HashEye

// SMART CONTRACT SECURITY AUDIT

GSquared

Security assessment by HashEye · prepared for Growth Labs

HASHEYE AUDITED

PROJECT	GSquared
CLIENT	Growth Labs
CATEGORY	Blockchain
PUBLISHED	October 1, 2022
REPORT ID	research-gsquared-2022-10-01-175yzn

This report was produced under HashEye's layered review process — **automated detection**, **pattern correlation**, and **senior manual verification** — with every finding signed off by a human reviewer. Full findings detail and on-chain attestation are available on the report page at hasheye.io/audits/research-gsquared-2022-10-01-175yzn.

Growth Labs GSquared Fix Review November 22, 2022 Prepared for: Kristian Domanski Growth Labs Prepared by: Michael Colburn

About HashEye Founded in 2012 and headquartered in New York, HashEye provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high- end security research with a real -world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel. We maintain an exhaustive list of publications at https://github.com/hasheye-io/publications , with links to papers, presentations, public audit reports, and podcast appearances. In recent years, HashEye consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon. We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom. HashEye also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash. To keep up to date with our latest news and announcements, please follow hasheye on Twitter and explore our public repositories at https://github.com/hasheye-io . To engage us directly, visit our "Contact" page at https://www.hasheye.io/contact , or email us at info@hasheye.io . HashEye, Inc. 228 Park Ave S #80688 New York, NY 10003 https://www.hasheye.io info@hasheye.io HashEye 1 GSquared Fix Review PUBLIC

 Test Coverage Disclaimer All activities undertaken by HashEye in association with this project were performed in accordance with a statement of work and agreed upon project plan. Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase. HashEye uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project. When undertaking a fix review, HashEye reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system. HashEye 2 GSquared Fix Review PUBLIC

Table of Contents About HashEye 1 Notices and Remarks 2 Table of Contents 3 Executive Summary 5 Project Summary 6 Project Methodology 7 Project Targets 8 Summary of Fix Review Results 9 Detailed Fix Review Results 11 1. Unbounded loop can cause denial of service 11 2. Lack of two-step process for contract ownership changes 13 3. Non-zero token balances in the GRouter can be stolen 14 4. Uninformative implementation of maxDeposit and maxMint from EIP-4626 16 5. moveStrategy runs of out gas for large inputs 18 6. GVault withdrawals from ConvexStrategy are vulnerable to sandwich attacks 20 7. Stop loss primer cannot be deactivated 23 8. getYieldTokenAmount uses convertToAssets instead of convertToShares 25 9. convertToShares can be manipulated to block deposits 27 10. Harvest operation could be blocked if eligibility check on a strategy reverts 29 11. Incorrect rounding direction in GVault 31 12. Protocol migration is vulnerable to front-running and a loss of funds 34 13. Incorrect slippage calculation performed during strategy investments and divestitures 36 HashEye 3 GSquared Fix Review PUBLIC

14. Potential division by zero in _calcTrancheValue 38 15. Token withdrawals from GTranche are sent to the incorrect address 40 16. Solidity compiler optimizations can be problematic 42 A. Status Categories 43 B. Vulnerability Categories 44 HashEye 4 GSquared Fix Review PUBLIC

Executive Summary Engagement Overview Growth Labs engaged HashEye to review the security of its GSquared Solidity smart contracts. From September 26 to October 7, 2022, a team of four consultants conducted a security review of the client-provided source code, with six person-weeks of effort. Details of the project's scope, timeline, test targets, and coverage are provided in the original audit report. Growth Labs contracted HashEye to review the fixes implemented for issues identified in the original report. On November 7, 2022, one consultant conducted a review of the client-provided source code, with one half person-day of effort. Summary of Findings The original audit uncovered significant flaws that could impact system confidentiality, integrity, or availability. A summary of the original findings is provided below. EXPOSURE ANALYSIS Severity Count High 2 Medium 6 Informational 8 CATEGORY BREAKDOWN Category Count Data Validation 9 Denial of Service 2 Timing 2 Undefined Behavior 3 Overview of Fix Review Results Growth Labs has sufficiently addressed eleven of the issues described in the original audit report, and partially addressed two additional issues. HashEye 5 GSquared Fix Review PUBLIC

Project Summary Contact Information The following managers were associated with this project: Dan Guido , Account Manager Anne Marie Barry , Project Manager dan@hasheye.io annemarie.barry@hasheye.io The following engineers were associated with this project: Gustavo Grieco , Consultant Michael Colburn , Consultant gustavo.grieco@hasheye.io michael.colburn@hasheye.io Anish Naik , Consultant Damilola Edwards , Consultant anish.naik@hasheye.io damilola.edwards@hasheye.io Project Timeline The significant events and milestones of the project are listed below. Date Event September 20, 2022 Pre-project kickoff call September 30, 2022 Status update meeting #1 October 11, 2022 Delivery of report draft October 11, 2022 Report readout meeting November 2, 2022 Delivery of final report November 22, 2022 Delivery of fix review HashEye 6 GSquared Fix Review PUBLIC

Project Methodology Our work in the fix review included the following: • A review of the findings in the original audit report • A manual review of the client-provided source code and configuration material HashEye 7 GSquared Fix Review PUBLIC

Project Targets The engagement involved a review of the fixes implemented in the following target. GSquared Repository https://github.com/groLabs/GSquared-internal/ Version b0cf03fa18b4549bd85c571c00e18ddf3218de59 Type Solidity Platform EVM HashEye 8 GSquared Fix Review PUBLIC

