// HALBORN

SyncSwap - Pool Contracts

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **November 27th, 2023 - January 15th, 2024**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	01/03/2024
0.2	Document Update	01/15/2024
0.3	Draft Review	01/15/2024
0.4	Draft Review	01/15/2024
1.0	Remediation Plan	01/31/2024
1.1	Remediation Plan Review	01/31/2024
1.2	Remediation Plan Review	02/02/2024

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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

SyncSwap is an exchange featuring a multi-pool design to allow the integration of various pool models.

SyncSwap engaged Halborn to conduct a security assessment on their smart contracts beginning on November 27th, 2023 and ending on January 15th, 2024. The security assessment was scoped to the smart contracts provided in the syncswap/core-contracts-aqua GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

Halborn was provided 6 weeks for the engagement and assigned a team of one full-time security engineer to review the security of the templates and smart contracts in scope. The security team consists of a blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security risks, which were successfully addressed by SyncSwap.

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 assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (Slither).
- Testnet deployment (Foundry, Brownie).

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

Code repositories:

1. SyncSwap Pool Contracts

- Repository: syncswap/core-contracts-aqua
- Commit ID: bc59c9629cbd6ad95d68abc660c32e91dc5fd077
- Smart contracts in scope:
  - contracts/pool/classic/SyncSwapClassicPool.sol
  - contracts/pool/stable/SyncSwapStablePool.sol
  - contracts/pool/crypto/SyncSwapCryptoPool.sol
- Final commit ID: 093af94cb5d82c87574344b52d7e6a8fb95e838d

Out-of-scope

- Third-party libraries and dependencies.
- Economic attacks.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	6	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) VARIABLES UPDATED IMPROPERLY IN DURING FEE MINTING	Low (2.5)	SOLVED - 01/22/2024
(HAL-02) STATE IS NOT RESET PROPERLY WHEN ALL LIQUIDITY IS WITHDRAWN FROM POOL	Low (2.5)	SOLVED - 01/20/2024
(HAL-03) INACCURATE FEE CALCULATION IN BURNSINGLE	Low (2.5)	SOLVED - 01/20/2024
(HAL-04) INACCURATE AMPLIFIER COEFFICIENT CALCULATION	Low (2.5)	SOLVED - 01/22/2024
(HAL-05) IMPROPER INVARIANT UPDATE IN TWEAKPRICE	Low (2.5)	SOLVED - 01/26/2024
(HAL-06) UNNECESSARY DEBUG FUNCTIONS	Low (2.0)	SOLVED - 01/22/2024
(HAL-07) LACK OF TEST COVARAGE	Informational (1.9)	SOLVED - 01/26/2024
(HAL-08) EXTERNAL CALLS	Informational (1.5)	SOLVED - 01/22/2024

FINDINGS & TECH DETAILS

4.1 (HAL-01) VARIABLES UPDATED IMPROPERLY IN DURING FEE MINTING - LOW (2.5)

Description:

It was identified that the _mintProtocolFee() function in the SyncSwapCryptoPool contract might return an invalid total supply value. The function returns without minting any fee if the newVirtualPrice is less than 1e18. However, in these cases, the _totalSupply state variable is already increased with the fee amount before the return statement.

It was also identified that, in the _mintProtocolFee() function, the xcpProfitLast state variable is always updated with the value xcpProfit if the protocol fee is minted. However, after the fee deduction, the current pool profit might be lower than the last profit at previous claim. In these cases, the xcpProfitLast state variable would store an inaccurate data.

Code Location:

The _totalSupply state variable is increased even in the fee is not minted:

Listing 1: contracts/pool/crypto/SyncSwapCryptoPool.sol (Lines 824,828-830)

```
823 if (_xcpProfit > _xcpProfitLast) {
824     uint _protocolFee = getProtocolFee();
825     uint fees = unsafe_div(unsafe_sub(_xcpProfit, _xcpProfitLast)
 ↳ * _protocolFee, 2e5);
826
827     if (fees != 0) {
828         address _feeRecipient = IPoolMaster(master).
 ↳ getFeeRecipient();
829
830         if (_feeRecipient != address(0)) {
831             uint _virtualPrice = virtualPrice;
```

```
832             uint frac = Math.mulDivUnsafeFirst(1e18, _virtualPrice
 ↳ , _virtualPrice - fees) - 1e18;
833             uint liquidity = Math.mulDivUnsafeLast(_totalSupply,
 ↳ frac, 1e18);
834             //uint claimed = ISyncSwapLPToken(token).mint_relative
 ↳ (receiver, frac);
835 
836             if (liquidity != 0) {
837                 uint newTotalSupply = _totalSupply += liquidity;
838                 uint newVirtualPrice = Math.mulDivUnsafeFirst(1e18
 ↳ , _getXCP(_invariant), newTotalSupply);
839 
840                 // Do not claim fees if doing so causes virtual
 ↳ price to drop below 10**18.
841                 if (newVirtualPrice < 1e18) {
842                     return _totalSupply;
843                 }
844 
845                 // Claim admin fees by minting admin's share
846                 // of the pool in LP tokens.
847                 _mint(_feeRecipient, liquidity);
```

The fee is not considered when updating the xcpProfitLast variable:

Listing 2: contracts/pool/crypto/SyncSwapCryptoPool.sol (Lines 839-841)

```
810 if (_xcpProfit > _xcpProfitLast) {
811     uint _protocolFee = getProtocolFee();
812     uint fees = unsafe_div(unsafe_sub(_xcpProfit, _xcpProfitLast)
 ↳ * _protocolFee, 2e5);
813 
814     if (fees != 0) {
815         address _feeRecipient = IPoolMaster(master).
 ↳ getFeeRecipient();
816 
817         if (_feeRecipient != address(0)) {
818             uint _virtualPrice = virtualPrice;
819             uint frac = Math.mulDivUnsafeFirst(1e18, _virtualPrice
 ↳ , _virtualPrice - fees) - 1e18;
820             uint liquidity = Math.mulDivUnsafeLast(_totalSupply,
 ↳ frac, 1e18);
821             //uint claimed = ISyncSwapLPToken(token).mint_relative
```

```
 ↳ (receiver, frac);
822
823             if (liquidity != 0) {
824                 uint newTotalSupply = _totalSupply += liquidity;
825                 uint newVirtualPrice = Math.mulDivUnsafeFirst(1e18
 ↳ , _getXCP(_invariant), newTotalSupply);
826
827                 // Do not claim fees if doing so causes virtual
 ↳ price to drop below 10**18.
828                 if (newVirtualPrice < 1e18) {
829                     return _totalSupply;
830                 }
831
832                 // Claim admin fees by minting admin's share
833                 // of the pool in LP tokens.
834                 _mint(_feeRecipient, liquidity);
835
836                 // Notifies the fee recipient.
837                 IFeeRecipient(_feeRecipient).notifyFees(3, address
 ↳ (this), liquidity, _protocolFee, "");
838
839                 _xcpProfit -= unsafe_mul(fees, 2); // `_xcpProfit`
 ↳  is up-to-date.
840                 xcpProfit = _xcpProfit;
841                 xcpProfitLast = _xcpProfit;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to not increase the _totalSupply before the return statement. It is also recommended to take the fee into account when updating the xcpProfitLast variable.

Remediation Plan:

SOLVED: The SyncSwap team solved the issue in commits b5c270e and c769dc5.

4.2 (HAL-02) STATE IS NOT RESET PROPERLY WHEN ALL LIQUIDITY IS WITHDRAWN FROM POOL - LOW (2.5)

Description:

If all liquidity is withdrawn from the SyncSwapCryptoPool contract, the following mint() call reinstates the pool's state variables. However, it was identified that in these cases, the xcpProfitLast variable is not reset back to zero, resulting in the protocol not minting fees until the profits at previous claim are reached again.

Code Location:

If all liquidity is withdrawn, the pool reinstates the state variables. However, xcpProfitLast is not set to 0:

Listing 3: contracts/pool/crypto/SyncSwapCryptoPool.sol (Lines 325-327)

