HALBORN

Jarivs - Synthereum

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 9th, 2021 - December 27th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/23/2021	Juned Ansari
0.2	Document Updates	12/27/2021	Juned Ansari
0.3	Draft Review	12/28/2021	Gabi Urrutia
1.0	Remediation Plan	01/04/2022	Juned Ansari
1.1	Remediation Plan Review	01/10/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Juned Ansari	Halborn	Juned.Ansari@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Jarivs engaged Halborn to conduct a security assessment on their Synthereum smart contracts beginning on December 9th, 2021 and ending December 27th, 2021. This security assessment was scoped to the Synthereum smart contracts code in Solidity.

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

1.2 AUDIT SUMMARY

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

The purpose of this audit to achieve the following:

- Ensure that all Synthereum Contract functions are intended.
- Identify potential security issues with the assets in scope.

In summary, Halborn identified several security risks that were mostly acknowledged and addressed by the Jarvis 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 Synthereum contract 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.
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Remix IDE)

RISK METHODOLOGY:

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

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.

2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

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

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

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

IN-SCOPE : Synthereum Contracts
The security assessment was scoped to the following smart contract:
Commit-Id : 942da0885098d7df34da628435663a647a1609ba
OUT-OF-SCOPE : External libraries, self-minting/v2, test and economics attacks
Fixed-Commit-Id : 8df1abdfc165ca4e0e3f86ca5160194ff40e3167

Listing 1: Synthereum-Contract

```
core/Constants.sol
core/FactoryVersioning.sol
core/Deployer.sol
core/registries/SelfMintingRegistry.sol
core/registries/PoolRegistry.sol
core/registries/Registry.sol
core/registries/interfaces/IRegistry.sol
core/Finder.sol
core/IdentifierWhitelist.sol
core/interfaces/ICollateralWhitelist.sol
core/interfaces/IManager.sol
core/interfaces/IFactoryVersioning.sol
core/interfaces/IIdentifierWhitelist.sol
core/interfaces/IFinder.sol
core/interfaces/IDeploymentSignature.sol
core/interfaces/IDeployer.sol
core/CollateralWhitelist.sol
core/Manager.sol
oracle/common/interfaces/IPriceFeed.sol
oracle/chainlink/ChainlinkPriceFeed.sol
oracle/chainlink/interfaces/IChainlinkPriceFeed.sol
synthereum-pool/v5/LiquidityPool.sol
synthereum-pool/v5/LiquidityPoolLib.sol
synthereum-pool/v5/interfaces/ILiquidityPoolInteraction.sol
synthereum-pool/v5/interfaces/ILiquidityPoolGeneral.sol
synthereum-pool/v5/interfaces/ILiquidityPoolStorage.sol
synthereum-pool/v5/interfaces/ILiquidityPool.sol
synthereum-pool/v5/LiquidityPoolCreator.sol
synthereum-pool/v5/LiquidityPoolFactory.sol
common/FactoryConditions.sol
common/ERC2771Context.sol
common/interfaces/ITypology.sol
```

```
33 common/interfaces/IDeployment.sol
34 common/interfaces/IEmergencyShutdown.sol
35 base/utils/EnumerableBytesSet.sol
36 base/utils/StringUtils.sol
37 base/interfaces/IStandardERC20.sol
38 tokens/MintableBurnableSyntheticTokenPermit.sol
39 tokens/MintableBurnableSyntheticToken.sol
40 tokens/factories/SyntheticTokenFactory.sol
41 tokens/factories/MintableBurnableTokenFactory.sol
42 tokens/factories/SyntheticTokenPermitFactory.sol
43 tokens/factories/interfaces/IMintableBurnableTokenFactory.sol
44 tokens/MintableBurnableERC20.sol
45 tokens/interfaces/IMintableBurnableERC20.sol
46 tokens/interfaces/BaseControlledMintableBurnableERC20.sol
```

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	10	3

LIKELIHOOD

IMPACT

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

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
POSSIBLE DOS	Low	SOLVED - 12/30/2021
RE-ENTRANCY PROTECTION	Low	ACKNOWLEDGED
INCORRECT POOL FACTORY MODIFIER	Low	NOT APPLICABLE
IMPROPER ERC2771 CONTEXT IMPORT	Low	SOLVED - 12/30/2021
IGNORE RETURN VALUES	Low	ACKNOWLEDGED
INCOMPATIBILITY WITH INFLATIONARY TOKENS	Low	ACKNOWLEDGED
USAGE OF BLOCK-TIMESTAMP	Low	ACKNOWLEDGED
DIVIDE BEFORE MULTIPLY	Low	ACKNOWLEDGED
MULTIPLE PRAGMA DEFINITION	Low	ACKNOWLEDGED
FLOATING PRAGMA	Low	ACKNOWLEDGED
USE OF INLINE ASSEMBLY	Informational	ACKNOWLEDGED
REDUNDANT BOOLEAN COMPARISON	Informational	SOLVED - 12/30/2021
POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) POSSIBLE DOS - LOW

Description:

It was observed that, in MintableBurnableTokenFactory.sol contract modifier onlyPoolFactory discovered there is a for loop on variable i that missing a for loop condition on i, also noticed counter and numberOfFactories are a type of uint256 where i is of type uint8. It should be noted that condition counter < numberOfFactories may result in always true if the try condition fails, and this may let i exceed uint8 range that may result in i having a garbage value and loops will never end since there is no condition on i.

Code Location:

Listing 2: MintableBurnableTokenFactory.sol (Lines 41,42)

```
30  modifier onlyPoolFactory() {
31    ISynthereumFactoryVersioning factoryVersioning =
32      ISynthereumFactoryVersioning(
33        synthereumFinder.getImplementationAddress(
34          SynthereumInterfaces.FactoryVersioning
35        )
36      );
37    uint256 numberOfFactories =
38      factoryVersioning.numberOfVerisonsOfFactory(
39        FactoryInterfaces.PoolFactory
40      );
41    uint256 counter = 0;
42    for (uint8 i = 0; counter < numberOfFactories; i++) {
43      try
44        factoryVersioning.getFactoryVersion(FactoryInterfaces.
             PoolFactory, i)
45      returns (address factory) {
46        if (msg.sender == factory) {
47          _;
48          break;
49        } else {
50          counter++;
51        }
52      } catch {}
```

```
53      }
54      if (numberOfFactories == counter) {
55        revert('Sender must be a Pool factory');
56      }
57    }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to set the max length for `i` to which a for loop can iterate, and `i` should be of type `uint256`.

Remediation Plan:

SOLVED: The Jarvis team fixed the above issue in the commit **8df1abdfc165ca4e0e3f86ca5160194ff40e3167**. As a result, the team modify `numberOfFactories` to `uint8` and the also return type of `numberOfVersionsOfFactory` to `uint8`. Furthermore, team claims that DOS is not possible since if `numberOfFactories` is `X`, that means in loop code have `1` entry to `IF` and `X-1` in `else` and remaining are `catch`, or `X` entries in `else` and remaining in `catch`. This will produce an `exit` from the loop when all the factories address are read or when code enters in the `IF`.

3.2 (HAL-02) MISSING RE-ENTRANCY PROTECTION - LOW

Description:

It was identified that some in-scope contracts of Jarvis Synthereum branch are missing nonReentrant guard. In these function, read/write of persistent state and external calls following an external call is identified, making it vulnerable to a Reentrancy attack.

- LiquidityPoolCreator.sol contract function createPool missing non-Reentrant guard.
- LiquidityPoolLib.sol contract function liquidate and function settleEmergencyShutdown missing nonReentrant guard.

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

Code Location:

Listing 3: LiquidityPoolCreator.sol (Lines 100,116,87-92,95,96,97-99)

