HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Interport
Date: March 22, 2023

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Interport
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 token; Staking; Bridge; DEX
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://interport.fi
Changelog	18.01.2023 - Initial Review 06.03.2023 - Second Review 22.03.2023 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Interport (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is review and security analysis of smart contracts in the repository:

Initial review scope

Repository	https://github.com/Interport-Finance/contracts-interport
Commit	2e6a44647233580466f672fa6ca3f88ac109f716
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts: File: ./contracts/ActionExecutor.sol SHA3: 73c15062c92778ef67156eb8cc9c511899d7f12f6831aefcedc59cde76a92f1c File: ./contracts/ActionExecutorRegistry.sol SHA3: 796206408352e68741ce16be711f524409b9d90807d4405bca37d2d5b757c8a4 File: ./contracts/Address.sol SHA3: 2242661fb9cb6ba889245fa0389d12992bb8d30836edca881b894b80a2d5b2f1 File: ./contracts/AnyCallGateway.sol SHA3: bdcb6b78ab5b70f6cf8e71a6ffa43a01e69d2f477ee197a9d9d6bcec1f53883a File: ./contracts/AssetSpenderRole.sol SHA3: fed00e3b12fd5db801635e856665886cca0481fc8523a2d12fb609ace62e7842 File: ./contracts/BalanceManagement.sol SHA3: 3fc4d6cfa156b372d36a844c34c6c1acd571de120309ab03163bc712cbf6d505 File: ./contracts/BurnerRole.sol SHA3: 8a87f3bd8cc6d6c53a6818bca8b1694c6c0f9c2588316e4205e9eff3b6c44406 File: ./contracts/CallerGuard.sol SHA3: 3428573e51b3cdb06df42636ca05cdd3654ebc65c4212b433443db0edce9fdd6 File: ./contracts/DataStructures.sol SHA3: 33bfb80ae0efff059ee7099f9b9a48f9a1ecb4b5d4b93130af7a453b0c21baec File: ./contracts/ERC20.sol SHA3: 3d6946547ae79a0220bf18d774c5f4e45197246f050d919371276b9d90114235 File: ./contracts/Errors.sol SHA3: 6375f74b74cd1b6fb84a23cfe64385eed11f6dffe7bb691c63405813b86cbb55 File: ./contracts/farm/Buyback.sol SHA3: 8a998434454982612df59dcf7293f8c1b26ee8760c19d64d6117458bb735584b	

```
File: ./contracts/farm/FeeMediator.sol
SHA3: 25148cb7d1a1bf01888604b420f732f0d2484ffadee0ff46b819925f3230d5dc

File: ./contracts/farm/ITPRevenueShare.sol
SHA3: e0b577e2f04ea8146486e5c0288cb6fb20941769dc2258f8cb6cc7e8d1fd29c1

File: ./contracts/farm/LPRevenueShare.sol
SHA3: 79622a976db4907e16eafa2f3daddf0def9faf7b7afe84c02ede2bfa088b1172

File: ./contracts/farm/StablecoinFarm.sol
SHA3: 2cf769c27fe73a8152d21ae07bb64bc8a74177048dee15113452950ee0aaeb2b

File: ./contracts/interfaces/IActionDataStructures.sol
SHA3: 6b663ac335d4a561785a196ed0da61528f36ebb9c276fc3511f9b1cd33d86751

File: ./contracts/interfaces/IAssetReceiver.sol
SHA3: 51f36079f1306cac511cb1e112aa5daa9781b590ec155b15133c8de2a9c70441

File: ./contracts/interfaces/IBuybackToken.sol
SHA3: 283f277cf0f9f36ef6c1ac3b19029706445de3e159f5755fc0a9b6ff2371a41c

File: ./contracts/interfaces/ICallExecutor.sol
SHA3: 6a8fc6fa4643d94f46bd63f1207980318f96c79bc1955d969857291df6087999

File: ./contracts/interfaces/ICallProxy.sol
SHA3: a81aab13c0bcc8f310119466cfe7253b500ef85a4c1b164ad24b68f4204d5aa7

File: ./contracts/interfaces/IERC20.sol
SHA3: 29811757d5c54423bf626fbee093b64ba7fa97e515ac00c02d94658d535697f5

File: ./contracts/interfaces/IERC20Permit.sol
SHA3: 11fe6ec451b86b083a3e668c7d3d5a4593b6baa9b3ba8290d99e69e1899c5837

File: ./contracts/interfaces/IGateway.sol
SHA3: 037a1985997ada39f070c182b99fdbf18a4f0c913284fe6a520dae4e5f105efc

File: ./contracts/interfaces/IGatewayClient.sol
SHA3: a9f904f7d7a2297295e47eff4809a2676b490a3041e7541e8e09b1a21b5c6559

File: ./contracts/interfaces/ILayerZeroProxy.sol
SHA3: 8a4a88c30aa13433bad2dbdba5e596b39475ae139d14dca3cd7eaeb4876e6f0a

File: ./contracts/interfaces/IReentrancyCaller.sol
SHA3: 4bdc60d9851fd2c14cd84b76a0238822f1a9778fc4d873c11b6db033d390c820

File: ./contracts/interfaces/IReentrancyCallTarget.sol
SHA3: 30783e54340155fada367267327b595b4ee3e09b3bc4b73b6136c9e95cfb9025

File: ./contracts/interfaces/IRegistry.sol
SHA3: fab744c6fb4929f24f39d0c321f831e73e7220d64359e7ba4f124ede6d8501f7

File: ./contracts/interfaces/IRevenueShare.sol
SHA3: 198d7562ec913ba7ef3609849bd5c3517f5bceb83e5074c4c9c4819f66042593

File: ./contracts/interfaces/ISettings.sol
SHA3: 3a705640b13d219c5ec9f5554c654cdbcbdad93adbde522bceb6993f29ec6cf7

File: ./contracts/interfaces/ISwapRouter.sol
SHA3: f990ffdcae3bbfed07bfc6c3a40f07d3ee3f3996b95d755530e0e5ecb0901f3e

File: ./contracts/interfaces/ITokenBalance.sol
SHA3: 2419fd94560faea412971b051c8be705c6c92fc4e093f997a08e3e0c0a358241

File: ./contracts/interfaces/ITokenBurn.sol
```

```
SHA3: 18f8c48e562a65f2c3bab76e6885779bf1cbb96cd4198f9e2deab05e92d97c8f

File: ./contracts/interfaces/ITokenDecimals.sol
SHA3: 438f7847beb02a18cd9409faf3e5e5f1f52669a9ec01528ee1cb0c0c70311591

File: ./contracts/interfaces/ITokenMint.sol
SHA3: 8f2cdf38c21bde24adc33bf1ed398fe047257c30643116dff74197255d3688c9

File: ./contracts/interfaces/IVariableTokenRecords.sol
SHA3: 1cb8119b448ce2c6bc14b0ce92163aca2d0b8049dc741c2fe12ce33446e9c48e

File: ./contracts/interfaces/IVault.sol
SHA3: d2b6c1280534e45f83252ac87cf5beebd1c1a28f39803c24810f10b0669cbaca

File: ./contracts/interport-token/InterportToken.sol
SHA3: 90a595b1f5fea25dd1b70b01fc3b6914f70f0cf8c4c297ed0791da78cb2a75a5

File: ./contracts/LayerZeroGateway.sol
SHA3: 4fdab72e8c38c5656c4452d7e7cfb57cd0c73991661a5452d651342eac2cdccf

File: ./contracts/ManagerRole.sol
SHA3: 74fc95b5bba00fa1b8fcfd8e74b4e3f4ea6f3e5c1dd3a119bcab9774f49d66cb

File: ./contracts/MinterRole.sol
SHA3: 4fc3cab7260620729d36f5ff0b8c384432715600bad2f6d097d63b16ac7d1716

File: ./contracts/MultichainRouterRole.sol
SHA3: bef59faca61370dcd76c7350ff47109d6ad81d904cf9202cb14d3cb721d1f353

File: ./contracts/NativeTokenAddress.sol
SHA3: 81422949ccf0e135c65633ce5bbfac5de2ff6d7a651d3b0dc895c641d98b6ef6

File: ./contracts/Ownable.sol
SHA3: c01bef7f5397aa38c9ee22b3b09afa5e7a786cc31360a3e0e1458fc98eabe1d1

File: ./contracts/Pausable.sol
SHA3: a51ab305bb66d2998eae51c5be3780f9f1e6705b02901d10182a27985fbc1de0

File: ./contracts/ReentrancyGuard.sol
SHA3: 7a383a665b946855521849defe5ee43cb5cea58e7711b1ec64d44a728e5a3657

File: ./contracts/SafeTransfer.sol
SHA3: d03a3c2f11c3d4d19b16650e062218d98c1d620b96c6e3f6a344c2fe89553f90

File: ./contracts/VariableToken.sol
SHA3: a508985ac5a7dc50b078f2dd3f6ee9aa7a2c498f7d5f9d5e775287743a839552

File: ./contracts/VariableTokenRecords.sol
SHA3: 6730ccefc961fefc88d2536886954e2a27c10ce2c5b217919c3a033b6e37ac4a

File: ./contracts/Vault.sol
SHA3: fd330f72fd6e3725a084ce996b203aebcb4f6a1f215b4e2c1dccd6253e7de7be

File: ./contracts/VaultBase.sol
SHA3: 0634b0eab8f045a9411f3d76c2f5b23077cef4cdeb40f92cb7e76e38c75542ce
```