```
324     } else {
325         invariantLast = params.newInvariant;
326         virtualPrice = 1e18;
327         xcpProfit = 1e18;
328 
329         // Ensures ratio between price scale and reserves.
330         uint newPriceScale = Math.mulDivUnsafeFirst(1e18, it.xp0,
 ↳ unsafe_mul(params.balance1, token1PrecisionMultiplier));
331         if (newPriceScale > Math.mulUnsafeFirst(2, _priceScale) ||
 ↳  newPriceScale < Math.divUnsafeLast(_priceScale, 2)) {
332             revert Loss();
333         }
334 
335         _mint(params.to, params.liquidity);
336     }
```

Protocol fee is not minted until the previous pool profit is not reached:

Listing 4: contracts/pool/crypto/SyncSwapCryptoPool.sol (Line 798)

```
787     function _mintProtocolFee(uint _totalSupply, uint _invariant)
 ↳ private returns (uint) {
788         // Do not claim admin fees if:
789         // 1. insufficient profits accrued since last claim, and
790         // 2. there are less than 10**18 (or 1 unit of) lp tokens,
 ↳  else it can lead
791         //    to manipulated virtual prices.
792         if (_totalSupply <= 1e18) {
793             return _totalSupply;
794         }
795
796         uint _xcpProfit = xcpProfit; // Current pool profits.
797         uint _xcpProfitLast = xcpProfitLast; // Profits at
 ↳ previous claim.
798         if (_xcpProfit <= _xcpProfitLast) {
799             return _totalSupply;
800         }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:L/R:N/S:U (2.5)

Recommendation:

It is recommended to also reset the xcpProfitLast variable to zero.

Remediation Plan:

SOLVED: The SyncSwap team solved the issue in commit d229830.

4.3 (HAL-03) INACCURATE FEE CALCULATION IN BURNSINGLE - LOW (2.5)

It was identified that the burnSingle() function in the SyncSwapCryptoPool contract calculates the protocol fee based on the initial reserve balances. This might result in inaccurate calculations, as the amount should be based on the resulting balance.

Code Location:

The fee calculation is based on the initial reserve balances:

Listing 5: contracts/pool/crypto/SyncSwapCryptoPool.sol (Lines 943-944,955-959)

```
930     function _calculateSingleWithdrawAmount(
931         address _sender,
932         uint _totalSupply,
933         uint a,
934         uint gamma,
935         uint liquidity,
936         bool isToken0Out,
937         bool calculatePrice
938     ) internal view returns (uint, uint, uint, uint, uint, uint24)
 ↳  {
939         CalculateSingleWithdrawAmountArgs memory it;
940         (it.reserve0, it.reserve1) = getReserves();
941 
942         it.xp1PriceScale = priceScale * token1PrecisionMultiplier;
943         it.xp0 = it.reserve0 * token0PrecisionMultiplier;
944         it.xp1 = Math.mulDivUnsafeLast(it.reserve1, it.
 ↳ xp1PriceScale, 1e18);
945 
946         if (futureParamsTime > block.timestamp) {
947             it.oldInvariant = MATH.computeD(a, gamma, it.xp0, it.
 ↳ xp1);
948         } else {
949             it.oldInvariant = invariantLast;
950         }
```

```
951
952         it.newInvariant = it.oldInvariant;
953
954         // Charge the fee on D, not on y, e.g. reducing invariant
 ↳ LESS than charging the user.
955         if (isToken0Out) {
956             it.fee = _getFee(_sender, token1, token0, it.xp0, it.
 ↳ xp1);
957         } else {
958             it.fee = _getFee(_sender, token0, token1, it.xp0, it.
 ↳ xp1);
959         }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:L/R:N/S:U (2.5)

Recommendation:

It is recommended to base the fee calculation on the resulting balance.

Remediation Plan:

SOLVED: The SyncSwap team solved the issue in commit ff5eca1 by updating the fee calculation algorithm.

4.4 (HAL-04) INACCURATE AMPLIFIER COEFFICIENT CALCULATION - LOW (2.5)

Description:

It was identified that the amplifier coefficient calculation in the SyncSwapStablePool contract is incorrect, as the difference is multiplied by the duration instead of the elapsed time.

Code Location:

Listing 6: contracts/pool/stable/SyncSwapStablePool.sol (Line 589)

```
575     function getA() public view returns (uint64) {
576         PoolParams memory params = poolParams;
577
578         // Ramps A.
579         if (uint64(block.timestamp) < params.futureTime) {
580             if (params.futureA < params.initialA) {
581                 uint64 diff;
582                 uint64 elapsed;
583                 uint64 duration;
584                 unchecked {
585                     diff = params.initialA - params.futureA;
586                     elapsed = uint64(block.timestamp) - params.
 ↳ initialTime;
587                     duration = params.futureTime - params.
 ↳ initialTime;
588                 }
589                 return params.initialA - diff * duration /
 ↳ duration;
590             }
591         }
592
593         return params.futureA;
594     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to correct the equation of the amplifier coefficient calculation.

Remediation Plan:

SOLVED: The SyncSwap team solved the issue in commit 7723da6.

4.5 (HAL-05) IMPROPER INVARIANT UPDATE IN TWEAKPRICE - LOW (2.5)

Description:

It was identified that the invariantLast state variable is still updated at the end of the _tweakPrice() function, even if the operation profit is below the threshold.