```
69   function createPool(Params calldata params)
70     public
71     virtual
72     returns (SynthereumLiquidityPool pool)
73   {
74     require(bytes(params.syntheticName).length != 0, 'Missing
         synthetic name');
75     require(
76       bytes(params.syntheticSymbol).length != 0,
77       'Missing synthetic symbol'
78     );
79
```

```
80    if (params.syntheticToken == address(0)) {
81      IMintableBurnableTokenFactory tokenFactory =
82        IMintableBurnableTokenFactory(
83          ISynthereumFinder(synthereumFinder).
                getImplementationAddress(
84            SynthereumInterfaces.TokenFactory
85          )
86        );
87      BaseControlledMintableBurnableERC20 tokenCurrency =
88        tokenFactory.createToken(
89          params.syntheticName,
90          params.syntheticSymbol,
91          18
92        );
93      pool = new SynthereumLiquidityPool(_convertParams(params,
            tokenCurrency));
94      // Give permissions to new pool contract and then hand over
            ownership.
95      tokenCurrency.addMinter(address(pool));
96      tokenCurrency.addBurner(address(pool));
97      tokenCurrency.addAdmin(
98        synthereumFinder.getImplementationAddress(
              SynthereumInterfaces.Manager)
99      );
100     tokenCurrency.renounceAdmin();
101   } else {
102     BaseControlledMintableBurnableERC20 tokenCurrency =
103       BaseControlledMintableBurnableERC20(params.syntheticToken)
              ;
104     require(
105       keccak256(abi.encodePacked(tokenCurrency.name())) ==
106         keccak256(abi.encodePacked(params.syntheticName)),
107       'Wrong synthetic token name'
108     );
109     require(
110       keccak256(abi.encodePacked(tokenCurrency.symbol())) ==
111         keccak256(abi.encodePacked(params.syntheticSymbol)),
112       'Wrong synthetic token symbol'
113     );
114     pool = new SynthereumLiquidityPool(_convertParams(params,
            tokenCurrency));
115   }
116   emit CreatedPool(address(pool), params.version, msg.sender);
117   return pool;
```

```
118   }
```

Listing 4: LiquidityPoolLib.sol (Lines 802,805,807-814)

```
692   function liquidate(
693     ISynthereumLiquidityPoolStorage.Storage storage self,
694     ISynthereumLiquidityPoolStorage.LPPosition storage lpPosition,
695     ISynthereumLiquidityPoolStorage.Liquidation storage
          liquidationData,
696     ISynthereumLiquidityPoolStorage.FeeStatus storage feeStatus,
697     FixedPoint.Unsigned calldata numSynthTokens,
698     address sender
699   )
700     external
701     returns (
702       uint256 synthTokensLiquidated,
703       uint256 collateralReceived,
704       uint256 rewardAmount
705     )
706   {
707     // Memory struct for saving local varibales
708     ExecuteLiquidation memory executeLiquidation;
709
710     executeLiquidation.totalCollateralAmount = lpPosition.
          totalCollateralAmount;
711
712     executeLiquidation.priceRate = getPriceFeedRate(
713       self.finder,
714       self.priceIdentifier
715     );
716
717     uint8 collateralDecimals = getCollateralDecimals(self.
          collateralToken);
718
719     // Collateral value of the synthetic token passed
720     {
721       (bool _isOverCollaterlized, ) =
722         lpPosition.isOverCollateralized(
723           liquidationData,
724           executeLiquidation.priceRate,
725           collateralDecimals,
726           executeLiquidation.totalCollateralAmount
727         );
728
```

```
729       // Revert if position is not undercollataralized
730       require(!_isOverCollaterlized, 'Position is
              overcollateralized');
731     }
732
733     IStandardERC20 _collateralToken = self.collateralToken;
734
735     executeLiquidation.tokensCollateralized = lpPosition.
            tokensCollateralized;
736
737     executeLiquidation.tokensInLiquidation = FixedPoint.min(
738       numSynthTokens,
739       executeLiquidation.tokensCollateralized
740     );
741
742     executeLiquidation.expectedCollateral =
            calculateCollateralAmount(
743       executeLiquidation.priceRate,
744       collateralDecimals,
745       executeLiquidation.tokensInLiquidation
746     );
747
748     executeLiquidation.userCollateralization = executeLiquidation
749       .tokensInLiquidation
750       .div(executeLiquidation.tokensCollateralized)
751       .mul(executeLiquidation.totalCollateralAmount);
752
753     executeLiquidation.settledCollateral;
754     executeLiquidation.rewardAmount;
755
756     if (
757       executeLiquidation.userCollateralization.isGreaterThan(
758         executeLiquidation.expectedCollateral
759       )
760     ) {
761       executeLiquidation.settledCollateral = executeLiquidation
762         .expectedCollateral;
763       executeLiquidation.rewardAmount = executeLiquidation
764         .userCollateralization
765         .sub(executeLiquidation.expectedCollateral)
766         .mul(liquidationData.liquidationReward);
767     } else {
768       executeLiquidation.unusedCollateral = self.
              calculateUnusedCollateral(
```

```
769         executeLiquidation.totalCollateralAmount,
770         feeStatus.totalFeeAmount,
771         FixedPoint.Unsigned(0)
772       );
773       executeLiquidation.settledCollateral = FixedPoint.min(
774         executeLiquidation.expectedCollateral,
775         executeLiquidation.totalCollateralAmount.add(
776           executeLiquidation.unusedCollateral
777         )
778       );
779     }
780
781     // Update Lp position
782     lpPosition.totalCollateralAmount = executeLiquidation
783       .totalCollateralAmount
784       .isGreaterThan(executeLiquidation.expectedCollateral)
785       ? executeLiquidation
786         .totalCollateralAmount
787         .sub(executeLiquidation.expectedCollateral)
788         .sub(executeLiquidation.rewardAmount)
789       : FixedPoint.Unsigned(0);
790
791     lpPosition.tokensCollateralized = executeLiquidation
792       .tokensCollateralized
793       .sub(executeLiquidation.tokensInLiquidation);
794
795     collateralReceived = executeLiquidation.settledCollateral.
          rawValue;
796
797     rewardAmount = executeLiquidation.rewardAmount.rawValue;
798
799     synthTokensLiquidated = executeLiquidation.tokensInLiquidation
          .rawValue;
800
801     // Burn synthetic tokens to be liquidated
802     self.burnSyntheticTokens(synthTokensLiquidated, sender);
803
804     // Transfer liquidated collateral and reward to the user
805     _collateralToken.safeTransfer(sender, collateralReceived +
          rewardAmount);
806
807     emit Liquidate(
808       sender,
809       synthTokensLiquidated,
```