Summary of Fix Review Results The table below summarizes each of the original findings and indicates whether the issue has been sufficiently resolved. ID Title Status 1 Unbounded loop can cause denial of service Resolved 2 Lack of two-step process for contract ownership changes Unresolved 3 Non-zero token balances in the GRouter can be stolen Unresolved 4 Uninformative implementation of maxDeposit and maxMint from EIP-4626 Resolved 5 moveStrategy runs of out gas for large inputs Resolved 6 GVault withdrawals from ConvexStrategy are vulnerable to sandwich attacks Partially Resolved 7 Stop loss primer cannot be deactivated Resolved 8 getYieldTokenAmount uses convertToAssets instead of convertToShares Resolved 9 convertToShares can be manipulated to block deposits Resolved 10 Harvest operation could be blocked if eligibility check on a strategy reverts Partially Resolved 11 Incorrect rounding direction in GVault Resolved 12 Protocol migration is vulnerable to front-running and a loss of funds Resolved HashEye 9 GSquared Fix Review PUBLIC

13 Incorrect slippage calculation performed during strategy investments and divestitures Resolved 14 Potential division by zero in _calcTrancheValue Resolved 15 Token withdrawals from GTranche are sent to the incorrect address Resolved 16 Solidity compiler optimizations can be problematic Unresolved HashEye 10 GSquared Fix Review PUBLIC

Detailed Fix Review Results 1. Unbounded loop can cause denial of service Status: Resolved Severity: High Difficulty: Low Type: Denial of Service Finding ID: TOB-GRO-1 Target: contracts/GVault.sol Description Under certain conditions, the withdrawal code will loop, permanently blocking users from getting their funds. The beforeWithdraw function runs before any withdrawal to ensure that the vault has sufficient assets. If the vault reserves are insufficient to cover the withdrawal, it loops over each strategy, incrementing the _ strategyId pointer value with each iteration, and withdrawing assets to cover the withdrawal amount. 643 function beforeWithdraw (uint256 _assets , ERC20 _token) 644 internal 645 returns (uint256) 646 { 647 // If reserves dont cover the withdrawal, start withdrawing from strategies 648 if (_assets > _token.balanceOf(address (this))) { 649 uint48 _strategyId = strategyQueue.head; 650 while (true) { 651 address _strategy = nodes[_strategyId].strategy; 652 uint256 vaultBalance = _token.balanceOf(address (this)); 653 // break if we have withdrawn all we need 654 if (_assets ⩽ vaultBalance) break ; 655 uint256 amountNeeded = _assets - vaultBalance; 656 657 StrategyParams storage _strategyData = strategies[_strategy]; 658 amountNeeded = Math.min(amountNeeded, _strategyData.totalDebt); 659 // If nothing is needed or strategy has no assets, continue 660 if

(amountNeeded == 0) { 661 continue ; 662 } Figure 1.1: The beforeWithdraw function in GVault.sol#L643-662 HashEye 11 GSquared Fix Review PUBLIC

However, during an iteration, if the vault raises enough assets that the amount needed by the vault becomes zero or that the current strategy no longer has assets, the loop would keep using the same strategyId until the transaction runs out of gas and fails, blocking the withdrawal. Fix Analysis The issue is resolved. The loop logic was reorganized so the pointer to the next element in the list will always be incremented on every iteration. HashEye 12 GSquared Fix Review PUBLIC

2. Lack of two-step process for contract ownership changes Status: Unresolved Severity: Informational Difficulty: High Type: Data Validation Finding ID: TOB-GRO-2 Target: contracts/pnl/PnLFixedRate.sol Description The setOwner() function is used to change the owner of the PnLFixedRate contract. Transferring ownership in one function call is error-prone and could result in irrevocable mistakes. 56 function setOwner (address _owner) external { 57 if (msg.sender ≠ owner) revert PnLErrors.NotOwner(); 58 address previous_owner = msg.sender ; 59 owner = _owner; 60 61 emit LogOwnershipTransferred(previous_owner, _owner); 62 } Figure 2.1: contracts/pnl/PnLFixedRate:56-62 This issue can also be found in the following locations: • contracts/pnl/PnL.sol:36-42 • contracts/strategy/ConvexStrategy.sol:447-453 • contracts/strategy/keeper/GStrategyGuard.sol:92-97 • contracts/strategy/stop-loss/StopLossLogic.sol:73-78 Fix Analysis The issue is not resolved. The Growth Labs team acknowledged the risk but does not consider this an issue. HashEye 13 GSquared Fix Review PUBLIC

3. Non-zero token balances in the GRouter can be stolen Status: Unresolved Severity: Informational Difficulty: Medium Type: Data Validation Finding ID: TOB-GRO-3 Target: GRouter.sol Description A non-zero balance of 3CRV, DAI, USDC, or USDT in the router contract can be stolen by an attacker. The GRouter contract is the entrypoint for deposits into a tranche and withdrawals out of a tranche. A deposit involves depositing a given number of a supported stablecoin (USDC, DAI, or USDT); converting the deposit, through a series of operations, into G3CRV, the protocol's ERC4626-compatible vault token; and depositing the G3CRV into a tranche. Similarly, for withdrawals, the user burns their G3CRV that was in the tranche and, after a series of operations, receives back some amount of a supported stablecoin (figure 3.1). 421 function withdrawFromTrancheForCaller (422 uint256 _amount , 423 uint256 _token_index , 424 bool _tranche , 425 uint256 _minAmount 426) internal returns (uint256 amount) { 427 ERC20(address (tranche.getTrancheToken(_tranche))).safeTransferFrom(428 msg.sender , 429 address (this), 430 _amount 431); 432 // withdraw from tranche 433 // index is zero for ETH mainnet as their is just one yield token 434 // returns usd value of withdrawal 435 (uint256 vaultTokenBalance ,) = tranche.withdraw(436 _amount, 437 0 , 438 _tranche, 439 address (this) 440); 441 442 // withdraw underlying from GVault 443 uint256 underlying = vaultToken.redeem(444 vaultTokenBalance, HashEye 14 GSquared Fix Review PUBLIC