Code Location:

The newInvariant variable still overwrites the invariantLast variable at the end of the function:

Listing 7: contracts/pool/crypto/SyncSwapCryptoPool.sol (Lines 1240,1256-1263,1271)

```
239                 // Calculate "extended constant product" invariant
 ↳  xCP and virtual price.
240                 newInvariant = MATH.computeD(a, gamma, _xp0, _xp1)
 ↳ ;
241
242                 _xp0 = Math.divUnsafeLast(newInvariant, 2);
243                 _xp1 = Math.mulDivUnsafeFirst(
244                     1e18, newInvariant, Math.mulUnsafeFirst(2,
 ↳ newPriceScale)
245                 );
246
247                 // ---------- Calculate new virtual price using
 ↳ new xp and D.
248                 //            Reuse `oldVirtualPrice` (but it has
 ↳ new virtual price).
249
250                 // We reuse `oldVirtualPrice` here but it's not
 ↳ old anymore.
251                 it.oldVirtualPrice = Math.mulDivUnsafeFirst(
252                     1e18, Math.geometricMean(_xp0, _xp1), it.
 ↳ totalSupply
253                 );
254
```

```
255                 // ---------------------------- Proceed if we've
 ↳ got enough profit.
256                 if (it.oldVirtualPrice > 1e18) {
257                     if (Math.mulUnsafeFirst(2, it.oldVirtualPrice)
 ↳  - 1e18 > it.xcpProfit) {
258                         priceScale = newPriceScale;
259                         invariantLast = newInvariant;
260                         virtualPrice = it.oldVirtualPrice;
261                         return newInvariant;
262                     }
263                 }
264             }
265         }
266
267         // --------- `priceScale` was not adjusted. Update the
 ↳ profit counter and D.
268
269         // If we are here, the `priceScale` adjustment did not
 ↳ happen,
270         // Still need to update the profit counter and D.
271         invariantLast = newInvariant;
272         virtualPrice = it.virtualPrice;
273         return newInvariant;
274     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended not to update the last invariant and instead return the original value if the operation profit is below the threshold.

Remediation Plan:

SOLVED: The SyncSwap team solved the issue in commit 093af94.

4.6 (HAL-06) UNNECESSARY DEBUG FUNCTIONS - LOW (2.0)

Description:

It was identified in the SyncSwapStablePool contract that the Owner can adjust the price configuration by using the setInitialPrice() function. This function is only intended for testing and is not required in the protocol.

Code Location:

Listing 8: contracts/pool/crypto/SyncSwapCryptoPool.sol

```
352     function setInitialPrice(uint _initialPrice) external
  ↳ onlyOwner {
353         priceScale = _initialPrice;
354         _priceOracle = _initialPrice;
355         lastPrices = _initialPrice;
356         lastPricesTimestamp = block.timestamp;
357     }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (2.0)

Recommendation:

It is recommended to remove the function if it is not needed.

Remediation Plan:

SOLVED: The SyncSwap team solved the issue in commit 43b9676 by removing the function from the contract.

4.7 (HAL-07) LACK OF TEST COVARAGE - INFORMATIONAL (1.9)

Description:

It was identified that the pool contracts lack test coverage. The contracts, especially the SyncSwapCryptoPool, use complex mathematical formulas to rebalance the liquidity, calculate prices, and perform swaps. Unlike traditional software, smart contracts cannot be modified unless deployed using a proxy contract.

The lack of extensive testing of the protocol functions, mathematical formulas, and modules increases the risk of hidden errors in the protocol. Because the pool contracts are not upgradable, it is also not possible to correct these errors in the deployed contracts, and funds might get stuck in the protocol.

It was also identified that there are minor deviations between the expected and received results in the unit tests of the functions of the SyncSwapCryptoPool contract:

```
> forge test --match-test test_AquaLPEqualCurveCryptoPool -vv
[⠒] Compiling...
No files changed, compilation skipped

Running 1 test for foundry-test/AquaPool/TestAquaPool.t.sol:TestAquaPool
[FAIL. Reason: Assertion failed.] test_AquaLPEqualCurveCryptoPool() (gas: 477968)
Logs:
  0x3D7Ebc40AF7092E3F1C81F2e996cbA5Cae2090d7
  Error: a == b not satisfied [uint]
    Expected: 1000000260033029860
      Actual: 1000000259999999860
  Error: a == b not satisfied [uint]
    Expected: 1000000260033029860
      Actual: 1000000259999999860
  Error: a == b not satisfied [uint]
    Expected: 6248753262292510266969
      Actual: 6248753261879639201042
  Error: a == b not satisfied [uint]
    Expected: 1246737707489733033
      Actual: 1246738120360798960

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 930.15ms

Ran 1 test suites: 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in foundry-test/AquaPool/TestAquaPool.t.sol:TestAquaPool
[FAIL. Reason: Assertion failed.] test_AquaLPEqualCurveCryptoPool() (gas: 477968)
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:L/A:N/D:M/Y:N/R:N/S:U (1.9)

Recommendation:

It is recommended performing as many test cases as possible to cover all conceivable scenarios in the pool contracts, including different edge cases.

It is also recommended to test the formulas of the contract using unit tests to ensure the calculations are aligned with business requirements and that any deviations are within the expected limits.

Remediation Plan:

SOLVED: The SyncSwap team solved the issue in commit 6f56db6 by adding additional tests.

4.8 (HAL-08) EXTERNAL CALLS - INFORMATIONAL (1.5)

Description:

It was identified that the pool contracts have external calls. If the external call reverts, the calling function reverts as well.

Code Location:

Example external calls in the SyncSwapCryptoPool contract:

Listing 9: contracts/pool/crypto/SyncSwapCryptoPool.sol (Lines 754,755)

```
747     function _getFee(
748         address _sender,
749         address _tokenIn,
750         address _tokenOut,
751         uint xp0,
752         uint xp1
753     ) internal view returns (uint24) {
754         IFeeManagerV2 feeManager = IFeeManagerV2(IPoolMaster(
 ↳ master).feeManager());
755         IFeeManagerV2.FeeData memory data = feeManager.
 ↳ getSwapFeeParams(address(this), _sender, _tokenIn, _tokenOut, "");
756         uint f = xp0 + xp1;
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:H/D:N/Y:N/R:N/S:U (1.5)

Recommendation:

It is recommended to make the external calls non-revertable in the pool contracts.

Remediation Plan:

SOLVED: The SyncSwap team solved the issue in commit ef478cd.

MANUAL TESTING

In the manual testing phase, the following scenarios were simulated. The scenarios listed below were selected based on the severity of the vulnerabilities Halborn was testing the contracts for.

5.1 POOL-DEPLOYMENTS

Description:

The pool contracts are deployed using a factory. It was tested that the deployment parameters are correctly configured.

Results:

- It was verified that the pool contracts are deployed according to their parameters. However, note that, the factory must ensure that the parameters are valid.
- It was identified that the pool contracts are not upgradeable. Therefore, fixing identified errors in the deployed contracts would not be possible.

5.2 ADMIN FUNCTIONS

Description:

The SyncSwapCryptoPool contract has admin functions to configure parameters used for price calculations and rebalancing. Halborn reviewed the authorization of these functions.

Results:

- It was verified that only the Owner can call the admin functions.