```
810         executeLiquidation.priceRate.rawValue,
811         executeLiquidation.expectedCollateral.rawValue,
812         collateralReceived,
813         rewardAmount
814       );
815     }
```

Listing 5: LiquidityPoolLib.sol (Lines 909-913,984,987,989-994)

```
882   function settleEmergencyShutdown(
883     ISynthereumLiquidityPoolStorage.Storage storage self,
884     ISynthereumLiquidityPoolStorage.LPPosition storage lpPosition,
885     ISynthereumLiquidityPoolStorage.FeeStatus storage feeStatus,
886     ISynthereumLiquidityPoolStorage.Shutdown storage
          emergencyShutdownData,
887     bool isLiquidityProvider,
888     address sender
889   ) external returns (uint256 synthTokensSettled, uint256
        collateralSettled) {
890     // Memory struct for saving local varibales
891     ExecuteSettlement memory executeSettlement;
892
893     IMintableBurnableERC20 syntheticToken = self.syntheticToken;
894
895     executeSettlement.emergencyPrice = emergencyShutdownData.price
          ;
896
897     executeSettlement.userNumTokens = FixedPoint.Unsigned(
898       syntheticToken.balanceOf(sender)
899     );
900
901     require(
902       executeSettlement.userNumTokens.rawValue > 0 ||
            isLiquidityProvider,
903       'Sender has nothing to settle'
904     );
905
906     if (executeSettlement.userNumTokens.rawValue > 0) {
907       // Move synthetic tokens from the user to the pool
908       // - This is because derivative expects the tokens to come
            from the sponsor address
909       syntheticToken.safeTransferFrom(
910         sender,
911         address(this),
```

```
912             executeSettlement.userNumTokens.rawValue
913           );
914         }
915
916         executeSettlement.totalCollateralAmount = lpPosition.
                totalCollateralAmount;
917         executeSettlement.tokensCollaterlized = lpPosition.
                tokensCollateralized;
918         executeSettlement.totalFeeAmount = feeStatus.totalFeeAmount;
919         executeSettlement.overCollateral;
920
921         IStandardERC20 _collateralToken = self.collateralToken;
922
923         uint8 collateralDecimals = getCollateralDecimals(
                _collateralToken);
924
925         // Add overcollateral and deposited synthetic tokens if the
                sender is the LP
926         if (isLiquidityProvider) {
927           FixedPoint.Unsigned memory totalRedeemableCollateral =
928             calculateCollateralAmount(
929               executeSettlement.emergencyPrice,
930               collateralDecimals,
931               executeSettlement.tokensCollaterlized
932             );
933
934           executeSettlement.overCollateral = executeSettlement
935             .totalCollateralAmount
936             .isGreaterThan(totalRedeemableCollateral)
937             ? executeSettlement.totalCollateralAmount.sub(
                    totalRedeemableCollateral)
938             : FixedPoint.Unsigned(0);
939
940           executeSettlement.userNumTokens = FixedPoint.Unsigned(
941             syntheticToken.balanceOf(address(this))
942           );
943         }
944
945         // Calculate expected and settled collateral
946         executeSettlement.redeemableCollateral =
                calculateCollateralAmount(
947           executeSettlement
948             .emergencyPrice,
949           collateralDecimals,
```

```
950         executeSettlement
951           .userNumTokens
952       )
953         .add(executeSettlement.overCollateral);
954
955       executeSettlement.unusedCollateral = self.
              calculateUnusedCollateral(
956         executeSettlement.totalCollateralAmount,
957         executeSettlement.totalFeeAmount,
958         FixedPoint.Unsigned(0)
959       );
960
961       executeSettlement.transferableCollateral = FixedPoint.min(
962         executeSettlement.redeemableCollateral,
963         executeSettlement.totalCollateralAmount
964       );
965
966       // Update Lp position
967       lpPosition.totalCollateralAmount = executeSettlement
968         .totalCollateralAmount
969         .isGreaterThan(executeSettlement.redeemableCollateral)
970         ? executeSettlement.totalCollateralAmount.sub(
971           executeSettlement.redeemableCollateral
972         )
973         : FixedPoint.Unsigned(0);
974
975       lpPosition.tokensCollateralized = executeSettlement.
              tokensCollaterlized.sub(
976         executeSettlement.userNumTokens
977       );
978
979       synthTokensSettled = executeSettlement.userNumTokens.rawValue;
980
981       collateralSettled = executeSettlement.transferableCollateral.
              rawValue;
982
983       // Burn synthetic tokens
984       syntheticToken.burn(synthTokensSettled);
985
986       // Transfer settled collateral to the user
987       _collateralToken.safeTransfer(sender, collateralSettled);
988
989       emit Settle(
990         sender,
```

```
991         synthTokensSettled,
992         executeSettlement.redeemableCollateral.rawValue,
993         collateralSettled
994       );
995     }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Change the code to follow the checks-effects-interactions pattern and use ReentrancyGuard through the nonReentrant modifier.

Remediation Plan:

ACKNOWLEDGED: The Jarvis team acknowledged this issue, claiming that they don't want ReentrancyGuard in CreatePool of LiquidityPoolCreator , otherwise code may have a conflict with the CreatePool of LiquidityPoolFactory (that use ReentrancyGuard, and it is a derived contract LiquidityPoolCreator). Furthermore, the team claims that all the functions of the LiquidityPoolLib don't need nonReentrant modifier, since it's already in the functions of LiquidityPool that call the ones in the library. And, SettleEmergencyShutdown has the modifier in the pool.

3.3 (HAL-03) INCORRECT POOL FACTORY MODIFIER - LOW

Description:

During the manual test, It was observed that, in MintableBurnableTokenFactory .sol contract modifier onlyPoolFactory does not always execute _; or revert and in this case, the execution of the function will return the default value, which can be misleading for the caller.