445 address (this), 446 address (this) 447); 448 449 // remove liquidity from 3crv to get desired stable from curve 450 threePool.remove_liquidity_one_coin(451 underlying, 452 int128 (uint128 (_token_index)), //value should always be 0,1,2 453 0 454); 455 456 ERC20 stableToken = ERC20(routerOracle.getToken(_token_index)); 457 458 amount = stableToken.balanceOf(address (this)); 459 460 if (amount < _minAmount) { 461 revert Errors.LTMinAmountExpected(); 462 } 463 464 // send stable to user 465 stableToken.safeTransfer(msg.sender , amount); 466 467 emit LogWithdrawal(msg.sender , _amount, _token_index, _tranche, amount); 468 } Figure 3.1: The withdrawFromTrancheForCaller function in GRouter.sol#L421-468 However, notice that during withdrawals the amount of stableTokens that will be transferred back to the user is a function of the current stableToken balance of the contract (see the highlighted line in figure 3.1). In the expected case, the balance should be only the tokens received from the threePool.remove_liquidity_one_coin swap (see L450 in figure 3.1). However, a non-zero balance could also occur if a user airdrops some tokens or they transfer tokens by mistake instead of calling the expected deposit or withdraw functions. As long as the attacker has at least 1 wei of G3CRV to burn, they are capable of withdrawing the whole balance of stableToken from the contract, regardless of how much was received as part of the threePool swap. A similar situation can happen with deposits. A non-zero balance of G3CRV can be stolen as long as the attacker has at least 1 wei of either DAI, USDC, or USDT. Fix Analysis The issue is not resolved. The Growth Labs team has acknowledged the risk, saying: "We don't consider this an issue as it will have no impact on the underlying protocol, so no fix applied." HashEye 15 GSquared Fix Review PUBLIC

4. Uninformative implementation of maxDeposit and maxMint from EIP-4626 Status: Resolved Severity: Informational Difficulty: High Type: Undefined Behavior Finding ID: TOB-GRO-4 Target: GVault.sol Description The GVault implementation of EIP-4626 is uninformative for maxDeposit and maxMint, as they return only fixed, extreme values. EIP-4626 is a standard to implement tokenized vaults. In particular, the following is specified: • maxDeposit : MUST factor in both global and user-specific limits, like if deposits are entirely disabled (even temporarily) it MUST return 0. MUST return 2

** 256 - 1 if there is no limit on the maximum amount of assets that may be deposited. • maxMint : MUST factor in both global and user-specific limits, like if mints are entirely disabled (even temporarily) it MUST return 0. MUST return 2 ** 256 - 1 if there is no limit on the maximum amount of assets that may be deposited. The current implementation of maxDeposit and maxMint in the GVault contract directly return the maximum value of the uint256 type: 293 /// @notice The maximum amount a user can deposit into the vault 294 function maxDeposit (address) 295 public 296 pure 297 override 298 returns (uint256 maxAssets) 299 { 300 return type(uint256).max; 301 } . . . 315 /// @notice maximum number of shares that can be minted 316 function maxMint (address) public pure override returns (uint256 maxShares) { 317 return type(uint256).max; 318 } HashEye 16 GSquared Fix Review PUBLIC

Figure 4.1: The maxDeposit and maxMint functions from GVault.sol This implementation, however, does not provide any valuable information to the user and may lead to faulty integrations with third-party systems. Fix Analysis The issue is resolved. The functions were updated to return more meaningful values, instead of always returning the maximum uint256 value. HashEye 17 GSquared Fix Review PUBLIC

5. moveStrategy runs of out gas for large inputs Status: Resolved Severity: Informational Difficulty: High Type: Undefined Behavior Finding ID: TOB-GRO-5 Target: GVault.sol Description Reordering strategies can trigger operations that will run out-of-gas before completion. A GVault contract allows different strategies to be added into a queue. Since the order of them is important, the contract provides moveStrategy , a function to let the owner to move a strategy to a certain position of the queue. 500 /// @notice Move the strategy to a new position 501 /// @param _strategy Target strategy to move 502 /// @param _pos desired position of strategy 503 /// @dev if the _pos value is ⩾ number of strategies in the queue, 504 /// the strategy will be moved to the tail position 505 function moveStrategy (address _strategy , uint256 _pos) external onlyOwner { 506 uint256 currentPos = getStrategyPositions(_strategy); 507 uint256 _strategyId = strategyId[_strategy]; 508 if (currentPos > _pos) 509 move(uint48 (_strategyId), uint48 (currentPos - _pos), false); 510 else move(uint48 (_strategyId), uint48 (_pos - currentPos), true); 511 } Figure 5.1: The moveStrategy function from GVault.sol The documentation states that if the position to move a certain strategy is larger than the number of strategies in the queue, then it will be moved to the tail of the queue. This implemented using the move function: 171 /// @notice move a strategy to a new position in the queue 172 /// @param _id id of strategy to move 173 /// @param _steps number of steps to move the strategy 174 /// @param _back move towards tail (true) or head (false) 175 /// @dev Moves a strategy a given number of steps. If the number 176 /// of steps exceeds the position of the head/tail, the 177 /// strategy will take the place of the current head/tail 178 function move (179 uint48 _id , HashEye 18 GSquared Fix Review PUBLIC