Only the owner can call the admin functions of the SyncSwapCryptoPool contract:

MANUAL TESTING

```
Running 1 test for foundry-test/AquaPool/HalbornTest.t.sol:HalbornTest
[PASS] test_addLiquidity2() (gas: 818801)
Logs:
  Initial price:  800000000000000000
  ** Pool data:
    total supply:  11180339887498948482045
    token0:  100000000000000000000
    token1:  125000000000000000000
    reserve0:  100000000000000000000
    reserve1:  125000000000000000000
    fee paid:  0
  ** Alice: balances:
    user token0 balance:  100000000000000000000
    user token1 balance:  100000000000000000000
    user pool share balance:  11180339887498948482045
  ** Bob: balances:
    user token0 balance:  100000000000000000000
    user token1 balance:  100000000000000000000
    user pool share balance:  0
  ---- ADDLIQUIDITY ----
  ** Pool data:
    total supply:  12253838141467910337862
    token0:  100000000000000000000
    token1:  150000000000000000000
    reserve0:  100000000000000000000
    reserve1:  150000000000000000000
    fee paid:  599389978769303183
  ** Alice: balances:
    user token0 balance:  100000000000000000000
    user token1 balance:  750000000000000000000
    user pool share balance:  1225323875148914103467 9
```

5.3 ADDING LIQUIDITY

Description:

It is possible to add liquidity to the pool contracts using the `mint` function. It was tested that this function operates correctly.

Results:

- It was verified that it is not possible to manipulate share prices due to rounding errors using the initial deposit. However, it is still recommended to lock some funds in the pool contracts. It is recommended to do this in the same transaction in which the pools are deployed, for example, using the factory.
- It is not possible to control the slippage using the mint functions. However, the pool contracts are not intended to be called directly, and the swap router contract can ensure that sufficient LP tokens are minted.
- It was verified that the protocol fee was deducted and transferred to the correct address.
- Several scenarios were tested, and no errors were discovered in the function during the assessment. However, it was identified that the contracts lack test coverage. It was also identified that some test cases of the `SyncSwapCryptoPool` contract failed due to slight differences between the expected and received results.

In the following example, the changes in the contract's internal state were observed as shares were minted for the test user:

```
Running 1 test for foundry-test/AquaPool/HalbornTest.t.sol:HalbornTest
[PASS] test_addLiquidity2() (gas: 818801)
Logs:
  Initial price:  800000000000000000
  ** Pool data:
    total supply:  11180339887498948482045
    token0:  100000000000000000000
    token1:  125000000000000000000
    reserve0:  100000000000000000000
    reserve1:  125000000000000000000
    fee paid:  0
  ** Alice: balances:
    user token0 balance:  100000000000000000000
    user token1 balance:  100000000000000000000
    user pool share balance:  11180339887498948482045
  ** Bob: balances:
    user token0 balance:  100000000000000000000
    user token1 balance:  100000000000000000000
    user pool share balance:  0
  ---- ADDLIQUIDITY ----
  ** Pool data:
    total supply:  12253838141467910337862
    token0:  100000000000000000000
    token1:  150000000000000000000
    reserve0:  100000000000000000000
    reserve1:  150000000000000000000
    fee paid:  599389978769303183
  ** Alice: balances:
    user token0 balance:  100000000000000000000
    user token1 balance:  750000000000000000000
    user pool share balance:  1225323875148914103467 9
```

5.4 REMOVING LIQUIDITY

It is possible to remove liquidity from the pool contracts using the `burn` and `burnSingle` functions. It was tested that this function operates correctly.

Results:

- It is not possible to control the slippage using the mint functions. However, the pool contracts are not intended to be called directly, and the swap router contract can ensure that sufficient tokens are received.
- It was identified that it is possible to withdraw all funds from the pools, and therefore, it is important to lock some funds in the pool contracts. It is recommended to do this in the same transaction in which the pools are deployed, for example, using the factory.
- It was verified that the protocol fee was deducted and transferred to the correct address.
- Several scenarios were tested, and no errors were discovered in the function during the assessment. However, it was identified that the contracts lack test coverage. It was also identified that some test cases of the `SyncSwapCryptoPool` contract failed due to slight differences between the expected and received results.

In the following example, all funds were withdrawn from the SyncSwapCryptoPool pool contract. Note that after the first transaction, the shares minted as a fee also need to be removed to empty the pool.

```
Running 1 test for foundry-test/AquaPool/HalbornTest.t.sol:HalbornTest
[PASS] test_removeLiquidity3() (gas: 143942)
Logs:
  Initial price:  80000000000000000
  test_removeLiquidity3
  ** Pool data:
    total supply:  1118033988749894848 2045
    token0:  100000000000000000000
    token1:  125000000000000000000
    reserve0:  100000000000000000000
    reserve1:  125000000000000000000
    fee paid:  0
  ** Alice pool balance: 1118033988749894848 2045
  ---- BURNLIQUIDITY ----
  ** Pool data:
    total supply:  0
    token0:  0
    token1:  0
    reserve0:  0
    reserve1:  0
    fee paid:  0

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.13s
```

5.5 SWAPS

It is possible to exchange tokens using the swap function of the pool contracts. It was tested that this function operates correctly.

- It is not possible to control the slippage using the mint functions. However, the pool contracts are not intended to be called directly, and the swap router contract can ensure that sufficient tokens are received.
- It was verified that the protocol fee was deducted and transferred to the correct address.
- Several scenarios were tested, and no errors were discovered in the function during the assessment. However, it was identified that the contracts lack test coverage. It was also identified that some test cases of the SyncSwapCryptoPool contract failed due to slight differences between the expected and received results.

In the following example, the changes in the contract's internal state and the user's token balances were observed as swapping was performed.

```
Initial price:  80000000000000000
---- ADDLIQUIDITY ----
** Pool data:
  total supply:  12253838141467910337862
  token0:  100000000000000000000
  token1:  150000000000000000000
  reserve0:  100000000000000000000
  reserve1:  150000000000000000000
  fee paid:  599389978769303183
** Bob: balances:
  user token0 balance:  100000000000000000000
  user token1 balance:  100000000000000000000
  user pool share balance:  0
---- SWAP (BOB) ----
** Pool data:
  total supply:  12253838141467910337862
  token0:  110000000000000000000
  token1:  13658065395236040223459
  reserve0:  110000000000000000000
  reserve1:  13658065395236040223459
  fee paid:  599389978769303183
** Bob: balances:
  user token0 balance:  90000000000000000000
  user token1 balance:  11341934604763959776541
  user pool share balance:  0
```

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.
The security team assessed all findings identified by the Slither software, however, findings with severity Information and Optimization are not included in the below results for the sake of report readability.