Code Location:

Listing 6: MintableBurnableTokenFactory.sol (Lines 46,54)

```
30   modifier onlyPoolFactory() {
31     ISynthereumFactoryVersioning factoryVersioning =
32       ISynthereumFactoryVersioning(
33         synthereumFinder.getImplementationAddress(
34           SynthereumInterfaces.FactoryVersioning
35         )
36       );
37     uint256 numberOfFactories =
38       factoryVersioning.numberOfVerisonsOfFactory(
39         FactoryInterfaces.PoolFactory
40       );
41     uint256 counter = 0;
42     for (uint8 i = 0; counter < numberOfFactories; i++) {
43       try
44         factoryVersioning.getFactoryVersion(FactoryInterfaces.
              PoolFactory, i)
45       returns (address factory) {
46         if (msg.sender == factory) {
47           _;
48           break;
49         } else {
50           counter++;
51         }
52       } catch {}
53     }
54     if (numberOfFactories == counter) {
```

```
55        revert('Sender must be a Pool factory');
56      }
57    }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to add a `revert` or `_;` condition when `if` fails, It should be noted that all the paths in a modifier must execute either `_;` or `revert`.

Remediation Plan:

NOT APPLICABLE: The Jarvis team claims that the modifier is correct, since it must not always execute `_;`. Furthermore, team added that the modifier can lead to two results, the first enter in the `if` execute `_;`, or the second that is `exit` from the loop with counter equal to number of factories and so `revert` (implies that the sender is not a pool factory of synthereum).

3.4 (HAL-04) IMPROPER ERC2771 CONTEXT IMPORT - LOW

Description:

It was observed that the contract LiquidityPool.sol does not import ERC2771Context correctly as the contract path(i.e. ../../common//ERC2771Context.sol) comments out the ERC2771Context.sol.

Code Location:

Listing 7: LiquidityPool.sol (Lines 25)

```
24 } from '@openzeppelin/contracts/security/ReentrancyGuard.sol';
25 import {ERC2771Context} from '../../common//ERC2771Context.sol';
26 import {
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

It is recommended to double-check the imported contract paths. Replace \\ to \ for the above case.

Remediation Plan:

SOLVED: The Jarvis team fixed this issue in the commit **8df1abdfc165ca4e0e3f86ca5160194ff40e3167**. As a result, now, code follows symmetry and readability.

3.5 (HAL-05) IGNORE RETURN VALUES - LOW

Description:

The return value of an external call is not stored in a local or state variable. In contracts FactoryVersioning.sol, Registry.sol, and LiquidityPoolLib.sol there is an instance where external method is being called, and return value is being ignored.

Code Location:

Listing 8: FactoryVersioning.sol (Lines 116)

```
108   function removeFactory(bytes32 factoryType, uint8 version)
109     external
110     override
111     onlyMaintainer
112   {
113     EnumerableMap.UintToAddressMap storage selectedFactories =
114       factories[factoryType];
115     address factoryToRemove = selectedFactories.get(version);
116     selectedFactories.remove(version);
117     emit RemoveFactory(factoryType, version, factoryToRemove);
118   }
```

Listing 9: Registry.sol (Lines 77-79,80-82)

```
 66   function register(
 67     string calldata syntheticTokenSymbol,
 68     IERC20 collateralToken,
 69     uint8 version,
 70     address element
 71   ) external override nonReentrant {
 72     address deployer =
 73       ISynthereumFinder(synthereumFinder).getImplementationAddress
           (
 74         SynthereumInterfaces.Deployer
 75       );
```

```
76      require(msg.sender == deployer, 'Sender must be Synthereum
            deployer');
77      symbolToElements[syntheticTokenSymbol][collateralToken][
            version].add(
78        element
79      );
80      syntheticTokens.add(syntheticTokenSymbol.stringToBytes32());
81      collaterals.add(address(collateralToken));
82      versions.add(version);
83    }
```

Listing 10: LiquidityPoolLib.sol (Lines 637-642)

```
598   function decreaseCollateral(
599     ISynthereumLiquidityPoolStorage.Storage storage self,
600     ISynthereumLiquidityPoolStorage.LPPosition storage lpPosition,
601     ISynthereumLiquidityPoolStorage.Liquidation storage
            liquidationData,
602     ISynthereumLiquidityPoolStorage.FeeStatus storage feeStatus,
603     FixedPoint.Unsigned calldata collateralToDecrease,
604     FixedPoint.Unsigned calldata collateralToWithdraw,
605     address sender
606   ) external returns (uint256 newTotalCollateral) {
607     // Check that collateral to be decreased is not 0
608     require(collateralToDecrease.rawValue > 0, 'No collateral to
            be decreased');
609
610     // Resulting total collateral amount
611     FixedPoint.Unsigned memory _newTotalCollateral =
612       lpPosition.totalCollateralAmount.sub(collateralToDecrease);
613
614     // Check that position doesn't become undercollateralized
615     (bool _isOverCollateralized, ) =
616       lpPosition.isOverCollateralized(
617         liquidationData,
618         getPriceFeedRate(self.finder, self.priceIdentifier),
619         getCollateralDecimals(self.collateralToken),
620         _newTotalCollateral
621       );
622
623     require(_isOverCollateralized, 'Position undercollateralized')
            ;
624
625     // Update new total collateral amount
```

```
626     lpPosition.totalCollateralAmount = _newTotalCollateral;
627
628     newTotalCollateral = _newTotalCollateral.rawValue;
629
630     emit DecreaseCollateral(
631       sender,
632       collateralToDecrease.rawValue,
633       newTotalCollateral
634     );
635
636     if (collateralToWithdraw.rawValue > 0) {
637       self._withdrawLiquidity(
638         lpPosition,
639         feeStatus,
640         collateralToWithdraw,
641         sender
642       );
643     }
644   }
```

Listing 11: LiquidityPoolLib.sol (Lines 502)

```
463   function exchangeMint(
464     ISynthereumLiquidityPoolStorage.Storage storage self,
465     ISynthereumLiquidityPoolStorage.LPPosition storage lpPosition,
466     ISynthereumLiquidityPoolStorage.FeeStatus storage feeStatus,
467     FixedPoint.Unsigned calldata collateralAmount,
468     FixedPoint.Unsigned calldata numTokens,
469     address recipient
470   ) external {
471     self.checkPool(ISynthereumLiquidityPoolGeneral(msg.sender));
472
473     // Sending amount must be different from 0
474     require(
475       collateralAmount.rawValue > 0,
476       'Sending collateral amount is equal to 0'
477     );
478
479     // Collateral available
480     FixedPoint.Unsigned memory unusedCollateral =
481       self.calculateUnusedCollateral(
482         lpPosition.totalCollateralAmount,
483         feeStatus.totalFeeAmount,
484         collateralAmount
```

```
485       );
486
487     // Update LP's collateralization status
488     FixedPoint.Unsigned memory overCollateral =
489       lpPosition.updateLpPositionInMint(
490         self.overCollateralization,
491         collateralAmount,
492         numTokens
493       );
494
495     //Check there is enough liquidity in the pool for
            overcollateralization
496     require(
497       unusedCollateral.isGreaterThanOrEqual(overCollateral),
498       'No enough liquidity for cover mint operation'
499     );
500
501     // Mint synthetic asset and transfer to the recipient
502     self.syntheticToken.mint(recipient, numTokens.rawValue);
503   }
```