180 uint48 _steps , 181 bool _back 182) internal { 183 Strategy storage oldPos = nodes[_id]; 184 if (_steps == 0) return ; 185 if (oldPos.strategy == ZERO_ADDRESS) revert NoIdEntry(_id); 186 uint48 _newPos = !_back ? oldPos.prev : oldPos.next; 187 188 for (uint256 i = 1 ; i < _steps; i++) { 189 _newPos = !_back ? nodes[_newPos].prev : nodes[_newPos].next; 190 } ... Figure 5.2: The header of the move function from StrategyQueue.sol However, if a large number of steps is used, the loop will never finish without running out of gas. A similar issue affects StrategyQueue.withdrawalQueue , if called directly. Fix Analysis The issue is resolved. The loop logic has been reordered to better handle moving a large number of steps. HashEye 19 GSquared Fix Review PUBLIC

6. GVault withdrawals from ConvexStrategy are vulnerable to sandwich attacks Status: Partially Resolved Severity: Medium Difficulty: High Type: Timing Finding ID: TOB-GRO-6 Target: strategy/ConvexStrategy.sol Description Token swaps that may be executed during vault withdrawals are vulnerable to sandwich attacks. Note that this is applicable only if a user withdraws directly from the GVault , not through the GRouter contract. The ConvexStrategy contract performs token swaps through Uniswap V2, Uniswap V3, and Curve. All platforms allow the caller to specify the minimum-amount-out value, which indicates the minimum amount of tokens that a user wishes to receive from a swap. This provides protection against illiquid pools and sandwich attacks. Many of the swaps that the ConvexStrategy contract performs have the minimum-amount-out value hardcoded to zero. But a majority of these swaps can be triggered only by a Gelato keeper, which uses a private channel to relay all transactions. Thus, these swaps cannot be sandwiched. However, this is not the case with the ConvexStrategy.withdraw function. The withdraw function will be called by the GVault contract if the GVault does not have enough tokens for a user withdrawal. If the balance is not sufficient, ConvexStrategy.withdraw will be called to retrieve additional assets to complete the withdrawal request. Note that the transaction to withdraw assets from the protocol will be visible in the public mempool (figure 6.1). 771 function withdraw (uint256 _amount) 772 external 773 returns (uint256 withdrawnAssets , uint256 loss) 774 { 775 if (msg.sender ≠ address (VAULT)) revert StrategyErrors.NotVault(); 776 (uint256 assets , uint256 balance ,) =

_estimatedTotalAssets(false); 777 // not enough assets to withdraw 778 if (_amount ⩾ assets) { 779 balance += sellAllRewards(); 780 balance += divestAll(false); 781 if (_amount > balance) { 782 loss = _amount - balance; HashEye 20 GSquared Fix Review PUBLIC

783 withdrawnAssets = balance; 784 } else { 785 withdrawnAssets = _amount; 786 } 787 } else { 788 // check if there is a loss, and distribute it proportionally 789 // if it exists 790 uint256 debt = VAULT.getStrategyDebt(); 791 if (debt > assets) { 792 loss = ((debt - assets) * _amount) / debt; 793 _amount = _amount - loss; 794 } 795 if (_amount ⩽ balance) { 796 withdrawnAssets = _amount; 797 } else { 798 withdrawnAssets = divest(_amount - balance, false) + balance; 799 if (withdrawnAssets < _amount) { 800 loss += _amount - withdrawnAssets; 801 } else { 802 if (loss > withdrawnAssets - _amount) { 803 loss -= withdrawnAssets - _amount; 804 } else { 805 loss = 0 ; 806 } 807 } 808 } 809 } 810 ASSET.transfer(msg.sender , withdrawnAssets); 811 return (withdrawnAssets, loss); 812 } Figure 6.1: The withdraw function in ConvexStrategy.sol#L771-812 In the situation where the _amount that needs to be withdrawn is more than or equal to the total number of assets held by the contract, the withdraw function will call sellAllRewards and divestAll with _ slippage set to false (see the highlighted portion of figure 6.1). The sellAllRewards function, which will call _sellRewards , sells all the additional reward tokens provided by Convex, its balance of CRV, and its balance of CVX for WETH. All these swaps have a hardcoded value of zero for the minimum-amount-out. Similarly, if _ slippage is set to false when calling divestAll , the swap specifies a minimum-amount-out of zero. By specifying zero for all these token swaps, there is no guarantee that the protocol will receive any tokens back from the trade. For example, if one or more of these swaps get sandwiched during a call to withdraw , there is an increased risk of reporting a loss that will directly affect the amount the user is able to withdraw. Fix Analysis The issue is partially resolved. A minimum withdrawal amount was added as a parameter for withdrawals through the GVault contract to ensure users received the expected funds; HashEye 21 GSquared Fix Review PUBLIC

however, the internal calls to sellAllRewards and divestAll continue to not have slippage protection. According to the Growth Labs team: “This is known and intended, we don't want to stop user withdrawals and guarded in the GRouter . Additional information regarding this will be provided to the user, and an additional withdraw with slippage functionality was added to the GVault in [PR 153] to also provide protection for user’s interactions with the vault.” HashEye 22 GSquared Fix Review PUBLIC