Second review scope

Repository	https://github.com/Interport-Finance/contracts-interport
Commit	6e1a595aae6920339b27a55e47c1fc0347ed8d1b
Whitepaper	Link

Functional Requirements	Link
Technical Requirements	Link

Contracts:

```
File: ./contracts/ActionExecutor.sol
SHA3: 722388ba1f47f69ba49f5798ebd2bf9e3d365716b403679e4837e44607ddba16

File: ./contracts/ActionExecutorRegistry.sol
SHA3: c6265927b735f2237233cfac8ce27854b8789f4495ab1e63821a1b0d7605205b

File: ./contracts/BalanceManagement.sol
SHA3: 35c0a8bc0970662e2b729bdf63d8ecb4bba73a0956dffbfa2d7c8859ca13fd75

File: ./contracts/CallerGuard.sol
SHA3: 0a67bd424aa97a9df6b9e7c4bd181acd819728d63559db43feb6f834d3485c5b

File: ./contracts/Constants.sol
SHA3: 30768552f013d729695fb0c354093421c196334085332c1e62995a6d8fbb8ece

File: ./contracts/crosschain/anycall-v7/AnyCallV7Gateway.sol
SHA3: 0e38f0e7ce95e78c2f1c088dfad01ab4f8de39d186df29bf7ac40f20e5c60e25

File: ./contracts/crosschain/anycall-v7/interfaces/IAnyCallV7Config.sol
SHA3: 7694c6abdfb3885be4755023e58fbdf11ceb302a113a932d1458e827426440d2

File: ./contracts/crosschain/anycall-v7/interfaces/IAnyCallV7Endpoint.sol
SHA3: 4b11d8812fc0f72a0f7b4f62129ca10d5a484be731d6b185423c06ae071d1739

File: ./contracts/crosschain/anycall-v7/interfaces/IAnyCallV7Executor.sol
SHA3: 0eb48da67e119cc6e258d16435b9f3e23faef897e394d0dbdd7e6b84c0ab5086

File: ./contracts/crosschain/GatewayBase.sol
SHA3: 364cd4b7e2cda774eb07e4e2aad2b27aa12d115bb4088ed789a971240f6a2978

File: ./contracts/crosschain/interfaces/IGateway.sol
SHA3: a20fa2bec3c1d5a123bea28571d32f4db798b47a71ff6a89bf99075cf7519fe8

File: ./contracts/crosschain/interfaces/IGatewayClient.sol
SHA3: ee17fd2fb8860122d04a34155d2098fd4ed5f45e23232fb94505c2639d8f69d3

File: ./contracts/crosschain/layerzero/interfaces/ILayerZeroEndpoint.sol
SHA3: 2adce3f72702d745b09aef34a69c1b067f9d13b0f587e7d5b2235319bd457be3

File: ./contracts/crosschain/layerzero/LayerZeroGateway.sol
SHA3: 199b96522b0339c9d01f2c37541fe2adcf59e60a764177f91fb0882cd31e5eb1

File: ./contracts/crosschain/TargetGasReserve.sol
SHA3: 1dc8e02a6d3c71debbe7e66dc0439b52060e6338bf030d2142f6394e3a279094

File: ./contracts/DataStructures.sol
SHA3: 67a37e0a129b963760e77865aa748380a37b8204183adac022185c708d8ab8ab

File: ./contracts/Errors.sol
SHA3: 2844e3177d14afab5235e4629a33eee29ac64bb188ba7d3c0c5e9d09e16a698a

File: ./contracts/farm/Buyback.sol
SHA3: 5994786b35a7a8ff9fd2918d6a459d8de8ec8335d715be0440df1a4917b6aa52

File: ./contracts/farm/FeeMediator.sol
SHA3: 459a7722efa990c6bfa5255d362451201811794905a6357b08c7316d36c5dc08

File: ./contracts/farm/ITPRevenueShare.sol
SHA3: 2a7a47690748b6fa86ecc440a08b97d484b3b3db135cfa151044d3ceffffc184
```

```
File: ./contracts/farm/LPRevenueShare.sol
SHA3: 588225fb3ca22016b5b7794a76b3bfc350510c203d743c8cb5c2e0523cfe9154

File: ./contracts/farm/RevenueShareBase.sol
SHA3: 57d908c06c6b7c9df3ed3180ab05efe0f684c96a8ef401027a859051ace0900b

File: ./contracts/farm/StablecoinFarm.sol
SHA3: 9ab9994d98d6550c9967ed45422a0a2c5d579f954ce45d89b8adde4babd6c181

File: ./contracts/helpers/AddressHelper.sol
SHA3: b7d31b2ddcd8924f442d7fceab71079ad4eb494b12b41818e0f6b313f3e43728

File: ./contracts/helpers/DecimalsHelper.sol
SHA3: 0781b3201fb32f2beb3ec09ea4bcd3965251033f8b1663339cba278d186fb0fa

File: ./contracts/helpers/GasReserveHelper.sol
SHA3: 6a4fea95e6c891ef139eab4acdefbdced1478cce4787604a5c3741836c746dca

File: ./contracts/helpers/TransferHelper.sol
SHA3: 6a914a64d849a75371d6a70c4a440405dbf19093366decea276e1054e6c3b06c

File: ./contracts/interfaces/IActionDataStructures.sol
SHA3: 1425d9f245d199748075fe390bf5daace40e3d16f2b36a2c80dec0e0c6e98712

File: ./contracts/interfaces/IBuybackToken.sol
SHA3: b13b2f0150f2b9f264cd7d3357f13bb7ac8d2e2963a079d72ead0507435afbc0

File: ./contracts/interfaces/IERC20.sol
SHA3: 9d3b5f32b65cb66dd00b8b966e82b936481ef8433bb6e5dfc3d15aad34863903

File: ./contracts/interfaces/IERC20Permit.sol
SHA3: 016f07ca41a0cb30cebf8931612e49ab401461579be42280215a9c03639cfb76

File: ./contracts/interfaces/IRegistry.sol
SHA3: ba33fe47fb51e0a7f3a2e26e0d560fce3963221145eb7ba77c432fe6acffbc64

File: ./contracts/interfaces/IRevenueShare.sol
SHA3: 1e6b7d4e003ce330b1322c633760df705908498247e70e042db4ba9b1eafa61f

File: ./contracts/interfaces/ISettings.sol
SHA3: 2cdb290041db17cfaf3f9ec44e9b24a590a897cb9d90cc773c97403c0d50e5c3

File: ./contracts/interfaces/ISwapRouter.sol
SHA3: f4118ea94222c00ec131bee9ee224da014db3b525071b6faf7afe4779a679f7a

File: ./contracts/interfaces/ITokenBalance.sol
SHA3: f2420ef7fdcf987e7cc09e5e1c73d255cb65f9b4525bb8ff2501c3ebe02bbd85

File: ./contracts/interfaces/ITokenBurn.sol
SHA3: 4abb9497fba2e698d7912115450c88ebb6e4ee2abf2382e5bc326bf2dc8f0488

File: ./contracts/interfaces/ITokenDecimals.sol
SHA3: 0b332ddb61d77737ea2178549d72f91d43230defad2ac204abe897fe8add3a07

File: ./contracts/interfaces/ITokenMint.sol
SHA3: c93e3ea7cb2240805723f1dab94156781e93c1398582594dfdba88a9776e82ab

File: ./contracts/interfaces/IVariableBalanceRecords.sol
SHA3: c4dab87334f8ffcf5d90172806ea851f908bfd4861bfae146686fe0c6a5aa6e6

File: ./contracts/interfaces/IVault.sol
SHA3: c6378c32eb9441ee90daa5e989c8b53f9d7e2267104b751772c8cd9cb4d15028

File: ./contracts/interport-token/InterportToken.sol
```

```
SHA3: 887498092a0dae56e2e42a6ebae442c4ab308a94f4573342848fb4a2ec106db3

File: ./contracts/Pausable.sol
SHA3: ad21143d05a18755709c2ceafc5f7af0b51a56f6fc0175f1aaacbd7dbf159d09

File: ./contracts/roles/AssetSpenderRole.sol
SHA3: bc71231887fa8460c6e7790235a4353c8d79490f9ea06306f7b82b101c5e3b5e

File: ./contracts/roles/BurnerRole.sol
SHA3: 4ead57f09f3018f876d7663bb39f85fbe0e5c195953e52371d8eec78a485cbc2

File: ./contracts/roles/ManagerRole.sol
SHA3: 59075c157b3c17bda429a3321a95ec80368d572f3b46fb02dcd8de93a6791d87

File: ./contracts/roles/MinterRole.sol
SHA3: e2a08bc128c89da7095954d73867a3e64fb41fc24ac6b370d3b65410f5d7bdc7

File: ./contracts/roles/MultichainRouterRole.sol
SHA3: 7c28e485767d0995a84e48c88b84e23374b4e15ce5a40bdfba02f1c3a523ce8e

File: ./contracts/roles/RoleBearers.sol
SHA3: 744139438e860e317447257c4dec809af5bc1150ae9e8e5fe609631a9bd36d7f

File: ./contracts/SystemVersionId.sol
SHA3: b5467bc38f4b643ceaec0ce759e694fea13e401a8f68993c31b42d61c9ccf908

File: ./contracts/VariableBalanceRecords.sol
SHA3: 10cc5fdda717f48fcd53a19fc6fd6f6b2a30d52732413087a8383825bee57cc7

File: ./contracts/VariableToken.sol
SHA3: 2e7b2863ab2f827ea679609f7518b21c6f5a2ba3cd449ed0c38ac41536a8cf7f

File: ./contracts/Vault.sol
SHA3: b4345007bdb90a2a465db5b0be94401cf81c8621a0a7bc7a3b7f81f176cd92e2

File: ./contracts/VaultBase.sol
SHA3: e6d621bbb0afced72bf2c74dbd3a6fb99f92de6e5fdec64d8250b5b5cd86bc60
```