Listing 12: LiquidityPoolLib.sol (Lines 1383-1386)

```
1339  function executeMint(
1340    ISynthereumLiquidityPoolStorage.Storage storage self,
1341    ISynthereumLiquidityPoolStorage.LPPosition storage lpPosition,
1342    ISynthereumLiquidityPoolStorage.FeeStatus storage feeStatus,
1343    ExecuteMintParams memory executeMintParams
1344  ) internal {
1345    // Sending amount must be different from 0
1346    require(
1347      executeMintParams.collateralAmount.rawValue > 0,
1348      'Sending collateral amount is equal to 0'
1349    );
1350
1351    // Collateral available
1352    FixedPoint.Unsigned memory unusedCollateral =
1353      self.calculateUnusedCollateral(
1354        lpPosition.totalCollateralAmount,
1355        feeStatus.totalFeeAmount,
1356        FixedPoint.Unsigned(0)
1357      );
1358
1359    // Update LP's collateralization status
```

```
360     FixedPoint.Unsigned memory overCollateral =
361       lpPosition.updateLpPositionInMint(
362         self.overCollateralization,
363         executeMintParams.collateralAmount,
364         executeMintParams.numTokens
365       );
366
367     //Check there is enough liquidity in the pool for
          overcollateralization
368     require(
369       unusedCollateral.isGreaterThanOrEqual(overCollateral),
370       'No enough liquidity for covering mint operation'
371     );
372
373     // Update fees status
374     feeStatus.updateFees(self.fee, executeMintParams.feeAmount);
375
376     // Pull user's collateral
377     self.pullCollateral(
378       executeMintParams.sender,
379       executeMintParams.totCollateralAmount
380     );
381
382     // Mint synthetic asset and transfer to the recipient
383     self.syntheticToken.mint(
384       executeMintParams.recipient,
385       executeMintParams.numTokens.rawValue
386     );
387
388     emit Mint(
389       executeMintParams.sender,
390       executeMintParams.totCollateralAmount.rawValue,
391       executeMintParams.numTokens.rawValue,
392       executeMintParams.feeAmount.rawValue,
393       executeMintParams.recipient
394     );
395   }
```

Risk Level:

Likelihood - 3

Impact - 2

Recommendation:

Add return value check to avoid unexpected crash of the contract. Return value check will help in handling the exceptions better way.

Remediation Plan:

ACKNOWLEDGED: The Jarvis team acknowledged this issue, claiming that the return values ignored do not produce any advantage.

3.6 (HAL-06) INCOMPATIBILITY WITH INFLATIONARY TOKENS - LOW

Description:

In multiple functions Jarvis Synthereum contracts use OpenZeppelin safeTransferFrom and safeTransfer to handle the token transfers. These functions call transferFrom and transfer internally in the token contract to actually execute the transfer. However, the actual amount transferred, i.e., the delta of previous (before transfer) and current (after transfer) balance is not verified. As a result, a malicious user may list a custom ERC20 token with the transferFrom or transfer function modified in such a way that it (e.g. does not transfer any tokens at all and the attacker is still going to have their liquidity pool tokens minted anyway). In this case, both tokens are set in the constructor by the creator of the contract, so they are trusted, but it would be still a good practice to perform this check.

Code Location:

Listing 13: LiquidityPoolLib.sol

```
#674: self.collateralToken.safeTransfer(sender, feeClaimed);
#805: _collateralToken.safeTransfer(sender, collateralReceived +
    rewardAmount);
#909: syntheticToken.safeTransferFrom(
#987: _collateralToken.safeTransfer(sender, collateralSettled);
#1437: self.collateralToken.safeTransfer(
#1502: self.collateralToken.safeTransfer(
#1554: self.collateralToken.safeTransfer(sender, _collateralAmount
    );
#1666: self.collateralToken.safeTransferFrom(
#1687: synthToken.safeTransferFrom(sender, address(this),
    numTokens);
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendations:

Whenever tokens are transferred, the delta of the previous (before transfer) and current (after transfer) token balance should be verified to match the declared token amount.

Remediation Plan:

ACKNOWLEDGED: The Jarvis team acknowledged this issue, claiming that the delta checking condition is a waste of gas, in this case, since synthetic tokens are not inflationary and the collaterals are whitelisted by Jarvis (and future DAO), so no possibility to have inflationary token.

3.7 (HAL-07) USAGE OF BLOCK-TIMESTAMP - LOW

Description:

During a manual review, usage of block.timestamp in some Jarvis Synthereum contracts were observed. The contract developers should be aware that this does not mean current time. now is an alias for block.timestamp. The value of block.timestamp can be influenced by miners to a certain degree, so the testers should be warned that this may have some risk if miners collude on time manipulation to influence the price oracles. Miners can influence the timestamp by a tolerance of 900 seconds.

Code Location:

Listing 14: LiquidityPoolLib.sol

```
1 #841: timestamp = block.timestamp;
2 #1811: require(block.timestamp <= expiration, 'Transaction expired
      ');
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

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

Remediation Plan:

ACKNOWLEDGED: The Jarvis team acknowledged this issue, claiming that the block.timestamp is not a problem, since the miner in this case can only allow a transaction expired to be executed setting a timestamp forward in time. The max forwarding time is 15 seconds, so, the worst-case scenario produce a longer life for the transaction of 15 seconds.

3.8 (HAL-08) DIVIDE BEFORE MULTIPLY - LOW

Description:

Solidity integer division might truncate. As a result, the loss of precision can sometimes be avoided by multiplying before dividing, although the manual implementation of the precision/decimal calculation is being taken care of by the developer. In the smart contracts set, there are multiple instances where division is being performed before multiplication.

Code Location:

Listing 15: LiquidityPoolLib.sol (Lines 1235-1238)

```
1235     FixedPoint.Unsigned memory collateralRedeemed =
1236       syntheticTokens.div(totalActualTokens).mul(
1237         lpPosition.totalCollateralAmount
1238       );
```

Listing 16: LiquidityPoolLib.sol (Lines 1296-1299)

```
1296     FixedPoint.Unsigned memory collateralRedeemed =
1297       syntheticTokens.div(totalActualTokens).mul(
1298         lpPosition.totalCollateralAmount
1299       );
```

Listing 17: LiquidityPoolLib.sol (Lines 1601,1602)

```
1601     FixedPoint.Unsigned memory fractionRedeemed =
1602       numTokens.div(totalActualTokens);
```

Listing 18: LiquidityPoolLib.sol (Lines 1604)

```
1604     collateralRedeemed = fractionRedeemed.mul(
           totalActualCollateral);
```