7. Stop loss primer cannot be deactivated Status: Resolved Severity: Medium Difficulty: High Type: Data Validation Finding ID: TOB-GRO-7 Target: strategy/keeper/GStrategyResolver.sol Description The stop loss primer cannot be deactivated because the keeper contract uses the incorrect function to check whether or not the meta pool has become healthy again. The stop loss primer is activated if the meta pool that is being used for yield becomes unhealthy. A meta pool is unhealthy if the price of the 3CRV token deviates from the expected price for a set amount of time. The primer can also be deactivated if, after it has been activated, the price of the token stabilizes back to a healthy value. Deactivating the primer is a critical feature because if the pool becomes healthy again, there is no reason to divest all of the strategy’s funds, take potential losses, and start all over again. The GStrategyResolver contract, which is called by a Gelato keeper, will check to identify whether a primer can be deactivated. This is done via the taskStopStopLossPrimer function. The function will attempt to call the GStrategyGuard.endStopLoss function to see whether the primer can be deactivated (figure 7.1). 46 function taskStopStopLossPrimer () 47 external 48 view 49 returns (bool canExec , bytes memory execPayload) 50 { 51 IGStrategyGuard executor = IGStrategyGuard(stopLossExecutor); 52 if (executor.endStopLoss()) { 53 canExec = true ; 54 execPayload = abi.encodeWithSelector(55 executor.stopStopLossPrimer.selector 56); 57 } 58 } Figure 7.1: The taskStopStopLossPrimer function in GStrategyResolver.sol#L46-58 However, the GStrategyGuard contract does not have an endStopLoss function. Instead, it has a canEndStopLoss function. Note that the executor variable in HashEye 23 GSquared Fix Review PUBLIC

taskStopStopLossPrimer is expected to implement the IGStrategyGuard function, which does have an endStopLoss function. However, the GStrategyGuard contract implements the IGuard interface, which does not have the endStopLoss function. Thus, the call to endStopLoss will simply return, which is equivalent to returning false , and the primer will not be deactivated. Fix Analysis The issue is resolved. The contract now inherits from the correct interface, and the affected function now calls the correct function to deactivate the stop loss primer. HashEye 24 GSquared Fix Review PUBLIC

8. getYieldTokenAmount uses convertToAssets instead of convertToShares Status: Resolved Severity: Medium Difficulty: Low Type: Data Validation Finding ID: TOB-GRO-8 Target: GTranche.sol Description The getYieldTokenAmount function does not properly convert a 3CRV token amount into a G3CRV token amount, which may allow a user to withdraw more or less than expected or lead to imbalanced tranches after a migration. The expected behavior of the getYieldTokenAmount function is to return the number of G3CRV tokens represented by a given 3CRV amount. For withdrawals, this will determine

how many G3CRV tokens should be returned back to the GRouter contract. For migrations, the function is used to figure out how many G3CRV tokens should be allocated to the senior and junior tranches. To convert a given amount of 3CRV to G3CRV, the GVault.convertToShares function should be used. However, the getYieldTokenAmount function uses the GVault.convertToAssets function (figure 8.1). Thus, getYieldTokenAmount takes an amount of 3CRV tokens and treats it as shares in the GVault , instead of assets. 169 function getYieldTokenAmount (uint256 _index , uint256 _amount) 170 internal 171 view 172 returns (uint256) 173 { 174 return getYieldToken(_index).convertToAssets(_amount); 175 } Figure 8.1: The getYieldTokenAmount function in GTranche.sol#L169-175 If the system is profitable, each G3CRV share should be worth more over time. Thus, getYieldTokenAmount will return a value larger than expected because one share is worth more than one asset. This allows a user to withdraw more from the GTranche contract than they should be able to. Additionally, a profitable system will cause the senior tranche to receive more G3CRV tokens than expected during migrations. A similar situation can happen if the system is not profitable.

Fix Analysis The issue is resolved. The function was updated to use the correct value.

9. convertToShares can be manipulated to block deposits Status: Resolved Severity: Medium Difficulty: Medium Type: Data Validation Finding ID: TOB-GRO-9 Target: GVault.sol Description An attacker can block operations by using direct token transfers to manipulate convertToShares , which computes the amount of shares to deposit. convertToShares is used in the GVault code to know how many shares correspond to certain amount of assets: 394 /// @notice Value of asset in shares 395 /// @param _assets amount of asset to convert to shares 396 function convertToShares (uint256 _assets) 397 public 398 view 399 override 400 returns (uint256 shares) 401 { 402 uint256 freeFunds_ = _freeFunds(); // Saves an extra SLOAD if _freeFunds is non-zero. 403 return freeFunds_ == 0 ? _assets : (_assets * totalSupply) / freeFunds_; 404 } Figure 9.1: The convertToShares function in GVault.sol This function relies on the _freeFunds function to calculate the amount of shares: 706 /// @notice the number of total assets the GVault has excluding and profits 707 /// and losses 708 function _freeFunds () internal view returns (uint256) { 709 return _totalAssets() - _calculateLockedProfit(); 710 } Figure 9.2: The _freeFunds function in GVault.sol In the simplest case, _calculateLockedProfit() can be assumed as zero if there is no locked profit. The _totalAssets function is implemented as follows:

820 /// @notice Vault adapters total assets including loose assets and debts 821 /// @dev note that this does not consider estimated gains/losses from the strategies 822 function _totalAssets () private view returns (uint256) { 823 return asset.balanceOf(address (this)) + vaultTotalDebt; 824 } Figure 9.3: The _totalAssets function in GVault.sol However, the fact that _totalAssets has a lower bound determined by asset.balanceOf(address(this)) can be exploited to manipulate the result by "donating" assets to the GVault address. Fix Analysis The issue is resolved. Additional internal bookkeeping was added to eliminate reliance on calls to balanceOf that could be manipulated and result in blocked deposits.