Third review scope

Repository	https://github.com/Interport-Finance/contracts-interport
Commit	c8bf3ea58469c65fc2210ee750a904011eded131
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link

Contracts:

```
File: ./contracts/ActionExecutor.sol
SHA3: 60517c801a26b6e3dabf11aacdc41367b1e2b9be325974ef78b42e8a1390a9f6

File: ./contracts/ActionExecutorRegistry.sol
SHA3: acb9ab10a32eb95b6b7a5d58c399b8f527dd4a3a5f5687fc20dbb7898dccdf44

File: ./contracts/BalanceManagement.sol
SHA3: 8134f7bb4f58267667e753d8b427275bd69186b2c8e3a18d293eec91b1409cea

File: ./contracts/CallerGuard.sol
SHA3: f4f09ab69ce37b740907b30ce260e6ccc403abd2b32e7225d9fa38d551e6fbd6
```

```
File: ./contracts/Constants.sol
SHA3: 3542dcb0a8de757abee36b9da59dd5cda13bfc42ef5241a87673f9c99d070931

File: ./contracts/crosschain/anycall-v7/AnyCallV7Gateway.sol
SHA3: b4f27d034e3ec88215be68f06e90093d4fd153d2958f4a9b20868cc547bc2a83

File: ./contracts/crosschain/anycall-v7/interfaces/IAnyCallV7Config.sol
SHA3: 24552c4abc5155c1dd427ffe0997229c2e5ecc18fd3e2910947476f8380055a9

File: ./contracts/crosschain/anycall-v7/interfaces/IAnyCallV7Endpoint.sol
SHA3: bff7af5a3e7613d357c6707139b35a1c41eb40dfc854484e5a77f2e62d2422a5

File: ./contracts/crosschain/anycall-v7/interfaces/IAnyCallV7Executor.sol
SHA3: e3103d0da9aa0a0bf0b838b737506056a4f4b465c673c307db785f023ab096f7

File: ./contracts/crosschain/GatewayBase.sol
SHA3: 5f3615bdac4dc263d966e71c1caf58056cbfb5d7ac2d9e81df0841b67fd3c8da

File: ./contracts/crosschain/interfaces/IGateway.sol
SHA3: 8e2979fb70191a17447369799596ec74521fbd417eb50395438ad2fd26e02888

File: ./contracts/crosschain/interfaces/IGatewayClient.sol
SHA3: e33c2ab263441e2da314f7221ce87aa77b9b37d00ceb87b5b523c49aff45fdfe

File: ./contracts/crosschain/layerzero/interfaces/ILayerZeroEndpoint.sol
SHA3: 7603f6dfae0f37ba6738d16e8620b01588edd7c99872446b8e1d26e420ea6568

File: ./contracts/crosschain/layerzero/LayerZeroGateway.sol
SHA3: 154d32913cff76f724de15b554edfff391a889301b4c6ba4ccc2f5856922ade2

File: ./contracts/crosschain/TargetGasReserve.sol
SHA3: 1643595e06bfc496ba7d228b7c88f7b97096db3f484b0e1b84fc7c0deb22f6f1

File: ./contracts/DataStructures.sol
SHA3: c6c9ccfdb83f05530562fa0e5d86fd1b69503409849c80fd451f7d9e78be5ac5

File: ./contracts/Errors.sol
SHA3: ee4b271eecbd6fe331ea544a51d9c980f9a7211fb1f3fb533ac6d920edc0ff0a

File: ./contracts/farm/Buyback.sol
SHA3: ee76b429ebf091c4d3dfb2aea65fd590f2afef986df853c68dd8ac5106de0105

File: ./contracts/farm/FeeMediator.sol
SHA3: db2e3b30d35b78ec5160d5c90bb6e0335ea2db41518cd1e49034867032a64b08

File: ./contracts/farm/ITPRevenueShare.sol
SHA3: 3279abe91ece5fc70619cbf7fbc06e742028a6e02ac82b568c7871cddcd30e8b

File: ./contracts/farm/LPRevenueShare.sol
SHA3: 038e2a70c57058d5f4854adc50a64abad8f1920108e038368068a6d5d30dd9eb

File: ./contracts/farm/RevenueShareBase.sol
SHA3: a0e31827f9fcf778379a34c888a4fb2493dcb75e15c58f24fdd9815f232a0521

File: ./contracts/farm/StablecoinFarm.sol
SHA3: b6c97f1d8084bcacd80800c2ec3844c5b5bda3076c4fc720dcddba1b174e234a

File: ./contracts/helpers/AddressHelper.sol
SHA3: d97b043ed1e3823dd03828b3d05161ac8606577ac2df0a6ad4e3a4c05e534e95

File: ./contracts/helpers/DecimalsHelper.sol
SHA3: 02aa83ffe52c7579e709d2832bccf0d723bb01ab6bbb96842aaaab65ccaf99d4

File: ./contracts/helpers/GasReserveHelper.sol
```

```
SHA3: 098ba7e082c2754993e7e044f5fbd952092a2a6abc201dc99c3ca9fdce808e04

File: ./contracts/helpers/RefundHelper.sol
SHA3: 7a7c17edc8c39761f602aed822b7e7b7f26c2a810e8eddb44bcc78d89968f204

File: ./contracts/helpers/TransferHelper.sol
SHA3: 3f01119391764cd12ea2c5c94b6f43804aff29e58341e6ecd15d0fc7f1c272e4

File: ./contracts/interfaces/IActionDataStructures.sol
SHA3: aa0d54a281902f329b16bca0187cc12bdce485a32bdfb823e578de3fb96463af

File: ./contracts/interfaces/IBuybackToken.sol