Listing 19: LiquidityPoolLib.sol (Lines 748-751)

```
748     executeLiquidation.userCollateralization = executeLiquidation
749       .tokensInLiquidation
750       .div(executeLiquidation.tokensCollateralized)
751       .mul(executeLiquidation.totalCollateralAmount);
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Consider performing multiplications before divisions to ensure precision in the results when using non floating-point data types.

Remediation Plan:

ACKNOWLEDGED: The Jarvis team acknowledged this issue, claiming that the team want to divide before multiple to calculate the redeem ratio and then apply this to the collateral, to underestimate the collateral to remove. Furthermore, team added that this doesn't change the redeeming and settle logic and profits, and has the advantage that some dust due to rounding is kept inside the position and not out the position, keeping the collateralization stronger.

3.9 (HAL-09) MULTIPLE PRAGMA DEFINITION - LOW

Description:

Jarvis Synthereum contracts use different pragma versions, i.e., ^0.8.0 and ^0.8.4. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly.
Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, either an outdated pragma version that might introduce bugs that affect the contract system negatively or a recently released pragma version which has not been extensively tested. The latest pragma version (0.8.9) was released in September 2021. Many pragma versions have been lately released, going from version 0.7.x to the recently released version 0.8.x. in just 8 months.

Reference: https://github.com/ethereum/solidity/releases

In the Solitidy Github repository, there is a json file with all bugs finding in the different compiler versions. It should be noted that pragma 0.6.12 and 0.7.6 are widely used by Solidity developers and have been extensively tested in many security audits.

Reference: https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

Consider locking and using a single pragma version without known bugs for the compiler version.

Remediation Plan:

ACKNOWLEDGED: The Jarvis team acknowledged this issue, claiming that the team uses version 0.8.4 as compiler but support all the compilers of version 0.8.x greater than 0.8.3. The 0.8.0 is used by the import from OpenZeppelin. As a result, the project can be compiled with every version greater than 0.8.3 (of 0.8.x release), and this is the wanted result by the team.

3.10 (HAL-10) FLOATING PRAGMA - LOW

Description:

Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively or recently released pragma versions may have unknown security vulnerabilities.

Code Location:

The following contracts and pragma versions where not locked:

Listing 20

```
BaseControlledMintableBurnableERC20.sol: pragma solidity ^0.8.4;
ChainlinkPriceFeed.sol: pragma solidity ^0.8.4;
CollateralWhitelist.sol: pragma solidity ^0.8.0;
Constants.sol: pragma solidity ^0.8.4;
Deployer.sol: pragma solidity ^0.8.4;
EnumerableBytesSet.sol: pragma solidity ^0.8.4;
ERC2771Context.sol: pragma solidity ^0.8.0;
FactoryConditions.sol: pragma solidity ^0.8.4;
FactoryVersioning.sol: pragma solidity ^0.8.4;
Finder.sol: pragma solidity ^0.8.4;
IChainlinkPriceFeed.sol: pragma solidity ^0.8.4;
ICollateralWhitelist.sol: pragma solidity ^0.8.0;
IdentifierWhitelist.sol: pragma solidity ^0.8.0;
IDeployer.sol: pragma solidity ^0.8.4;
IDeploymentSignature.sol: pragma solidity ^0.8.4;
IDeployment.sol: pragma solidity ^0.8.4;
IEmergencyShutdown.sol: pragma solidity ^0.8.4;
IFactoryVersioning.sol: pragma solidity ^0.8.4;
IFinder.sol: pragma solidity ^0.8.4;
IIdentifierWhitelist.sol: pragma solidity ^0.8.0;
ILiquidityPoolGeneral.sol: pragma solidity ^0.8.4;
ILiquidityPoolInteraction.sol: pragma solidity ^0.8.4;
```

```
23 ILiquidityPool.sol: pragma solidity ^0.8.4;
24 ILiquidityPoolStorage.sol: pragma solidity ^0.8.4;
25 IManager.sol: pragma solidity ^0.8.4;
26 IMintableBurnableERC20.sol: pragma solidity ^0.8.4;
27 IMintableBurnableTokenFactory.sol: pragma solidity ^0.8.4;
28 IPriceFeed.sol: pragma solidity ^0.8.4;
29 IRegistry.sol: pragma solidity ^0.8.4;
30 IStandardERC20.sol: pragma solidity ^0.8.4;
31 ITypology.sol: pragma solidity ^0.8.4;
32 LiquidityPoolCreator.sol: pragma solidity ^0.8.4;
33 LiquidityPoolFactory.sol: pragma solidity ^0.8.4;
34 LiquidityPoolLib.sol: pragma solidity ^0.8.4;
35 LiquidityPool.sol: pragma solidity ^0.8.4;
36 Manager.sol: pragma solidity ^0.8.4;
37 MintableBurnableERC20.sol: pragma solidity ^0.8.4;
38 MintableBurnableSyntheticTokenPermit.sol: pragma solidity ^0.8.4;
39 MintableBurnableSyntheticToken.sol: pragma solidity ^0.8.4;
40 MintableBurnableTokenFactory.sol: pragma solidity ^0.8.4;
41 PoolRegistry.sol: pragma solidity ^0.8.4;
42 Registry.sol: pragma solidity ^0.8.4;
43 SelfMintingRegistry.sol: pragma solidity ^0.8.4;
44 StringUtils.sol: pragma solidity ^0.8.4;
45 SyntheticTokenFactory.sol: pragma solidity ^0.8.4;
46 SyntheticTokenPermitFactory.sol: pragma solidity ^0.8.4;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

Consider locking the pragma version as it is not recommended using a floating pragma in production. It is possible to lock the pragma by fixing the version both in truffle-config.js for the Truffle framework or in hardhat.config.js for the HardHat framework.

Remediation Plan:

ACKNOWLEDGED: The Jarvis team acknowledged this issue, claiming that the team wants a floating 0.8.x greater than 0.8.3 to be composable with other projects. As a result, locking the version can produce that other protocols of version 0.8.x greater than 0.8.4 cannot import and use team contracts.

3.11 (HAL-11) USE OF INLINE ASSEMBLY - INFORMATIONAL

Description:

Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This discards several important safety features in Solidity and could incur in some risks should they used incorrectly.

Code Location:

The following contracts make use of inline assembly:

Listing 21

```
1 LiquidityPool.sol#866: assembly {
2 ERC2771Context.sol#27: assembly {
3 StringUtils.sol#24: assembly {
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

When possible, it is not recommended to use inline assembly because it allows access to the Ethereum Virtual Machine (EVM) at a low level. An attacker could bypass many important safety features of Solidity.

Remediation Plan:

ACKNOWLEDGED: The Jarvis team acknowledged this issue, claiming that the team only uses inline assembly in the part of code where it is necessary.

3.12 (HAL-12) REDUNDANT BOOLEAN COMPARISON - INFORMATIONAL

Description:

In the Solidity language, Boolean constants can be used directly and do not need to be compared to true or false. In one of the Jarvis Synthereum contract, boolean constant is compared with true.

Code Location:

Listing 22

```
1 FactoryVersioning.sol#97: if (isNewVersion == true) {
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended to compare boolean constants directly in the if statement.

Remediation Plan:

SOLVED: The Jarvis team fixed this issue in the commit **8df1abdfc165ca4e0e3f86ca5160194ff40e3167**.

3.13 (HAL-13) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.
Also, methods do not necessarily have to be public if they are only called within the contract-in such case they should be marked internal.

Code Location:

Below are smart contracts and their corresponding functions affected:
MintableBurnableERC20.sol:
addAdmin() addAdminAndMinterAndBurner() addBurner() addMinter() renounceAdmin() renounceAdminAndMinterAndBurner() renounceBurner() renounceMinter()

MintableBurnableSyntheticToken.sol:
isAdmin() isBurner() isMinter()

MintableBurnableTokenFactory.sol:
createToken()

SynthereumLiquidityPoolCreator.sol:
createPool()

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider, as much as possible, declaring external variables instead of public variables. As for best practice, you should use external if you expect that the function will only be called externally and use public if you need to call the function internally. To sum up, all can access to public functions, external functions only can be accessed externally, and internal functions can only be called within the contract.