10. Harvest operation could be blocked if eligibility check on a strategy reverts Status: Partially Resolved Severity: Informational Difficulty: Medium Type: Denial of Service Finding ID: TOB-GRO-10 Target: contracts/strategy/keeper/GStrategyGuard.sol Description During harvest, if any of the strategies in the queue were to revert, it would prevent the loop from reaching the end of the queue and also block the entire harvest operation. When the harvest function is executed, a loop iterates through each of the strategies in the strategies queue, and the canHarvest() check runs on each strategy to determine if it is eligible for harvesting; if it is, the harvest logic is executed on that strategy. 312 /// @notice Execute strategy harvest 313 function harvest () external { 314 if (msg.sender ≠ keeper) revert GuardErrors.NotKeeper(); 315 uint256 strategiesLength = strategies.length; 316 for (uint256 i ; i < strategiesLength; i++) { 317 address strategy = strategies[i]; 318 if (strategy == address (0)) continue ; 319 if (IStrategy(strategy).canHarvest()) { 320 if (strategyCheck[strategy].active) { 321 IStrategy(strategy).runHarvest(); 322 try IStrategy(strategy).runHarvest() {} catch Error(... Figure 10.1: The harvest function in GStrategyGuard.sol However, if the canHarvest() check on a particular strategy within the loop reverts, external calls from the canHarvest() function to check the status of rewards could also revert. Since the call to canHarvest() is not inside of a try block, this would prevent the loop from proceeding to the next strategy in the queue (if there is one) and would block the entire harvest operation. Additionally, within the harvest function, the runHarvest function is called twice on a strategy on each iteration of the loop. This could lead to unnecessary waste of gas and possibly undefined behavior.

Fix Analysis The issue is partially resolved. The redundant call to runHarvest has been removed. The Growth Labs teams acknowledged the risk, saying: “We don't consider a try-catch for a view being appropriate, considering that the view can be built in a way to ensure a true/false return

value (i.e., it’s the responsibility of the strategy to ensure that it returns the correct value).” HashEye 30 GSquared Fix Review PUBLIC

11. Incorrect rounding direction in GVault Status: Resolved Severity: Medium Difficulty: Low Type: Data Validation Finding ID: TOB-GRO-11 Target: GVault.sol Description The minting and withdrawal operations in the GVault use rounding in favor of the user instead of the protocol, giving away a small amount of shares or assets that can accumulate over time . convertToShares is used in the GVault code to know how many shares correspond to a certain amount of assets: 394 /// @notice Value of asset in shares 395 /// @param _assets amount of asset to convert to shares 396 function convertToShares (uint256 _assets) 397 public 398 view 399 override 400 returns (uint256 shares) 401 { 402 uint256 freeFunds_ = _freeFunds(); // Saves an extra SLOAD if _freeFunds is non-zero. 403 return freeFunds_ == 0 ? _assets : (_assets * totalSupply) / freeFunds_; 404 } Figure 11.1: The convertToShares function in GVault.sol This function rounds down, providing slightly fewer shares than expected for some amount of assets. Additionally, convertToAssets i s used in the GVault code to know how many assets correspond to certain amount of shares: 406 /// @notice Value of shares in underlying asset 407 /// @param _shares amount of shares to convert to tokens 408 function convertToAssets (uint256 _shares) 409 public 410 view HashEye 31 GSquared Fix Review PUBLIC

411 override 412 returns (uint256 assets) 413 { 414 uint256 _totalSupply = totalSupply; // Saves an extra SLOAD if _totalSupply is non-zero. 415 return 416 _totalSupply == 0 417 ? _shares 418 : ((_shares * _freeFunds()) / _totalSupply); 419 } Figure 11.2: The convertToAssets function in GVault.sol This function also rounds down, providing slightly fewer assets than expected for some amount of shares. However, the mint function uses previewMint , which uses convertToAssets : 204 function mint (uint256 _shares , address _receiver) 205 external 206 override 207 nonReentrant 208 returns (uint256 assets) 209 { 210 // Check for rounding error in previewMint. 211 if ((assets = previewMint(_shares)) == 0) revert Errors.ZeroAssets(); 212 213 _mint(_receiver, _shares); 214 215 asset.safeTransferFrom(msg.sender , address (this), assets); 216 217 emit Deposit(msg.sender , _receiver, assets, _shares); 218 219 return assets; 220 } Figure 12.3: The mint function in GVault.sol This means that the function favors the user, since they get some fixed amount of shares for a rounded-down amount of assets. In a similar way, the withdraw function uses convertToShares : 227 function withdraw (228 uint256 _assets , 229 address _receiver , 230 address _owner 231) external override nonReentrant returns (uint256 shares) { 232 if (_assets == 0) revert Errors.ZeroAssets(); 233 HashEye 32 GSquared Fix Review PUBLIC

234 shares = convertToShares(_assets); 235 236 if (shares > balanceOf[_owner]) revert Errors.InsufficientShares(); 237 238 if (msg.sender ≠ _owner) { 239 uint256 allowed = allowance[_owner][msg.sender]; // Saves gas for limited approvals. 240 241 if (allowed ≠ type(uint256).max) 242 allowance[_owner][msg.sender] = allowed - shares; 243 } 244 245 _assets = beforeWithdraw(_assets, asset); 246 247 _burn(_owner, shares); 248 249 asset.safeTransfer(_receiver, _assets); 250 251 emit Withdraw(msg.sender , _receiver, _owner, _assets, shares); 252 253 return shares; 254 } Figure 11.4: The withdraw function in GVault.sol This means that the function favors the user, since they get some fixed amount of assets for a rounded-down amount of shares. This issue should also be also considered when minting fees, since they should favor the protocol instead of the user or the strategy. Fix Analysis The issue is resolved. The arithmetic has been updated to use ceiling division to always favor the protocol when rounding. HashEye 33 GSquared Fix Review PUBLIC