SHA3: 23b6605658d88f3a11ea29746d64982a92b521f99766071209acdc7ff7805625

File: ./contracts/interfaces/IERC20.sol
SHA3: de5b78e630697d1d24623cde2314d28ae9f8207bf4babdb6892440a1048a887d

File: ./contracts/interfaces/IERC20Permit.sol
SHA3: 28ccce8b96b98e3c256b897d9a42427dccfcd64dd1820c477bf1ec90145df7a0

File: ./contracts/interfaces/IRegistry.sol
SHA3: 072998e953c193d0a8de1e14360f18772fbaaa4f5583f7ab2a5f44adf4d720b2

File: ./contracts/interfaces/IRevenueShare.sol
SHA3: e6545fa79080b02aad5d3ee54752576b9cad37f47dc44bd349f82eaba6982bb6

File: ./contracts/interfaces/ISettings.sol
SHA3: a86b7196f0d433565e2fe9398f35471cb7c6e6a4199a815c4b28bc7126db47ae

File: ./contracts/interfaces/ISwapRouter.sol
SHA3: f9cfa4cc0b54c314a4c4297c8b9f36db9f3077f1a5ca6ee5c2ade084665d9439

File: ./contracts/interfaces/ITokenBalance.sol
SHA3: 71941b8207e063a5738aaa1de5f7891c72e99be2ba0ad3df0a6885f0ebf26112

File: ./contracts/interfaces/ITokenBurn.sol
SHA3: 3ba9ab73a16c05a142c0e0111e9010d6b8a8a25657b009081830fd80f4f4ba4b

File: ./contracts/interfaces/ITokenDecimals.sol
SHA3: 0509afbb3087266a525c862b2b5fd686b75309411eb4157120c748098ac1a34f

File: ./contracts/interfaces/ITokenMint.sol
SHA3: b773fe9fdfa586e37ddcf2b7b5b7e535184314bf4b2ec4bb44bb78071e6f21f5

File: ./contracts/interfaces/IVariableBalanceRecords.sol
SHA3: 40d11f311f588325dbd88136baa2d46a895494d795beb516e6797f169ad6c73b

File: ./contracts/interfaces/IVault.sol
SHA3: 191194c50901248964167fb30b3c95182fa46d074f6700c9f3f6b3dd7d48a8da

File: ./contracts/interport-token/InterportToken.sol
SHA3: 887498092a0dae56e2e42a6ebae442c4ab308a94f4573342848fb4a2ec106db3

File: ./contracts/MultichainTokenBase.sol
SHA3: 1fcab249364ff68c74fd43915d4f61a4f67e757495c058c6eef9d4990b05b47f

File: ./contracts/Pausable.sol
SHA3: 59877c31204c7ecee4964eb5bec5190987b6611943c7a392315a20653ea1b8b4

File: ./contracts/roles/AssetSpenderRole.sol
SHA3: 92b9003a1d9406f5387c2cbb6938e994e581100d89dd906e6aff1857ca9a7711

File: ./contracts/roles/BurnerRole.sol
SHA3: cfaa15496a133d15fbe91591ce1bb030ebef0b826aefda76f69725331068afde
```

```
File: ./contracts/roles/ManagerRole.sol
SHA3: 7119f372434670ce4df12e2ed2e991da8135cf9d7372ff2448c2e76a754dbaff

File: ./contracts/roles/MinterRole.sol
SHA3: 55e383e57d8efa8a5d046a1504525ee6dfd02608d2e011644d80e0e3ad9a61ba

File: ./contracts/roles/MultichainRouterRole.sol
SHA3: d83454863c447fb026c572c1093672ca70b0022e910f77aa74d6bfeaa7b245f5

File: ./contracts/roles/RoleBearers.sol
SHA3: 4513cb38e4dfbc1388106c9f8600e2ca777db4cd523001839169c7f31b718baf

File: ./contracts/SystemVersionId.sol
SHA3: 49116e033caa46540f7ea8c448581b142bca36f7ee3a6c78ce533a5ac1f48433

File: ./contracts/VariableBalanceRecords.sol
SHA3: 0bb2ec3185b0b6501c21daff56c70a43a4483f5ddf5d7e262b0ac20f2fb4ca32

File: ./contracts/VariableToken.sol
SHA3: 1b812f26736245f8c60ee3be1385fb04436c0c5248d6c1d52fefa3bc4f228820

File: ./contracts/Vault.sol
SHA3: 6bf18ffb5a215d482fb7f4b18380950f7b994eab40414351307329a05b9618bc

File: ./contracts/VaultBase.sol
SHA3: d038cd52fd2cc368e6c011f72bc3cd237ccdd7f49e5f9a4425b4cd740fdc218f
```

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor gas optimization. These issues won't have a significant impact on code execution but affect code quality

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Technical requirements properly describe the system.
- Functional requirements of the system are provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The Development environment is configured.
- The System is well architected and divided by multiple components following the single responsibility principle.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Code is properly covered with tests.

Security score

As a result of the audit, the code does not contain any issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10**.