Remediation Plan:

ACKNOWLEDGED: The Jarvis team acknowledged this issue, claiming that the team want above functions as public to not use it only as external but give the possibility in future to be used also by future derived contracts as internal ones.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

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

Results:

```
SynthereumFactoryVersioning.removeFactory(bytes32,uint8) (libs/contracts/contracts/core/FactoryVersioning.sol#108-118) ignores return value by selectedFactories.remove(version) (li
bs/contracts/contracts/core/FactoryVersioning.sol#116)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by symbolToElements[syntheticTokenSymbol
][collateralToken][version].add(element) (libs/contracts/contracts/core/registries/Registry.sol#77-79)
SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by syntheticTokens.add(syntheticTokenSym
bol.stringToBytes32()) (libs/contracts/contracts/core/registries/Registry.sol#80)
SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by collaterals.add(address(collateralTok
en)) (libs/contracts/contracts/core/registries/Registry.sol#81)
SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by versions.add(version) (libs/contracts
/contracts/core/registries/Registry.sol#82)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by symbolToElements[syntheticTokenSymbol
][collateralToken][version].add(element) (libs/contracts/contracts/core/registries/Registry.sol#77-79)
SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by syntheticTokens.add(syntheticTokenSym
bol.stringToBytes32()) (libs/contracts/contracts/core/registries/Registry.sol#80)
SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by collaterals.add(address(collateralTok
en)) (libs/contracts/contracts/core/registries/Registry.sol#81)
SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by versions.add(version) (libs/contracts
/contracts/core/registries/Registry.sol#82)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by symbolToElements[syntheticTokenSymbol
][collateralToken][version].add(element) (libs/contracts/contracts/core/registries/Registry.sol#77-79)
SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by syntheticTokens.add(syntheticTokenSym
bol.stringToBytes32()) (libs/contracts/contracts/core/registries/Registry.sol#80)
SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by collaterals.add(address(collateralTok
en)) (libs/contracts/contracts/core/registries/Registry.sol#81)
SynthereumRegistry.register(string,IERC20,uint8,address) (libs/contracts/contracts/core/registries/Registry.sol#66-83) ignores return value by versions.add(version) (libs/contracts
/contracts/core/registries/Registry.sol#82)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

SynthereumLiquidityPoolLib.liquidate(ISynthereumLiquidityPoolStorage.Storage,ISynthereumLiquidityPoolStorage.LPPosition,ISynthereumLiquidityPoolStorage.Liquidation,ISynthereumLiqui
dityPoolStorage.FeeStatus,FixedPoint.Unsigned,address) (libs/contracts/contracts/synthereum-pool/v5/LiquidityPoolLib.sol#692-815) performs a multiplication on the result of a divis
ion:
        -executeLiquidation.userCollateralization = executeLiquidation.tokensInLiquidation.div(executeLiquidation.tokensCollateralized).mul(executeLiquidation.totalCollateralAmount
) (libs/contracts/contracts/synthereum-pool/v5/LiquidityPoolLib.sol#748-751)
```

```
SyntheremLiquidityPoolLib.exchangeMint(ISyntheremLiquidityPoolStorage.Storage,ISyntheremLiquidityPoolStorage.LPPosition,ISyntheremLiquidityPoolStorage.FeeStatus,FixedPoint.Unsi
gned,FixedPoint.Unsigned,address) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#463-503) ignores return value by self.syntheticToken.mint(recipient,numTokens.ra
wValue) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#502)
SyntheremLiquidityPoolLib.decreaseCollateral(ISyntheremLiquidityPoolStorage.Storage,ISyntheremLiquidityPoolStorage.LPPosition,ISyntheremLiquidityPoolStorage.Liquidation,ISynthe
reumLiquidityPoolStorage.FeeStatus,FixedPoint.Unsigned,FixedPoint.Unsigned,address) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#598-644) ignores return value
by self._withdrawLiquidity(lpPosition,feeStatus,collateralToWithdraw,sender) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#637-642)
SyntheremLiquidityPoolLib.executeMint(ISyntheremLiquidityPoolStorage.Storage,ISyntheremLiquidityPoolStorage.LPPosition,ISyntheremLiquidityPoolStorage.FeeStatus,SyntheremLiquid
ityPoolLib.ExecuteMintParams) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1339-1395) ignores return value by self.syntheticToken.mint(executeMintParams.recipi
ent,executeMintParams.numTokens.rawValue) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1383-1386)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

SyntheremLiquidityPoolLib.liquidate(ISyntheremLiquidityPoolStorage.Storage,ISyntheremLiquidityPoolStorage.LPPosition,ISyntheremLiquidityPoolStorage.Liquidation,ISyntheremLiqui
dityPoolStorage.FeeStatus,FixedPoint.Unsigned,address) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#692-815) performs a multiplication on the result of a divis
ion:
        -executeLiquidation.userCollateralization = executeLiquidation.tokensInLiquidation.div(executeLiquidation.tokensCollateralized).mul(executeLiquidation.totalCollateralAmount
) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#748-751)
SyntheremLiquidityPoolLib.getRedeemTradeInfo(ISyntheremLiquidityPoolStorage.Storage,ISyntheremLiquidityPoolStorage.LPPosition,FixedPoint.Unsigned) (libs/contracts/contracts/synt
heremum-pool/v5/LiquidityPoolLib.sol#1211-1247) performs a multiplication on the result of a division:
        -collateralRedeemed = syntheticTokens.div(totalActualTokens).mul(lpPosition.totalCollateralAmount) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1235-12
38)
SyntheremLiquidityPoolLib.getExchangeTradeInfo(ISyntheremLiquidityPoolStorage.Storage,ISyntheremLiquidityPoolStorage.LPPosition,FixedPoint.Unsigned,ISyntheremLiquidityPoolGener
al) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1259-1326) performs a multiplication on the result of a division:
        -collateralRedeemed = syntheticTokens.div(totalActualTokens).mul(lpPosition.totalCollateralAmount) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1296-12
99)
SyntheremLiquidityPoolLib.updateLpPositionInRedeem(ISyntheremLiquidityPoolStorage.LPPosition,FixedPoint.Unsigned) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.so
l#1591-1611) performs a multiplication on the result of a division:
        -fractionRedeemed = numTokens.div(totalActualTokens) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1601-1602)
        -collateralRedeemed = fractionRedeemed.mul(totalActualCollateral) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1604)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

SyntheremLiquidityPoolLib.settleEmergencyShutdown(ISyntheremLiquidityPoolStorage.Storage,ISyntheremLiquidityPoolStorage.LPPosition,ISyntheremLiquidityPoolStorage.FeeStatus,ISyn
theremLiquidityPoolStorage.Shutdown,bool,address).executeSettlement (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#891) is a local variable never initialized
SyntheremLiquidityPoolLib.liquidate(ISyntheremLiquidityPoolStorage.Storage,ISyntheremLiquidityPoolStorage.LPPosition,ISyntheremLiquidityPoolStorage.Liquidation,ISyntheremLiqui
dityPoolStorage.FeeStatus,FixedPoint.Unsigned,address).executeLiquidation (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#708) is a local variable never initializ
ed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
SyntheremLiquidityPoolLib.ExecuteRedeemParams) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1404-1449):
        External calls:
        - self.burnSyntheticTokens(executeRedeemParams.numTokens.rawValue,executeRedeemParams.sender) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1431-1434)