12. Protocol migration is vulnerable to front-running and a loss of funds Status: Resolved Severity: High Difficulty: High Type: Timing Finding ID: TOB-GRO-12 Target: GMigration.sol Description The migration from Gro protocol to GSquared protocol can be front-run by manipulating the share price enough that the protocol loses a large amount of funds. The GMigration contract is responsible for initiating the migration from Gro to GSquared. The G Migration.prepareMigration function will deposit liquidity into the three-pool and then attempt to deposit the 3CRV LP token into the GVault contract in exchange for G3CRV shares (figure 12.1). Note that this migration occurs on a newly deployed GVault contract that holds no assets and has no supply of shares. 61 function prepareMigration (uint256 minAmountThreeCRV) external onlyOwner { 62 if (!IsGTrancheSet) { 63 revert Errors.TrancheNotSet(); 64 } 65 66 // read senior tranche value before migration 67 seniorTrancheDollarAmount = SeniorTranche(PWRD).totalAssets(); 68 69 uint256 DAI_BALANCE = ERC20(DAI).balanceOf(address (this)); 70 uint256 USDC_BALANCE = ERC20(USDC).balanceOf(address (this)); 71 uint256 USDT_BALANCE = ERC20(USDT).balanceOf(address (this)); 72 73 // approve three pool 74 ERC20(DAI).safeApprove(THREE_POOL, DAI_BALANCE); 75 ERC20(USDC).safeApprove(THREE_POOL, USDC_BALANCE); 76 ERC20(USDT).safeApprove(THREE_POOL, USDT_BALANCE); 77 78 // swap for 3crv 79 IThreePool(THREE_POOL).add_liquidity(80 [DAI_BALANCE, USDC_BALANCE, USDT_BALANCE], 81 minAmountThreeCRV 82); 83 84 //check 3crv amount received 85 uint256 depositAmount = ERC20(THREE_POOL_TOKEN).balanceOf(HashEye 34 GSquared Fix Review PUBLIC

86 address (this) 87); 88 89 // approve 3crv for GVault 90 ERC20(THREE_POOL_TOKEN).safeApprove(address (gVault), depositAmount); 91 92 // deposit into GVault 93 uint256 shareAmount = gVault.deposit(depositAmount, address (this)); 94 95 // approve gVaultTokens for gTranche 96 ERC20(address (gVault)).safeApprove(address (gTranche), shareAmount); 97 } 98 } Figure 12.1: The prepareMigration function in GMigration.sol#L61-98 However, this prepareMigration function call is vulnerable to a share price inflation attack. As noted in this issue , the end result of the attack is that the shares (G3CRV) that the GMigration contract will receive can redeem only a portion of the assets that were originally deposited by GMigration into the GVault contract. This occurs because the first depositor in the GVault is capable of manipulating the share price significantly, which is compounded by the fact that the deposit function in GVault rounds in favor of the protocol due to a division in convertToShares (see TOB-GRO-11). Fix Analysis The issue is resolved. prepareMigration now takes an additional parameter to ensure users receive a minimum amount of shares from their deposit into the vault. HashEye 35 GSquared Fix Review PUBLIC

13. Incorrect slippage calculation performed during strategy investments and divestitures Status: Resolved Severity: Medium Difficulty: Medium Type: Data Validation Finding ID: TOB-GRO-13 Target: strategy/ConvexStrategy.sol Description The incorrect arithmetic calculation for slippage tolerance during strategy investments and divestitures can lead to an increased rate of failed profit-and-loss (PnL) reports and withdrawals. The ConvexStrategy contract is tasked with investing excess funds into a meta pool to obtain yield and divesting those funds from the pool whenever necessary. Investments are done via the invest function, and divestitures for a given amount are done via the divest function. Both functions have the ability to manage the amount of slippage that is allowed during the deposit and withdrawal from the meta pool. For example, in the divest function, the withdrawal will go through only if the amount of 3CRV tokens that will be transferred out from the pool (by burning meta pool tokens) is greater than or equal to the _debt , the amount of 3CRV that needs to be transferred out from the pool, discounted by baseSlippage (figure 13.1). Thus, both sides of the comparison must have units of 3CRV. 883 function divest (uint256 _debt , bool _slippage) internal returns (uint256) { 884 uint256 meta_amount = ICurveMeta(metaPool).calc_token_amount(885 [0 , _debt], 886 false 887); 888 if (_slippage) { 889 uint256 ratio = curveValue(); 890 if (891 (meta_amount * PERCENTAGE_DECIMAL_FACTOR) / ratio < 892 ((_debt * (PERCENTAGE_DECIMAL_FACTOR - baseSlippage)) / 893 PERCENTAGE_DECIMAL_FACTOR) 894) { 895 revert StrategyErrors.LTMinAmountExpected(); 896 } 897 } 898 Rewards(rewardContract).withdrawAndUnwrap(meta_amount, false); 899 return 900 ICurveMeta(metaPool).remove_liquidity_one_coin(HashEye 36 GSquared Fix Review PUBLIC