Results:

contracts/pool/classic/SyncSwapClassicPool.sol

Slither results for SyncSwapClassicPool.sol	
Finding	**Impact**
SyncSwapClassicPool.burn(bytes,address,address,bytes).params (contracts/pool/classic/SyncSwapClassicPool.sol#186) is a local variable never initialized	Medium
SyncSwapClassicPool.swap(bytes,address,address,bytes).params (contracts/pool/classic/SyncSwapClassicPool.sol#332) is a local variable never initialized	Medium
SyncSwapClassicPool.burnSingle(bytes,address,address,bytes).params (contracts/pool/classic/SyncSwapClassicPool.sol#246) is a local variable never initialized	Medium
SyncSwapClassicPool.mint(bytes,address,address,bytes).params (contracts/pool/classic/SyncSwapClassicPool.sol#114) is a local variable never initialized	Medium

Finding	Impact
SyncSwapClassicPool._transferTokens(address,address,uint256,uint8) (contracts/pool/classic/SyncSwapClassicPool.sol#408-421) ignores return value by IVault(vault).deposit(token,to) (contracts/pool/classic/SyncSwapClassicPool.sol#412)	Medium
SyncSwapClassicPool.constructor()._token1 (contracts/pool/classic/SyncSwapClassicPool.sol#63) lacks a zero-check on : - (token0,token1) = (_token0,_token1) (contracts/pool/classic/SyncSwapClassicPool.sol#69)	Low
SyncSwapClassicPool.constructor()._token0 (contracts/pool/classic/SyncSwapClassicPool.sol#63) lacks a zero-check on : - (token0,token1) = (_token0,_token1) (contracts/pool/classic/SyncSwapClassicPool.sol#69)	Low
SyncSwapClassicPool.constructor()._master (contracts/pool/classic/SyncSwapClassicPool.sol#64) lacks a zero-check on : - master = _master (contracts/pool/classic/SyncSwapClassicPool.sol#66) - wETH = IPoolMaster(_master).wETH() (contracts/pool/classic/SyncSwapClassicPool.sol#67) - vault = IPoolMaster(_master).vault() (contracts/pool/classic/SyncSwapClassicPool.sol#68)	Low

Finding	Impact
Reentrancy in SyncSwapClassicPool.burn(bytes,address,address,bytes) (contracts/pool/classic/SyncSwapClassicPool.sol#180-235):External calls: - (_feeOn,params.totalSupply) = _mintProtocolFee(params.balance0,params.balance1,0) (contracts/pool/classic/SyncSwapClassicPool.sol#194) - IFeeRecipient(_feeRecipient).notifyFees(1,address(this),_liquidity,_protocolFee,) (contracts/pool/classic/SyncSwapClassicPool.sol#473) State variables written after the call(s): - _updateReserves(params.balance0,params.balance1) (contracts/pool/classic/SyncSwapClassicPool.sol#212) - (reserve0_,reserve1_) = (uint128(_balance0),uint128(_balance1)) (contracts/pool/classic/SyncSwapClassicPool.sol#404) - _updateReserves(params.balance0,params.balance1) (contracts/pool/classic/SyncSwapClassicPool.sol#212) - (reserve0_,reserve1_) = (uint128(_balance0),uint128(_balance1)) (contracts/pool/classic/SyncSwapClassicPool.sol#404)	Low
Reentrancy in SyncSwapClassicPool.burnSingle(bytes,address,address,bytes) (contracts/pool/classic/SyncSwapClassicPool.sol#240-322):External calls: - (_feeOn,params.totalSupply) = _mintProtocolFee(params.balance0,params.balance1,0) (contracts/pool/classic/SyncSwapClassicPool.sol#254) - IFeeRecipient(_feeRecipient).notifyFees(1,address(this),_liquidity,_protocolFee,) (contracts/pool/classic/SyncSwapClassicPool.sol#473) State variables written after the call(s): - _updateReserves(params.balance0,params.balance1) (contracts/pool/classic/SyncSwapClassicPool.sol#299) - (reserve0_,reserve1_) = (uint128(_balance0),uint128(_balance1)) (contracts/pool/classic/SyncSwapClassicPool.sol#404) - _updateReserves(params.balance0,params.balance1) (contracts/pool/classic/SyncSwapClassicPool.sol#299) - (reserve0_,reserve1_) = (uint128(_balance0),uint128(_balance1)) (contracts/pool/classic/SyncSwapClassicPool.sol#404)	Low
End of table for SyncSwapClassicPool.sol	

contracts/pool/stable/SyncSwapStablePool.sol

Slither results for SyncSwapStablePool.sol	
Finding	**Impact**
SyncSwapStablePool.swap(bytes,address,address,bytes).params (contracts/pool/stable/SyncSwapStablePool.sol#363) is a local variable never initialized	Medium
SyncSwapStablePool.mint(bytes,address,address,bytes).params (contracts/pool/stable/SyncSwapStablePool.sol#139) is a local variable never initialized	Medium
SyncSwapStablePool.burn(bytes,address,address,bytes).params (contracts/pool/stable/SyncSwapStablePool.sol#213) is a local variable never initialized	Medium
SyncSwapStablePool.burnSingle(bytes,address,address,bytes).params (contracts/pool/stable/SyncSwapStablePool.sol#276) is a local variable never initialized	Medium
SyncSwapStablePool._transferTokens(address,address,uint256,uint8) (contracts/pool/stable/SyncSwapStablePool.sol#445-457) ignores return value by IVault(vault).deposit(token,to) (contracts/pool/stable/SyncSwapStablePool.sol#448)	Medium
SyncSwapStablePool.constructor()._token1 (contracts/pool/stable/SyncSwapStablePool.sol#82) lacks a zero-check on : - (token0,token1,token0PrecisionMultiplier,token1PrecisionMultiplier) = (_token0,_token1,_token0PrecisionMultiplier,_token1PrecisionMultiplier) (contracts/pool/stable/SyncSwapStablePool.sol#90-92)	Low
SyncSwapStablePool.constructor()._master (contracts/pool/stable/SyncSwapStablePool.sol#85) lacks a zero-check on : - master = _master (contracts/pool/stable/SyncSwapStablePool.sol#87) - wETH = IPoolMaster(_master).wETH() (contracts/pool/stable/SyncSwapStablePool.sol#88) - vault = IPoolMaster(_master).vault() (contracts/pool/stable/SyncSwapStablePool.sol#89)	Low

Finding	Impact
SyncSwapStablePool.constructor()._token0 (contracts/pool/stable/SyncSwapStablePool.sol#82) lacks a zero-check on : - (token0,token1,token0PrecisionMultiplier,token1PrecisionMultiplier) = (_token0,_token1,_token0PrecisionMultiplier,_token1PrecisionMultiplier) (contracts/pool/stable/SyncSwapStablePool.sol#90-92)	Low
Reentrancy in SyncSwapStablePool.burnSingle(bytes,address,address,bytes) (contracts/pool/stable/SyncSwapStablePool.sol#270-353):External calls: - (_feeOn,params.totalSupply) = _mintProtocolFee(_a,params.balance0,params.balance1,0) (contracts/pool/stable/SyncSwapStablePool.sol#286) - IFeeRecipient(_feeRecipient).notifyFees(2,address(this),_liquidity,_protocolFee,) (contracts/pool/stable/SyncSwapStablePool.sol#508) State variables written after the call(s): - _updateReserves(params.balance0,params.balance1) (contracts/pool/stable/SyncSwapStablePool.sol#333) - (reserve0_,reserve1_) = (uint128(_balance0),uint128(_balance1)) (contracts/pool/stable/SyncSwapStablePool.sol#441) - _updateReserves(params.balance0,params.balance1) (contracts/pool/stable/SyncSwapStablePool.sol#333) - (reserve0_,reserve1_) = (uint128(_balance0),uint128(_balance1)) (contracts/pool/stable/SyncSwapStablePool.sol#441)	Low