1
2
3
4
5
6
7
8
9
10
The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
18 January 2023	8	13	5	1
28 February 2023	11	3	3	0
22 March 2023	0	0	0	0

Risks

- The bridging logic highly relies on third-party integrations (LayerZero and Multichain), they could have their own vulnerabilities that are out of the audit's scope.
- In case a transaction fails on the target chain, the off-chain service should refund tokens on the initial chain. The service is out of the audit's scope.
- The system highly depends on the owner and managers. In case of a private keys leak, unauthorized accounts may obtain access to user funds.
- The bridging logic uses different DEXes to swap tokens. The DEXes are out of the audit's scope and could have security vulnerabilities.
- Funds deposited to vaults may be withdrawn by the system, so the depositors may need to wait for liquidity in an original chain, or bridge their iUSDT/iUSDC to another chain and withdraw funds there. In the scope of the audit, it's not possible to verify if the system would have a possibility to bridge iUSDT/iUSDC tokens.
- Contracts may be paused and user funds may be locked.
- According to the MultiChain standard, the *InterportToken* contract system owners are able to mint and burn any amount of user funds.
- The system may be vulnerable to interactions with multiple endpoint tokens. Multiple endpoint tokens may be unexpectedly withdrawn by system managers.
- The reward token should not be collided with any staking token on the *StablecoinFarm* contract to keep user funds safe.
- The *StablecoinFarm* contract may have not enough funds to satisfy earned rewards. However, it is possible to withdraw staked funds at any moment and get the earned value after the contract is fulfilled.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant

Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

Interport is a decentralized exchange that allows cross-chain swaps:

- *Role Contracts* – abstract access control contracts. Allow update/view role for owners.
- *BalanceManagement* – an abstract contract that allows to withdraw unexpected tokens from contract balance.
- *CallerGuard* – an abstract contract with a functionality to decline calls from non-whitelisted contracts.
- *Pausable* – an abstract contract that allows to pause/unpause critical functionality.
- *MultichainTokenBase* – an abstract contract with a mint and burnFrom functionality.
- *VaultBase* – an abstract ERC20-vault contract that inherits *MultichainTokenBase*. Provides the ability to deposit/withdraw funds.
- *Vault* – a contract that inherits *VaultBase*. Functionality:
  - Converting variable token to the main vault asset.
  - Withdrawing funds by asset spenders.
- *VariableToken* – an ERC20 contract used in *Vault*, based on *MultichainTokenBase*.
- *VariableBalanceRecords* – a contract for storing temporary user balances at *ActionExecutor*. It is fully controlled by the *ActionExecutor*.
- *ActionExecutor* – a contract for interaction of cross-chain swaps. Functionality:
  - executeLocal – executes the single-chain token swap.
  - execute – executes a cross-chain token swap.
  - claimVariableToken – allows a variable token claim from the user's variable balance.
  - convertVariableBalanceToVaultAsset – a vault asset claim by user's variable balance.
  - messageFeeEstimate – a cross-chain message fee estimation.
  - calculateLocalAmount – a swap result amount for single-chain actions, taking the system fee into account.
  - calculateVaultAmount – a swap result amount for cross-chain actions, taking the system fee into account.
  - variableBalance – the variable balance of the account.
  - handleExecutionPayload – a cross-chain message handler on the target chain.
- *ActionExecutorRegistry* – a storage contract for *ActionExecutor*.
- *GatewayBase* - a base contract for the AnyCallV7 and LayerZero Gateways that implements shared logic between child contracts. Manages list of peers on other chains.

- *AnyCallV7Gateway* - a contract that implements the cross-chain messaging logic specific to AnyCall v7. It is an intermediate contract between the *ActionExecutor* and the *AnyCall* protocol.
- *LayerZeroGateway* - a contract that implements the cross-chain messaging logic specific to LayerZero. It is an intermediate contract between the *ActionExecutor* and the *LayerZero* protocol.
- *InterportToken* – a simple ERC-20 token with unlimited minting. The contract owner can specify a multichainRouter address, which is allowed to burn the user's tokens.
  It has the following attributes:
  - Name: Interport Token
  - Symbol: ITP
  - Decimals: 18
  - Total supply: unlimited
- *Buyback* – a contract for buyback fee receival. Received funds are swapped to buyback tokens.
- *FeeMediator* – a contract for fee processing. The contract balance is distributed to the destinations based on proportion. The proportion is defined by a contract manager.
  Fee destinations:
  - Buyback contract
  - FeeDistributionLPLockers contract
  - FeeDistributionITPLockers contract
  - Treasury contract
- *StablecoinFarm* – staking/vesting contract. Functionality:
  - stake – allows staking funds for the reward.
  - withdraw – allows withdrawing staked funds.
  - emergencyWithdraw – allows withdrawing staked funds and dropping obtained rewards.
  - setRewardTokenPerSecond – allows managers to set the contract APR.
  - add – allows new staking pool creation.
  - set – allows staking pool APR share updation.
  - vest – allows vesting the pending rewards.
  - withdrawVestedRewards – allows withdrawing the vested rewards.
  - exitEarly – allows withdrawing vested rewards immediately but applies a penalty.
  - lockVesting – allows locking vested rewards on an ITP revenue contract.
  - lockPending – allows locking pending rewards on an ITP revenue contract.
- *RevenueShareBase* – an abstract revenue contract. Functionality:
  - withdraw – allows withdrawing unlocked rewards.
  - claimableRewards – returns obtained rewards for vestings.
  - getReward – allows withdrawing pending rewards.

- *ITPRevenueShare* – a revenue contract based on RevenueShareBase. Functionality:
  - lock – allows locking funds to obtain rewards.
  - lock – allows lockers locking funds on behalf of other users.
- *LPRevenueShare* – a revenue contract based on RevenueShareBase. Functionality:
  - lock – allows locking funds to obtain rewards.

Privileged roles

The *InterportToken* contract has the following privileged roles:

- Owner
  - Can mint and burn tokens
  - Can assign MultichainRouter role
  - Can transfer ownership to any non zero address
- MultichainRouter
  - Can mint and burn tokens

The *Vault* contract has the following privileged roles:

- Owner:
  - can assign a manager role
- Manager:
  - can assign an AssetSpender role
  - can assign a MultichainRouter role
  - can set a variable token
  - can enable or disable variable token repayments
  - can pause or unpause contract functionality
- Multichain router:
  - can mint and burn (using allowance) tokens
- Asset spender:
  - can withdraw any amount of tokens from the vault

The *VariableBalanceRecords* contract has the following privileged roles:

- Owner:
  - can assign a manager role
- Manager:
  - can withdraw any token from the contract
  - can set an ActionExecutor role
- ActionExecutor:
  - can modify the variable token balance for a specific user

The *VariableToken* contract has the following privileged roles:

- Owner:
  - can assign a manager role
- Manager:
  - can assign a minter role
  - can assign a burner role
  - can assign a multichain router role

    - can withdraw any token from the contract
    - can pause or unpause contract functionality
- Minter:
    - can mint variable tokens if *useExplicitAccess* is enabled
- Burner:
    - can burn (using allowance) variable tokens if *useExplicitAccess* is enabled
- MultichainRouter:
    - can mint and burn (using allowance) variable tokens

The *LayerZeroGateway* contract has the following privileged roles:

- Owner:
    - can assign a manager role
- Manager:
    - can assign a client role
    - can specify a Layer Zero proxy address
    - can add/remove peers
    - can add/remove chain id pairs
    - can change target gas
    - can withdraw any token from the contract
    - can pause or unpause contract functionality
- Client:
    - can send a message to layer zero proxy
- Layer Zero endpoint:
    - can bring cross-chain message to the system

The *AnyCallV7Gateway* contract has the following privileged roles:

- Owner:
    - can assign a manager role
- Manager:
    - can withdraw any token from the contract
    - can set any call proxy
    - can assign client role
    - can add/remove peers
    - can pause or unpause contract functionality
    - can change target gas
- Client:
    - can send a message to call proxy
- Any Call endpoint:
    - can bring cross-chain message to the system

The *ActionExecutorRegistry* contract has the following privileged roles:

- Owner:
    - can assign a manager role
- Manager:
    - can change target gas
    - can withdraw any token from the contract

    - can add/remove gateway address
    - can add/remove swap routers
    - can add or update a registered swap router transfer contract address
    - can add/remove vaults
    - can set/unset vault decimals
    - can specify fees
    - can specify fee collector addresses
    - can add/remove from the whitelist
    - can specify min and max swap amount

The *ActionExecutor* contract has the following privileged roles:

- Owner:
    - can assign a manager role
- Manager:
    - can withdraw any token from the contract
    - can set a registry address
    - can set a variable balance records address
    - can pause or unpause contract functionality

The *StablecoinFarm* contract has the following privileged roles:

- Owner:
    - can assign a manager role
    - can specify an ITPRevenueShare contract address
    - can specify a LPRevenueShare contract address
    - can specify a percent share for early exist
- Manager:
    - can withdraw any token from the contract
    - can specify a rewards token per second value
    - can add new pools
    - can change the end time
    - can update reward token allocation point per pool
    - can pause or unpause contract functionality

The *LPRevenueShare* contract have the following privileged roles:

- Owner:
    - can assign a manager role
    - can enable public exit
    - can add reward tokens addresses
- Manager:
    - can withdraw any token from the contract
    - can pause or unpause contract functionality

The *ITPRevenueShare* contract have the following privileged roles:

- Owner:
    - can assign a manager role
    - can enable public exit
    - can add reward tokens addresses

  - can assign or remove lockers roles
- Manager:
  - can withdraw any token from the contract
  - can pause or unpause contract functionality
- Locker:
  - can lock tokens to receive rewards

The *FeeMediator* contract has the following privileged roles:

- Owner:
  - can assign a manager role
- Manager:
  - can withdraw any token from the contract
  - can specify a buyback address
  - can specify fee distribution addresses
  - can specify a treasury address
  - can specify buyback, ITPLockers and LPLockers distribution percents
  - can specify assets addresses
  - can initiate fees processing

The *Buyback* contract has the following privileged roles:

- Owner:
  - can assign a manager role
- Manager:
  - can withdraw any token from the contract
  - can specify a router to native and router from native addresses
  - can specify swap tolerance

Findings

■■■■ Critical

C01. Double spending; Data Consistency

During the *_getReward* call the *_notifyReward* function is called, resulting in double-taken rewards corrupting the *rewardRate* value.

Example flow:
_notifyReward(T, A) is called by a mediator or a farm => *rewardRate = X*
rewardsDuration length gone
claim is called => *_notifyReward(T, A)* is called => *rewardRate = X*
rewardsDuration length gone
claim is called => inconsistent state appears: rewards are offered to the user, but there are no tokens to satisfy them

This may lead to wrong financial data stored on the contract, an unexpectedly high reward rate and inability to satisfy all user rewards debentures.

Paths:
./contracts/farm/LPRevenueShare.sol: _getReward()
./contracts/farm/ITPRevenueShare.sol: _getReward()

Recommendations: rework the logic to keep user rewards consistent during the entire interaction flow.

Status: Fixed (Revised commit: 6e1a595)

■■■ High

H01. Denial of Service Vulnerability

RevenueShare contracts allow locking funds for a specific user (not only for *msg.sender*). During withdrawal, in cases when not all locks are expired, contracts iterate over all user deposits (unlimited).

This may lead to a DoS vulnerability due to exceeding transaction Gas limit.

An attacker may send some dust to a specific address to disable the possibility of funds withdrawal.

Paths:
./contracts/farm/LPRevenueShare.sol : lock()
./contracts/farm/ITPRevenueShare.sol : lock()

Recommendation: implement a limit of maximum active locks or provide a possibility to manage locks by range.

Status: Fixed (Revised commit: 6e1a595)

H02. Insufficient balance; Requirement violation

According to the provided documentation, users should not interact with vaults directly (only through *ActionExecutor*). However, the *VaultBase* contract has a public *deposit* function, so any user is allowed to deposit funds.

Currently, a user could deposit their funds in a vault (without any profit), these funds could be withdrawn by *ActionExecutor* using the *requestAsset* function, which will result in insufficient funds in the vault, so users cannot withdraw their deposits.

Path: ./contracts/VaultBase.sol: deposit()

Recommendation: check if any requirements are violated, consider limiting deposit function access.

Status: Mitigated (client provided documentation, that described this case: if the vault is empty - depositors can bridge their iUSDT/iUSDC tokens to another chain and swap them in another vault)

H03. Invalid Calculations

According to the project workflow, during an *exitEarly* call, the *totalVesting* amount should be distributed through *totalLocked* and *totalPaid* proportionally to exit-early shares. However, according to the current implementation, both the *totalLocked* and *totalPaid* increase the *amountUser* value.

This logic works only when *exitEarlyUserShare == exitEarlyITPShare + exitEarlyLPShare*. However, these values are not constants and could be updated.

This may lead to corruption of the pending rewards calculation.

Path: ./contracts/farm/StablecoinFarm: exitEarly()

Recommendation: rework the logic to distribute any tokens consciously.

Status: Fixed (Revised commit: 6e1a595)

H04. Funds Lock; Requirements violation

During a *_notifyReward* call the *rewardPerTokenStored* is not updated but *lastUpdateTime* is increased to *block.timestamp*, and some unclaimed tokens may be locked on the contract.

Although, during a *_getReward* call, most of the rewards are re-distributed, a requirement violation appears as users should receive rewards in accordance with their lock period.

This may lead to inability for users to claim part of their rewards and incorrect rewards distribution.

Paths:
./contracts/farm/LPRevenueShare.sol: _notifyReward()
./contracts/farm/ITPRevenueShare.sol: _notifyReward()

Recommendation: rework the logic to keep user rewards consistent during the entire interaction flow.

Status: Fixed (Revised commit: 6e1a595)

H05. Undocumented Behavior; Requirement Violation

The System uses both *LayerZero* and *AnyCall* integrations, but only *LayerZero* is mentioned in the docs.

According to the provided documentation - in case of a transaction failure on the destination chain - users can claim tokens on the origin chain as a refund, but the *LayerZero* integration doesn’t support fallback.

Paths:
./contracts/LayerZeroGateway.sol: sendMessage()
./contracts/AnyCalGateway.sol

Recommendation: make documentation and code consistent with each other.

Status: Fixed (Revised commit: 6e1a595)

H06. Highly Permissive Role Access

The owner of the contract may burn any user funds.

User funds should not be accessible without proper allowances.

Paths:
./contracts/interport-token/InterportToken.sol: burn()
./contracts/Vault.sol: burn()
./contracts/VariableToken.sol: burn()

Recommendation: use the *ERC20Burnable* pattern if burnable functionality is needed.

Status: Mitigated (*InterportToken* is deployed without ability of upgrade) & Fixed (Revised commit: c8bf3ea)

H07. Highly Permissive Role Access

The manager of the *ActionExecutor* contract may drop the unpaid user funds table. The function provides an ability to change the balances registry.

The manager of the *Vault* contract may change the variable token address after deployment, which could affect the functionality of the variable tokens to the stablecoins exchange.

This lowers tsystem sustainability and may lead to users being unable to withdraw their funds.

Paths:
./contracts/ActionExecutor.sol: setVariableBalanceRecords()
./contracts/Vault.sol: redeemVariableToken()

Recommendation: block the ability to change important contracts implementations.

Status: Fixed (Revised commit: c8bf3ea)

H08. Requirement Violation

There are no guarantees that there are enough funds to pay rewards.

Reward tokens are not locked on the contract on call to *setRewardTokenPerSecond*. Managers are able to withdraw the reward token from the contract using the *BalanceManagement* functionality.

