts/dex/MiniChefV2Zapper.sol#296-312)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

- No major issues found by Slither. The reentrancy flagged by Slither is a false positive.

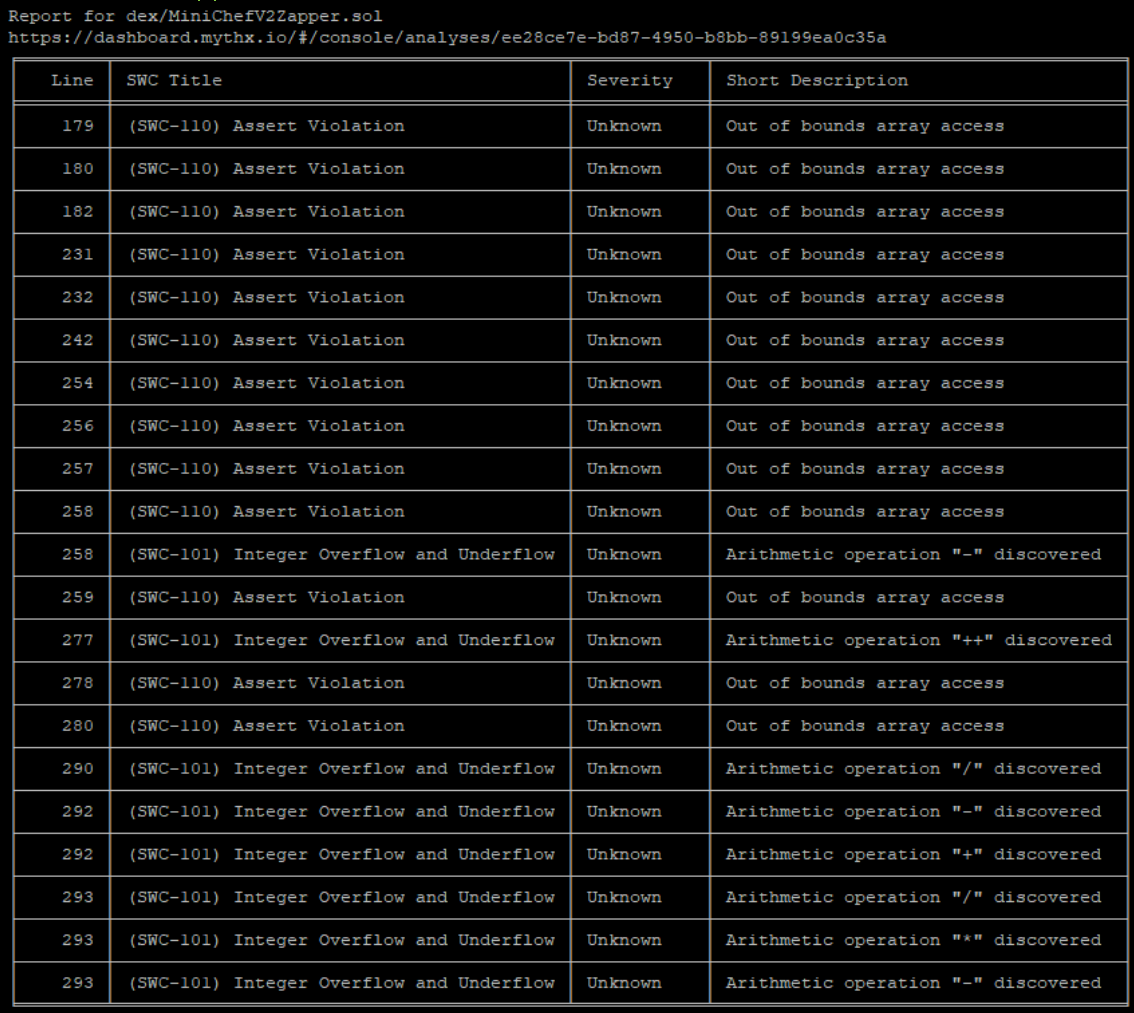
4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits 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 all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

MiniChefV2Zapper.sol



Report for dex/MiniChefV2Zapper.sol
https://dashboard.mythx.io/#/console/analyses/ee28ce7e-bd87-4950-b8bb-89199ea0c35a

Line	SWC Title	Severity	Short Description
179	(SWC-110) Assert Violation	Unknown	Out of bounds array access
180	(SWC-110) Assert Violation	Unknown	Out of bounds array access
182	(SWC-110) Assert Violation	Unknown	Out of bounds array access
231	(SWC-110) Assert Violation	Unknown	Out of bounds array access
232	(SWC-110) Assert Violation	Unknown	Out of bounds array access
242	(SWC-110) Assert Violation	Unknown	Out of bounds array access
254	(SWC-110) Assert Violation	Unknown	Out of bounds array access
256	(SWC-110) Assert Violation	Unknown	Out of bounds array access
257	(SWC-110) Assert Violation	Unknown	Out of bounds array access
258	(SWC-110) Assert Violation	Unknown	Out of bounds array access
258	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
259	(SWC-110) Assert Violation	Unknown	Out of bounds array access
277	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
278	(SWC-110) Assert Violation	Unknown	Out of bounds array access
280	(SWC-110) Assert Violation	Unknown	Out of bounds array access
290	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
292	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
292	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
293	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
293	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
293	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered

- Integer Overflows and Underflows flagged by MythX are false positives, as the contract is using Solidity 0.8.11 version. After the Solidity version 0.8.0 Arithmetic operations revert to underflow and overflow by default.
- Assert violations are false positives.

THANK YOU FOR CHOOSING

HALBORN