Finding	Impact
Reentrancy in SyncSwapStablePool.burn(bytes,address,address,bytes) (contracts/pool/stable/SyncSwapStablePool.sol#207-265):External calls: - (_feeOn,params.totalSupply) = _mintProtocolFee(_a,params.balance0,params.balance1,0) (contracts/pool/stable/SyncSwapStablePool.sol#223) - IFeeRecipient(_feeRecipient).notifyFees(2,address(this),_liquidity,_protocolFee,) (contracts/pool/stable/SyncSwapStablePool.sol#508) State variables written after the call(s): - _updateReserves(params.balance0,params.balance1) (contracts/pool/stable/SyncSwapStablePool.sol#242) - (reserve0_,reserve1_) = (uint128(_balance0),uint128(_balance1)) (contracts/pool/stable/SyncSwapStablePool.sol#441) - _updateReserves(params.balance0,params.balance1) (contracts/pool/stable/SyncSwapStablePool.sol#242) - (reserve0_,reserve1_) = (uint128(_balance0),uint128(_balance1)) (contracts/pool/stable/SyncSwapStablePool.sol#441)	Low
SyncSwapStablePool.rampA(uint64,uint64) (contracts/pool/stable/SyncSwapStablePool.sol#601-609) uses timestamp for comparisons Dangerous comparisons: - require(bool)(_futureTime > uint64(block.timestamp) + 3600) (contracts/pool/stable/SyncSwapStablePool.sol#602) - require(bool)(_initialA != _futureA) (contracts/pool/stable/SyncSwapStablePool.sol#606)	Low
SyncSwapStablePool.getA() (contracts/pool/stable/SyncSwapStablePool.sol#575-594) uses timestamp for comparisons Dangerous comparisons: - uint64(block.timestamp) < params.futureTime (contracts/pool/stable/SyncSwapStablePool.sol#579) - params.futureA < params.initialA (contracts/pool/stable/SyncSwapStablePool.sol#580)	Low
End of table for SyncSwapStablePool.sol	

contracts/pool/crypto/SyncSwapCryptoPool.sol

Slither results for SyncSwapCryptoPool.sol	
Finding	**Impact**
SyncSwapCryptoPool._getFee(address,address,address,uint256,uint256) (contracts/pool/crypto/SyncSwapCryptoPool.sol#747-764) performs a multiplication on the result of a division: - f = unsafe_mul(data.gamma,1e18) / (unsafe_add(data.gamma,1e18) - unsafe_div(4e18 * xp0 / f * xp1,f)) (contracts/pool/crypto/SyncSwapCryptoPool.sol#757-759) - uint24(unsafe_div(data.minFee * f + data.maxFee * (1e18 - f),1e18)) (contracts/pool/crypto/SyncSwapCryptoPool.sol#760-763)	Medium
SyncSwapCryptoPool._getFee(address,address,address,uint256,uint256) (contracts/pool/crypto/SyncSwapCryptoPool.sol#747-764) performs a multiplication on the result of a division: - f = unsafe_mul(data.gamma,1e18) / (unsafe_add(data.gamma,1e18) - unsafe_div(4e18 * xp0 / f * xp1,f)) (contracts/pool/crypto/SyncSwapCryptoPool.sol#757-759)	Medium
SyncSwapCryptoPool.mint(bytes,address,address,bytes) (contracts/pool/crypto/SyncSwapCryptoPool.sol#222-360) uses a dangerous strict equality: - params.amount0 == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#299)	Medium
SyncSwapCryptoPool.mint(bytes,address,address,bytes) (contracts/pool/crypto/SyncSwapCryptoPool.sol#222-360) uses a dangerous strict equality: - params.reserve0 == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#274)	Medium
SyncSwapCryptoPool._transferTokens(address,address,uint256,uint8) (contracts/pool/crypto/SyncSwapCryptoPool.sol#711-723) uses a dangerous strict equality: - withdrawMode == 1 && token == wETH (contracts/pool/crypto/SyncSwapCryptoPool.sol#716)	Medium
SyncSwapCryptoPool._tweakPrice(uint256,uint256,uint256,uint256,uint256,uint256) (contracts/pool/crypto/SyncSwapCryptoPool.sol#1111-1274) uses a dangerous strict equality: - newInvariant == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#1160)	Medium

Finding	Impact
SyncSwapCryptoPool.burnSingle(bytes,address,address,bytes) (contracts/pool/crypto/SyncSwapCryptoPool.sol#435-522) uses a dangerous strict equality: - require(bool)(params.tokenOut == token0) (contracts/pool/crypto/SyncSwapCryptoPool.sol#478)	Medium
SyncSwapCryptoPool.mint(bytes,address,address,bytes) (contracts/pool/crypto/SyncSwapCryptoPool.sol#222-360) uses a dangerous strict equality: - params.amount0 == 0 || params.amount1 == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#298)	Medium
SyncSwapCryptoPool.mint(bytes,address,address,bytes) (contracts/pool/crypto/SyncSwapCryptoPool.sol#222-360) uses a dangerous strict equality: - params.liquidity == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#268)	Medium
SyncSwapCryptoPool._transferTokens(address,address,uint256,uint8) (contracts/pool/crypto/SyncSwapCryptoPool.sol#711-723) uses a dangerous strict equality: - withdrawMode == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#712)	Medium
SyncSwapCryptoPool.burnSingle(bytes,address,address,bytes) (contracts/pool/crypto/SyncSwapCryptoPool.sol#435-522) uses a dangerous strict equality: - (params.amountOut,_price,_newInvariant,_xp0,_xp1,params.swapFee) = _calculateSingleWithdrawAmount(_sender,params.totalSupply,a,gamma,params.liquidity,params.tokenOut == token0,true) (contracts/pool/crypto/SyncSwapCryptoPool.sol#458-466)	Medium
SyncSwapCryptoPool.burnSingle(bytes,address,address,bytes) (contracts/pool/crypto/SyncSwapCryptoPool.sol#435-522) uses a dangerous strict equality: - params.tokenOut == token1 (contracts/pool/crypto/SyncSwapCryptoPool.sol#502)	Medium
SyncSwapCryptoPool.burnSingle(bytes,address,address,bytes) (contracts/pool/crypto/SyncSwapCryptoPool.sol#435-522) uses a dangerous strict equality: - params.tokenOut == token1 (contracts/pool/crypto/SyncSwapCryptoPool.sol#471)	Medium