        - self.collateralToken.safeTransfer(executeRedeemParams.recipient,executeRedeemParams.collateralAmount.rawValue) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolL
ib.sol#1437-1440)
        Event emitted after the call(s):
        - Redeem(executeRedeemParams.sender,executeRedeemParams.numTokens.rawValue,executeRedeemParams.collateralAmount.rawValue,executeRedeemParams.feeAmount.rawValue,executeRedee
mParams.recipient) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1442-1448)
Reentrancy in SyntheremLiquidityPoolLib.increaseCollateral(ISyntheremLiquidityPoolStorage.Storage,ISyntheremLiquidityPoolStorage.LPPosition,ISyntheremLiquidityPoolStorage.FeeSt
atus,FixedPoint.Unsigned,FixedPoint.Unsigned,address) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#540-583):
        External calls:
        - self.pullCollateral(sender,collateralToTransfer) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#553)
        Event emitted after the call(s):
        - IncreaseCollateral(sender,collateralToIncrease.rawValue,newTotalCollateral) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#578-582)
Reentrancy in SyntheremLiquidityPoolLib.liquidate(ISyntheremLiquidityPoolStorage.Storage,ISyntheremLiquidityPoolStorage.LPPosition,ISyntheremLiquidityPoolStorage.Liquidation,IS
yntheremLiquidityPoolStorage.FeeStatus,FixedPoint.Unsigned,address) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#692-815):
        External calls:
        - self.burnSyntheticTokens(synthTokensLiquidated,sender) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#802)
        - _collateralToken.safeTransfer(sender,collateralReceived + rewardAmount) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#805)
        Event emitted after the call(s):
        - Liquidate(sender,synthTokensLiquidated,executeLiquidation.priceRate.rawValue,executeLiquidation.expectedCollateral.rawValue,collateralReceived,rewardAmount) (libs/contrac
ts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#807-814)
Reentrancy in SyntheremLiquidityPoolLib.settleEmergencyShutdown(ISyntheremLiquidityPoolStorage.Storage,ISyntheremLiquidityPoolStorage.LPPosition,ISyntheremLiquidityPoolStorage.
FeeStatus,ISyntheremLiquidityPoolStorage.Shutdown,bool,address) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#882-995):
        External calls:
        - syntheticToken.safeTransferFrom(sender,address(this),executeSettlement.userNumTokens.rawValue) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#909-913)
        - syntheticToken.burn(synthTokensSettled) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#984)
        - _collateralToken.safeTransfer(sender,collateralSettled) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#987)
        Event emitted after the call(s):
        - Settle(sender,synthTokensSettled,executeSettlement.redeemableCollateral.rawValue,collateralSettled) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#989-
994)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

SyntheremLiquidityPoolLib.checkExpiration(uint256) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1810-1812) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiration,Transaction expired) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolLib.sol#1811)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
SyntheremFactoryVersioning.setFactory(bytes32,uint8,address) (libs/contracts/contracts/core/FactoryVersioning.sol#90-102) compares to a boolean constant:
        -isNewVersion == true (libs/contracts/contracts/core/FactoryVersioning.sol#97)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

SyntheremLiquidityPool (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPool.sol#35-886) does not implement functions:
        - ERC2771Context.isTrustedForwarder(address) (libs/contracts/contracts/common/ERC2771Context.sol#12-16)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

SyntheremLiquidityPool (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPool.sol#35-886) does not implement functions:
        - ERC2771Context.isTrustedForwarder(address) (libs/contracts/contracts/common/ERC2771Context.sol#12-16)
BaseControlledMintableBurnableERC20 (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#10-101) does not implement functions:
        - BaseControlledMintableBurnableERC20.addAdmin(address) (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#46)
        - BaseControlledMintableBurnableERC20.addAdminAndMinterAndBurner(address) (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#52)
        - BaseControlledMintableBurnableERC20.addBurner(address) (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#40)
        - BaseControlledMintableBurnableERC20.addMinter(address) (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#34)
        - IMintableBurnableERC20.burn(uint256) (libs/contracts/contracts/tokens/interfaces/IMintableBurnableERC20.sol#14)
        - IMintableBurnableERC20.mint(address,uint256) (libs/contracts/contracts/tokens/interfaces/IMintableBurnableERC20.sol#20)
        - BaseControlledMintableBurnableERC20.renounceAdmin() (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#71)
        - BaseControlledMintableBurnableERC20.renounceAdminAndMinterAndBurner() (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#76)
        - BaseControlledMintableBurnableERC20.renounceBurner() (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#66)
        - BaseControlledMintableBurnableERC20.renounceMinter() (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

createPool(SyntheremLiquidityPoolCreator.Params) should be declared external:
        - SyntheremLiquidityPoolCreator.createPool(SyntheremLiquidityPoolCreator.Params) (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPoolCreator.sol#69-118)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

SyntheremLiquidityPool (libs/contracts/contracts/syntheremum-pool/v5/LiquidityPool.sol#35-886) does not implement functions:
        - ERC2771Context.isTrustedForwarder(address) (libs/contracts/contracts/common/ERC2771Context.sol#12-16)
BaseControlledMintableBurnableERC20 (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#10-101) does not implement functions:
        - BaseControlledMintableBurnableERC20.addAdmin(address) (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#46)
        - BaseControlledMintableBurnableERC20.addAdminAndMinterAndBurner(address) (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#52)
        - BaseControlledMintableBurnableERC20.addBurner(address) (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#40)
        - BaseControlledMintableBurnableERC20.addMinter(address) (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#34)
        - IMintableBurnableERC20.burn(uint256) (libs/contracts/contracts/tokens/interfaces/IMintableBurnableERC20.sol#14)
        - IMintableBurnableERC20.mint(address,uint256) (libs/contracts/contracts/tokens/interfaces/IMintableBurnableERC20.sol#20)
        - BaseControlledMintableBurnableERC20.renounceAdmin() (libs/contracts/contracts/tokens/interfaces/BaseControlledMintableBurnableERC20.sol#71)
```

According to the test results, some findings found by these tools were considered as false positives, while some of these findings were real

security concerns. All relevant findings were reviewed by the auditors and relevant findings addressed in the report as security concerns.

4.2 AUTOMATED SECURITY SCAN

Description:

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

Results:

ChainlinkPriceFeed.sol

Report for libs/contracts/contracts/oracle/chainlink/ChainlinkPriceFeed.sol

Line	SWC Title	Severity	Short Description
77	(SWC-115) Authorization through tx.origin	Low	Use of "tx.origin" as a part of authorization control.

All relevant valid findings were founded in the manual code review.

THANK YOU FOR CHOOSING

// HALBORN