This may lead to users being unable to withdraw or vest promised rewards until someone puts them on the contract.

Path: ./contracts/farm/StablecoinFarm.sol: isReservedToken(), setRewardTokenPerSecond()

Recommendation: check that there are enough reward tokens on the contract to satisfy promises on rewards payout or auto calculate the *rewardTokenPerSecond* value depending on the reward balance.

Status: Mitigated (according to documentation, the manager should top up the *StablecoinFarm* contract)

Medium

M01. Undocumented Behavior

VariableToken mint and burn are allowed for the multichain router even if contract functionality is paused, but not allowed for the minter role under the same conditions.

The behavior is not documented and looks like a logical mistake.

Path: ./contracts/farm/VariableToken.sol : mint(), burn()

Recommendation: check if existing logic is correct and provide corresponding documentation.

Status: Fixed (Revised commit: 6e1a595)

M02. Requirements Violation

According to the documentation, the mentioned functions should be accessible only by the owner but the contract allows execution by all managers.

Path: ./contracts/farm/StablecoinFarm.sol: add(), set()

Recommendation: update project requirements or fix the issue.

Status: Fixed (Revised commit: c8bf3ea)

M03. Requirement violation

StablecoinFarm.add should not be called for already added tokens, to prevent any problem with rewards. It is mentioned in the comments, but there is no verification if the specific token is already part of the pools, which allows violating the requirement.

Path: ./contracts/farm/StablecoinFarm: add()

Recommendation: provide a safe-check if the token was already added to pools if it is needed.

Status: Fixed (Revised commit: 6e1a595)

M04. Denial of Service Vulnerability

StablecoinFarm allows an unlimited number of pools. Multiple functions use *_massUpdatePools* to update all pools. In case of a high pool amount, the mass pools update may run out of Gas during looping over all of them, so transactions would be reverted and functionality of the contract would be blocked.

Path: ./contracts/farm/StablecoinFarm: _massUpdatePools()

Recommendation: consider limiting max pools number.

Status: Fixed (Revised commit: 6e1a595)

M05. Sign of non-finalized code

The *requestAsset* function parameter's *_forVariableToken* purpose is not intuitive. It is not checked if *variableToken* is not *0x0* as is checked in the *redeemVariableToken* function.

It looks like the code is not finalized. The parameter is possibly redundant.

Path: ./contracts/Vault: requestAsset()

Recommendation: check if the logic inside this function is correct and provide corresponding documentation.

Status: Fixed (Revised commit: 6e1a595)

M06. Fee is not limited

Consider limiting fees to a reasonable max value (10%, for example) to keep contract behavior predictable from user's perspective.

The owner of the contract could change fees at any time. The upper bound limit for fees is not set or specified up to 100%.

Paths:
./contracts/ActionExecutorRegistry.sol: setFallbackFee(), _setSystemFee()
./contracts/farm/Buyback.sol: setSwapTolerance()
./contracts/farm/FeeMediator.sol: _setFeeDistributionLPLockersPart(), _setFeeDistributionITPLockersPart(), _setBuybackPart()

Recommendation: specify an upper limit for the fees.

Status: Fixed (Revised commit: 6e1a595)

M07. Missing return value validation

According to the AnySwap standard, *mint* and *burn* functions return boolean values which may indicate action status. In order to be compatible with different AnySwap tokens, the return value should be validated.

Paths:
./contracts/ActionExecutor.sol: _processVariableTokenClaim()
./contracts/Vault.sol: redeemVariableToken()

Recommendation: keep code consistent, avoid solutions which work only for specific cases.

Status: Fixed (Revised commit: 6e1a595)

M08. Inconsistent flow

The *LPRevenueShare* contract allows fund deposits at any time without any restrictions, but withdrawals could be blocked by pausing the contract.

This may lead to user funds being locked in a critical system state.

Paths:
./contracts/farm/LPRevenueShare.sol: lock()
./contracts/farm/ITPRevenueShare.sol: lock()

Recommendation: do not allow deposits when contract functionality is paused.

Status: Fixed (Revised commit: 6e1a595)

M09. Undocumented behavior

Users can lock their funds in two ways: by *StablecoinFarm.lockVesting* and by *IRevenueShare.lock*. *StablecoinFarm* uses *IRevenueShare.lock* under the hood. Direct funds lock in *IRevenueShare* implementation will lead to incorrect *totalLocked* value in *StablecoinFarm* and affect the *pendingRewardToken* calculation.

As this behavior is not documented it’s unclear if *IRevenueShare* should have limited access to avoid direct locks.

Paths:
./contracts/farm/LPRevenueShare.sol: lock()
./contracts/farm/ITPRevenueShare.sol: lock()
./contracts/farm/StablecoinFarm.sol : lockVesting()

Recommendation: clarify the requirements and check if existing logic is correct.

Status: Fixed (Revised commit: 6e1a595)

M10. Highly permissive role

Any manager is able to withdraw all funds from the contract.

This may lead to conflict situations between managers.

Path: ./contracts/farm/Buyback.sol: withdraw()

Recommendation: send the funds to the treasury or document the functionality.

Status: Mitigated (Client provided documentation stating that highly permissive manager role is part of their system business logic)

M11. Data validation

Upper limits for the *lockDuration* in **RevenueShare* contracts are not validated. In case the owner sets a huge value by mistake, users could lock their funds for this time without the possibility to unlock them earlier.

Paths:
./contracts/farm/LPRevenueShare.sol: lock()
./contracts/farm/ITPRevenueShare.sol: lock()

Recommendation: introduce an upper limit for *lockDuration*.

Status: Fixed (Revised commit: 6e1a595)

M12. Funds lock; Data validation

StablecoinFarm doesn't allow users to withdraw their full deposits, because of validation: *user.amount < _amount*.

Path: ./contracts/farm/StablecoinFarm.sol: withdraw()

Recommendation: change validation to <=.

Status: Fixed (Revised commit: 6e1a595)

M13. Undocumented behavior; Unchecked Call Return Value

The return value of a message call should be checked as an unsuccessful call execution may be missed.

Path: ./contracts/SafeTransfer.sol: safeTransferNativeUnchecked()

Recommendation: validate the call return value or provide documentation for the function.

Status: Fixed (Revised commit: 6e1a595)

M14. Failed call returns success

The contracts implement empty *fallback* functions.

In case a contract is called with the wrong data, the function will accept the call and the call will be considered performed successfully.

This may lead to wrong assumptions on cross-contract call results.

Paths:
./contracts/crosschain/GatewayBase.sol: fallback()
./contracts/ActionExecutor.sol: fallback()

Recommendation: remove empty *fallback* functions.

Status: Fixed (Revised commit: c8bf3ea)

M15. Requirement Violation

According to the requirements, functions should unlock all finalized vestings/locks. However, the data may not be ordered by *unlockTime*. This can cause the break condition to be false-positive.

This may lead to users being unable to withdraw unlocked funds for a period of time or users able to withdraw all locked before the unlock should happen.

Paths:
./contracts/farm/StablecoinFarm.sol: withdrawVestedRewards()
./contracts/farm/RevenueShareBase.sol: withdraw()

Recommendation: do not rely on possibly incorrect assumptions.

Status: Fixed (Revised commit: c8bf3ea)

Low

L01. Floating Pragma

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Path: most of the contracts

Recommendation: consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (Revised commit: 6e1a595)

L02. Modification of a Well-Known Contract

Imported or copy-pasted contracts (such as SafeMath, Context, Ownable, etc.) should not be modified to keep the code clear.