Finding	Impact
Reentrancy in SyncSwapCryptoPool._mintProtocolFee(uint256,uint256) (contracts/pool/crypto/SyncSwapCryptoPool.sol#787-856):External calls: - IFeeRecipient(_feeRecipient).notifyFees(3,address(this),liquidity,_protocolFee,) (contracts/pool/crypto/SyncSwapCryptoPool.sol#837) State variables written after the call(s): - virtualPrice = newVirtualPrice (contracts/pool/crypto/SyncSwapCryptoPool.sol#847)SyncSwapCryptoPool.virtualPrice (contracts/pool/crypto/SyncSwapCryptoPool.sol#107) can be used in cross function reentrancies: - SyncSwapCryptoPool.getLiquidityPrice() (contracts/pool/crypto/SyncSwapCryptoPool.sol#1282-1284) - SyncSwapCryptoPool.virtualPrice (contracts/pool/crypto/SyncSwapCryptoPool.sol#107) - xcpProfit = _xcpProfit (contracts/pool/crypto/SyncSwapCryptoPool.sol#840)SyncSwapCryptoPool.xcpProfit (contracts/pool/crypto/SyncSwapCryptoPool.sol#110) can be used in cross function reentrancies: - SyncSwapCryptoPool.xcpProfit (contracts/pool/crypto/SyncSwapCryptoPool.sol#110) - xcpProfitLast = _xcpProfit (contracts/pool/crypto/SyncSwapCryptoPool.sol#841)SyncSwapCryptoPool.xcpProfitLast (contracts/pool/crypto/SyncSwapCryptoPool.sol#112) can be used in cross function reentrancies: - SyncSwapCryptoPool.xcpProfitLast (contracts/pool/crypto/SyncSwapCryptoPool.sol#112)	Medium
SyncSwapCryptoPool._getAmountOut(address,uint256,uint256,uint256,uint256,uint256,uint256,bool).it (contracts/pool/crypto/SyncSwapCryptoPool.sol#635) is a local variable never initialized	Medium
SyncSwapCryptoPool.burnSingle(bytes,address,address,bytes).params (contracts/pool/crypto/SyncSwapCryptoPool.sol#441) is a local variable never initialized	Medium
SyncSwapCryptoPool.swap(bytes,address,address,bytes).params (contracts/pool/crypto/SyncSwapCryptoPool.sol#532) is a local variable never initialized	Medium

Finding	Impact
SyncSwapCryptoPool.mint(bytes,address,address,bytes).p (contracts/pool/crypto/SyncSwapCryptoPool.sol#296) is a local variable never initialized	Medium
SyncSwapCryptoPool._calculateSingleWithdrawAmount(address,uint256,uint256,uint256,uint256,bool,bool).p (contracts/pool/crypto/SyncSwapCryptoPool.sol#977) is a local variable never initialized	Medium
SyncSwapCryptoPool._calculateSingleWithdrawAmount(address,uint256,uint256,uint256,uint256,bool,bool).it (contracts/pool/crypto/SyncSwapCryptoPool.sol#939) is a local variable never initialized	Medium
SyncSwapCryptoPool.burn(bytes,address,address,bytes).params (contracts/pool/crypto/SyncSwapCryptoPool.sol#370) is a local variable never initialized	Medium
SyncSwapCryptoPool.swap(bytes,address,address,bytes).price (contracts/pool/crypto/SyncSwapCryptoPool.sol#578) is a local variable never initialized	Medium
SyncSwapCryptoPool.mint(bytes,address,address,bytes).params (contracts/pool/crypto/SyncSwapCryptoPool.sol#228) is a local variable never initialized	Medium
SyncSwapCryptoPool._tweakPrice(uint256,uint256,uint256,uint256,uint256,uint256).it (contracts/pool/crypto/SyncSwapCryptoPool.sol#1119) is a local variable never initialized	Medium
SyncSwapCryptoPool.mint(bytes,address,address,bytes).it (contracts/pool/crypto/SyncSwapCryptoPool.sol#238) is a local variable never initialized	Medium
SyncSwapCryptoPool.calcTokenAmount(uint256[2]).it (contracts/pool/crypto/SyncSwapCryptoPool.sol#1024) is a local variable never initialized	Medium
SyncSwapCryptoPool._transferTokens(address,address,uint256,uint8) (contracts/pool/crypto/SyncSwapCryptoPool.sol#711-723) ignores return value by IVault(vault).deposit(token,to) (contracts/pool/crypto/SyncSwapCryptoPool.sol#714)	Medium

Finding	Impact
SyncSwapCryptoPool.setInitialPrice(uint256) (contracts/pool/crypto/SyncSwapCryptoPool.sol#1352-1357) should emit an event for: - priceScale = _initialPrice (contracts/pool/crypto/SyncSwapCryptoPool.sol#1353) - _priceOracle = _initialPrice (contracts/pool/crypto/SyncSwapCryptoPool.sol#1354) - lastPrices = _initialPrice (contracts/pool/crypto/SyncSwapCryptoPool.sol#1355)	Low
SyncSwapCryptoPool.constructor()._master (contracts/pool/crypto/SyncSwapCryptoPool.sol#141) lacks a zero-check on : - master = _master (contracts/pool/crypto/SyncSwapCryptoPool.sol#145) - wETH = IPoolMaster(_master).wETH() (contracts/pool/crypto/SyncSwapCryptoPool.sol#146) - vault = IPoolMaster(_master).vault() (contracts/pool/crypto/SyncSwapCryptoPool.sol#147)	Low
SyncSwapCryptoPool.constructor()._token0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#138) lacks a zero-check on : - (token0,token1,token0PrecisionMultiplier,token1PrecisionMultiplier) = (_token0,_token1,_token0PrecisionMultiplier,_token1PrecisionMultiplier) (contracts/pool/crypto/SyncSwapCryptoPool.sol#148-150)	Low
SyncSwapCryptoPool.constructor()._token1 (contracts/pool/crypto/SyncSwapCryptoPool.sol#138) lacks a zero-check on : - (token0,token1,token0PrecisionMultiplier,token1PrecisionMultiplier) = (_token0,_token1,_token0PrecisionMultiplier,_token1PrecisionMultiplier) (contracts/pool/crypto/SyncSwapCryptoPool.sol#148-150)	Low
SyncSwapCryptoPool.calcTokenAmount(uint256[2]) (contracts/pool/crypto/SyncSwapCryptoPool.sol#1022-1050) uses timestamp for comparisons Dangerous comparisons: - futureParamsTime > 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#1033)	Low

Finding	Impact
SyncSwapCryptoPool._transferTokens(address,address,uint256,uint8) (contracts/pool/crypto/SyncSwapCryptoPool.sol#711-723) uses timestamp for comparisons Dangerous comparisons: - withdrawMode == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#712) - withdrawMode == 1 && token == wETH (contracts/pool/crypto/SyncSwapCryptoPool.sol#716)	Low
SyncSwapCryptoPool._params() (contracts/pool/crypto/SyncSwapCryptoPool.sol#889-907) uses timestamp for comparisons Dangerous comparisons: - _futureParamsTime > block.timestamp (contracts/pool/crypto/SyncSwapCryptoPool.sol#896)	Low
SyncSwapCryptoPool._tweakPrice(uint256,uint256,uint256,uint256,uint256,uint256) (contracts/pool/crypto/SyncSwapCryptoPool.sol#1111-1274) uses timestamp for comparisons Dangerous comparisons: - it.lastPricesTimestamp < block.timestamp (contracts/pool/crypto/SyncSwapCryptoPool.sol#1134) - newInvariant == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#1160) - price != 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#1166) - it.oldVirtualPrice != 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#1181) - futureParamsTime < block.timestamp (contracts/pool/crypto/SyncSwapCryptoPool.sol#1191) - it.virtualPrice < it.oldVirtualPrice (contracts/pool/crypto/SyncSwapCryptoPool.sol#1192) - Math.mulUnsafeFirst(2,it.virtualPrice) - 1e18 > it.xcpProfit + unsafe_mul(_rebalancingParams.allowedExtraProfit,2) (contracts/pool/crypto/SyncSwapCryptoPool.sol#1204) - norm > 1e18 (contracts/pool/crypto/SyncSwapCryptoPool.sol#1210) - norm > _adjustmentStep (contracts/pool/crypto/SyncSwapCryptoPool.sol#1222) - it.oldVirtualPrice > 1e18 (contracts/pool/crypto/SyncSwapCryptoPool.sol#1256) - Math.mulUnsafeFirst(2,it.oldVirtualPrice) - 1e18 > it.xcpProfit (contracts/pool/crypto/SyncSwapCryptoPool.sol#1257)	Low