Path:
./contracts/ReentrancyGuard.sol
./contracts/Ownable.sol
./contracts/Pausable.sol
./contracts/SafeTransfer.sol
./contracts/ERC20.sol
./contracts/Address.sol

Recommendation: import the files from the source instead of using copy-paste.

Status: Fixed (Revised commit: 6e1a595)

L03. Unused library

A Library is declared in the project but never used.

Path: ./contracts/Address.sol

Recommendation: remove the unused library.

Status: Fixed (Revised commit: 6e1a595)

L04. Unused constant

The constant is declared in the contract but never used.

Path: ./contracts/ActionExecutor.sol: INFINITY

Recommendation: remove unused constants/variables.

Status: Fixed (Revised commit: 6e1a595)

L05. Missing Events Emitting

Contracts do not emit events after changing important values.

Events for critical state changes should be emitted for tracking things off-chain.

Path: ./contracts/VariableTokenRecords.sol: setActionExecutor()

Recommendation: implement the corresponding event and emit it there.

Status: Fixed (Revised commit: 6e1a595)

L06. Unused import

The import statement is redundant.

import { ZeroAddressError } from './Errors.sol';

Path: ./contracts/VariableTokenRecords.sol

Recommendation: remove unused statements.

Status: Fixed (Revised commit: 6e1a595)

L07. Variables that Could Be Declared Immutable

As the variables are never changed, declaring them *immutable* saves Gas.

Paths:
./contracts/farm/ITPRevenueShare.sol: USDC, USDT, ITP, lockToken
./contracts/farm/LPRevenueShare.sol: USDC, USDT, ITP, lockToken

Recommendation: use the *immutable* attribute for static variables.

Status: Fixed (Revised commit: 6e1a595)

L08. Functions that Could Be Declared as External

public functions that are never called by the contract should be declared *external* to save Gas.

Paths:
./contracts/ActionExecutor.sol: variableBalance()
./contracts/CallerGuard.sol: listedCallerGuardContractCount(), fullListedCallerGuardContractList()
./contracts/Pausable.sol: pause(), unpause()
./contracts/VaultBase.sol: deposit(), withdraw()
./contracts/crosschain/anycall-v7/AnyCallV7Gateway.sol: messageFee()
./contracts/crosschain/layerzero/LayerZeroGateway.sol: messageFee()
./contracts/farm/Buyback.sol: buybackForToken()
./contracts/farm/StablecoinFarm.sol: vest(), withdrawVestedRewards(), exitEarly(), lockVesting(), lockPending()
./contracts/roles/AssetSpenderRole.sol: assetSpenderCount(), fullAssetSpenderList()
./contracts/roles/BurnerRole.sol: burnerCount(), fullBurnerList()
./contracts/roles/ManagerRole.sol: renounceManagerRole(), managerCount(), fullManagerList()
./contracts/roles/MinterRole.sol: minterCount(), fullMinterList()
./contracts/roles/MultichainRouterRole.sol: multichainRouterCount(), fullMultichainRouterList()

Recommendation: use the *external* attribute for functions never called from the contract.

Status: Fixed (Revised commit: c8bf3ea)

L09. Redundant Use of SafeMath

Since Solidity v0.8.0, the overflow/underflow check is implemented via ABIEncoderV2 on the language level - it adds the validation to the bytecode during compilation.

There is no need to use the SafeMath library.

Path: All contracts

Recommendation: remove the SafeMath library.

Status: Fixed (Revised commit: c8bf3ea)

L10. Unscalable functionality

According to the implementation, the *_notifyReward* function should be called only after the *_updateReward* function call. Updating the reward rate without rpt may lead to a Double Spending issue.

However, the code is designed in a way that *_notifyReward* does not force invoke *_updateReward* method.

This may lead to escalation of the Double Spending issue during further development.

Path: ./contracts/farm/RevenueShareBase.sol: _notifyReward()

Recommendation: add @dev comment which describes the fact to the function.

Status: Fixed (Revised commit: c8bf3ea)

L11. Inefficient Gas model

The function is designed to clear the address list and then rewrite it with new entries. However, it may consume an unreasonable amount of gas to add/remove one address to/from the list.

This may lead to additional Gas waste.

Path: ./contracts/farm/ITPRevenueShare.sol: setLockers()

Recommendation: provide functions to add/remove one or several addresses from the list.

Status: Fixed (Revised commit: c8bf3ea)

L12. Inefficient Gas model; Missing validation

The function is payable, but it is not checked that there are enough funds provided to make the call.

It’s recommended to check that *msg.value* is equal to *_action.sourceSwapInfo.fromAmount* to prevent extra Gas usage in case of incorrect call data.

Path: ./contracts/ActionExecutor.sol: execute(), executeLocal()

Recommendation: add check that that *msg.value* equals *_action.sourceSwapInfo.fromAmount*.

Status: Fixed (Revised commit: c8bf3ea)

L13. Missing validation

It’s recommended to specify the upper bound for the limit for a number of items in iterable collections.

Iterations over or return of large lists could lead to a transaction failure due to Gas limit.

Paths:
./contracts/farm/FeeMediator.sol: _setAsset()
./contracts/farm/LPRevenueShare.sol: addReward()
./contracts/farm/ITPRevenueShare.sol: addReward()
./contracts/roles/RoleBearers.sol: _setRoleBearer()
./contracts/ActionExecutorRegistry.sol : setWhitelist(), setVaultDecimals(), setVault(), _setRouter(), _setGateway()
./contracts/CallerGuard.sol : setListedCallerGuardContracts()
./contracts/crosschain/GatewayBase.sol: _setPeer()

Recommendation: specify a meaningful upper bound limit for each array size.

Status: Fixed (Revised commit: c8bf3ea)

L14. Code duplication

The functionality of *DataStructures.uniqueAddressListUpdate* is manually implemented in the function.

Path: ./contracts/ActionExecutorRegistry.sol : setWhitelist()

Recommendation: replace duplicate functionality with existing one.

Status: Fixed (Revised commit: c8bf3ea)

L15. Code duplication

The logic of the *rewardTokenAmount* and *tokenReward* calculation is duplicated between multiple functions.

Path: ./contracts/farm/StablecoinFarm.sol: pendingRewardToken(), _updatePool()

Recommendation: consider moving the functionality to a separate function.

Status: Mitigated (it is a design of a publicly known farm contract)

L16. Requirement violation

According to the implementation, any tokens except for *buybackToken* may be directly withdrawn from the contract using *BalanceManagement* functionality. However, according to the documentation, any tokens should be converted to the *buybackToken* and then withdrawn.

Path: ./contracts/farm/Buyback.sol: isReservedToken()

Recommendation: remove the *BalanceManagement* functionality from the contract or make it clear that the manager may not process the buyback and withdraw the tokens directly.

Status: Mitigated (according to documentation, the manager is able to withdraw any asset from the contract)

L17. Missing Zero Address Validation; Denial of Service

In case zero addresses are provided as token receivers, the *process* function may fail due to transfers to *0x0*.

Path: ./contracts/farm/FeeMediator.sol: constructor(), setBuybackAddress(), setFeeDistributionITPLockersAddress(), setFeeDistributionLPLockersAddress(), setTreasuryAddress(), process()

Recommendation: add the corresponding checks.

Status: Fixed (Revised commit: c8bf3ea)

L18. Unchecked receive

The contracts implement *receive* functions that accept any native token deposit.

In case a contract accepts deposits from a specific address, the check for the source of funding should be implemented in the function.

This may lead to accidently sending funds accepted by the contracts.

Paths:
./contracts/crosschain/GatewayBase.sol: receive()
./contracts/ActionExecutor.sol: receive()

Recommendation: provide corresponding checks.

Status: Mitigated (the *BalanceManagement* functionality allows to withdraw the funds)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.