Finding	Impact
SyncSwapCryptoPool._updateReserves(uint256,uint256) (contracts/pool/crypto/SyncSwapCryptoPool.sol#200-209) uses timestamp for comparisons Dangerous comparisons: - _balance0 > type()(uint128).max (contracts/pool/crypto/SyncSwapCryptoPool.sol#201) - _balance1 > type()(uint128).max (contracts/pool/crypto/SyncSwapCryptoPool.sol#204)	Low
SyncSwapCryptoPool._getAmountOut(address,uint256,uint256,uint256,uint256,uint256,bool) (contracts/pool/crypto/SyncSwapCryptoPool.sol#625-709) uses timestamp for comparisons Dangerous comparisons: - futureParamsTime > block.timestamp (contracts/pool/crypto/SyncSwapCryptoPool.sol#640)	Low
SyncSwapCryptoPool.priceOracle() (contracts/pool/crypto/SyncSwapCryptoPool.sol#1286-1306) uses timestamp for comparisons Dangerous comparisons: - _lastPricesTimestamp < block.timestamp (contracts/pool/crypto/SyncSwapCryptoPool.sol#1291)	Low
SyncSwapCryptoPool._mintProtocolFee(uint256,uint256) (contracts/pool/crypto/SyncSwapCryptoPool.sol#787-856) uses timestamp for comparisons Dangerous comparisons: - _totalSupply <= 1e18 (contracts/pool/crypto/SyncSwapCryptoPool.sol#792) - _xcpProfit <= _xcpProfitLast (contracts/pool/crypto/SyncSwapCryptoPool.sol#798) - _xcpProfit > _xcpProfitLast (contracts/pool/crypto/SyncSwapCryptoPool.sol#810) - liquidity != 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#823) - newVirtualPrice < 1e18 (contracts/pool/crypto/SyncSwapCryptoPool.sol#828)	Low

Finding	Impact
SyncSwapCryptoPool._calculateSingleWithdrawAmount(address,uint256,uint256,uint256,uint256,bool,bool) (contracts/pool/crypto/SyncSwapCryptoPool.sol#930-1013) uses timestamp for comparisons Dangerous comparisons: - futureParamsTime > block.timestamp (contracts/pool/crypto/SyncSwapCryptoPool.sol#946) - it.amountOut > 1e5 (contracts/pool/crypto/SyncSwapCryptoPool.sol#979) - liquidity > 1e5 (contracts/pool/crypto/SyncSwapCryptoPool.sol#980)	Low
SyncSwapCryptoPool._getMintFee(address,bool,uint256,uint256,uint256,uint256) (contracts/pool/crypto/SyncSwapCryptoPool.sol#1064-1095) uses timestamp for comparisons Dangerous comparisons: - amount0 > avg (contracts/pool/crypto/SyncSwapCryptoPool.sol#1083) - amount1 > avg (contracts/pool/crypto/SyncSwapCryptoPool.sol#1088)	Low
SyncSwapCryptoPool.burnSingle(bytes,address,address,bytes) (contracts/pool/crypto/SyncSwapCryptoPool.sol#435-522) uses timestamp for comparisons Dangerous comparisons: - require(bool)(params.liquidity != 0) (contracts/pool/crypto/SyncSwapCryptoPool.sol#447) - (params.amountOut,_price,_newInvariant,_xp0,_xp1,params.swapFee) = _calculateSingleWithdrawAmount(_sender,params.totalSupply,a,gamma,params.liquidity,params.tokenOut == token0,true) (contracts/pool/crypto/SyncSwapCryptoPool.sol#458-466) - params.tokenOut == token1 (contracts/pool/crypto/SyncSwapCryptoPool.sol#471) - require(bool)(params.tokenOut == token0) (contracts/pool/crypto/SyncSwapCryptoPool.sol#478) - params.tokenOut == token1 (contracts/pool/crypto/SyncSwapCryptoPool.sol#502)	Low
SyncSwapCryptoPool.rampParams(uint256,uint256,uint256) (contracts/pool/crypto/SyncSwapCryptoPool.sol#1313-1325) uses timestamp for comparisons Dangerous comparisons: - require(bool)(_futureTime >= block.timestamp) (contracts/pool/crypto/SyncSwapCryptoPool.sol#1316)	Low

Finding	Impact
SyncSwapCryptoPool.mint(bytes,address,address,bytes) (contracts/pool/crypto/SyncSwapCryptoPool.sol#222-360) uses timestamp for comparisons Dangerous comparisons: - require(bool)(params.amount0 != 0 || params.amount1 != 0) (contracts/pool/crypto/SyncSwapCryptoPool.sol#236) - futureParamsTime > block.timestamp (contracts/pool/crypto/SyncSwapCryptoPool.sol#253) - params.totalSupply != 0 && params.oldInvariant != 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#262) - params.liquidity == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#268) - params.totalSupply != 0 && params.oldInvariant != 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#277) - params.swapFee = _getMintFee(_sender,params.amount1 < _amount1Optimal,it.xp0 - it.oldXp0,it.xp1 - it.oldXp1,it.xp0,it.xp1) (contracts/pool/crypto/SyncSwapCryptoPool.sol#278-285) - params.liquidity > 1e5 (contracts/pool/crypto/SyncSwapCryptoPool.sol#297) - params.amount0 == 0 || params.amount1 == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#298) - params.amount0 == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#299) - newPriceScale > Math.mulUnsafeFirst(2,_priceScale) || newPriceScale < Math.divUnsafeLast(_priceScale,2) (contracts/pool/crypto/SyncSwapCryptoPool.sol#331) - params.reserve1 != 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#345) - params.amount1 >= _amount1Optimal (contracts/pool/crypto/SyncSwapCryptoPool.sol#346) - params.reserve0 == 0 (contracts/pool/crypto/SyncSwapCryptoPool.sol#274)	Low
End of table for SyncSwapCryptoPool.sol	

Results summary:

The findings obtained as a result of the Slither scan were reviewed. The majority of Slither findings were determined false-positives.

THANK YOU FOR CHOOSING

// HALBORN