901 meta_amount, 902 CRV3_INDEX, 903 0 904); 905 } Figure 13.1: The divest function in ConvexStrategy.sol#L883-905 To calculate the value of a meta pool token (mpLP) in terms of 3CRV, the curveValue function is called (figure 13.2). The units of the return value, ratio , are 3CRV/mpLP. 1170 function curveValue () internal view returns (uint256) { 1171 uint256 three_pool_vp = ICurve3Pool(CRV_3POOL).get_virtual_price(); 1172 uint256 meta_pool_vp = ICurve3Pool(metaPool).get_virtual_price(); 1173 return (meta_pool_vp * PERCENTAGE_DECIMAL_FACTOR) / three_pool_vp; 1174 } Figure 13.2: The curveValue function in ConvexStrategy.sol#L1170-1174 However, note that in figure 13.1, meta_amount value, which is the amount of mpLP tokens that need to be burned, is divided by ratio . From a unit perspective, this is multiplying an mpLP amount by a mpLP/3CRV ratio. The resultant units are not 3CRV. Instead, the arithmetic should be meta_amount multiplied by ratio. This would be mpLP times 3CRV/mpLP, which would result in the final units of 3CRV. Assuming 3CRV/mpLP is greater than one, the division instead of multiplication will result in a smaller value, which increases the likelihood that the slippage tolerance is not met. The invest and divest functions are called during PnL reporting and withdrawals. If there is a higher risk for the functions to revert because the slippage tolerance is not met, the likelihood of failed PnL reports and withdrawals also increases. Fix Analysis The issue is resolved. The order of the terms in the arithmetic has been corrected. HashEye 37 GSquared Fix Review PUBLIC

14. Potential division by zero in _calcTrancheValue Status: Resolved Severity: Informational Difficulty: High Type: Data Validation Finding ID: TOB-GRO-14 Target: GTranche.sol Description Junior tranche withdrawals may fail due to an unexpected division by zero error. One of the key steps performed during junior tranche withdrawals is to identify the dollar value of the tranche tokens that will be burned by calling _calcTrancheValue (figure 14.1). 559 function _calcTrancheValue (560 bool _tranche , 561 uint256 _amount , 562 uint256 _total 563) public view returns (uint256) { 564 uint256 factor = getTrancheToken(_tranche).factor(_total); 565 uint256 amount = (_amount * DEFAULT_FACTOR) / factor; 566 if (amount > _total) return _total; 567 return amount; 568 } Figure 14.1: The _calcTrancheValue function in GTranche.sol#L559-568 To calculate the dollar value, the factor function is called to identify how many tokens represent one dollar. The dollar value, amount , is then the token amount provided, _amount , divided by factor . However, an edge case in the factor function will occur if the total supply of tranche tokens (junior or senior) is non-zero while the amount of assets backing those tokens is zero. Practically, this can

happen only if the system is exposed to a loss large enough that the assets backing the junior tranche tokens are completely wiped. In this edge case, the factor function returns zero (figure 14.2). The subsequent division by zero in _calcTrancheValue will cause the transaction to revert. 525 function factor (uint256 _totalAssets) 526 public HashEye 38 GSquared Fix Review PUBLIC

527 view 528 override 529 returns (uint256) 530 { 531 if (totalSupplyBase() == 0) { 532 return getInitialBase(); 533 } 534 535 if (_totalAssets > 0) { 536 return totalSupplyBase().mul(BASE).div(_totalAssets); 537 } 538 539 // This case is totalSupply > 0 && totalAssets == 0, and only occurs on system loss 540 return 0 ; 541 } Figure 14.2: The factor function in GToken.sol#L525-541 It is important to note that if the system enters a state where there are no assets backing the junior tranche, junior tranche token holders would be unable to withdraw anyway. However, this division by zero should be caught in _calcTrancheValue , and the requisite error code should be thrown. Fix Analysis The issue is resolved. The tranche value calculation has been updated to check for a non-zero factor and return a custom error in the event that factor is zero. HashEye 39 GSquared Fix Review PUBLIC

15. Token withdrawals from GTranche are sent to the incorrect address Status: Resolved Severity: Informational Difficulty: Low Type: Data Validation Finding ID: TOB-GRO-15 Target: GTranche.sol Description The GTranche withdrawal function takes in a _recipient address to send the G3CRV shares to, but instead sends those shares to msg.sender (figure 15.1). 212 function withdraw (213 uint256 _amount , 214 uint256 _index , 215 bool _tranche , 216 address _recipient 217) 218 external 219 override 220 returns (uint256 yieldTokenAmounts , uint256 calcAmount) 221 { . [...] . 245 246 trancheToken.burn(msg.sender , factor, calcAmount); 247 token.transfer(msg.sender , yieldTokenAmounts); 248 249 emit LogNewWithdrawal(250 msg.sender , 251 _recipient, 252 _amount, 253 _index, 254 _tranche, 255 yieldTokenAmounts, 256 calcAmount 257); 258 return (yieldTokenAmounts, calcAmount); 259 } Figure 15.1: The withdraw function in GTranche.sol#L219-259 Since GTranche withdrawals are performed by the GRouter contract on behalf of the user, the msg.sender and _recipient address are the same. However, a direct call to GTranche.withdraw by a user could lead to unexpected consequences. HashEye 40 GSquared Fix Review PUBLIC

Fix Analysis The issue is resolved. withdraw was updated to send tokens to the intended recipient instead of msg.sender . HashEye 41 GSquared Fix Review PUBLIC

16. Solidity compiler optimizations can be problematic Status: Unresolved Severity: Informational Difficulty: High Type: Undefined Behavior Finding ID: TOB-GRO-16 Target: GSquared Protocol Description The GSquared Protocol contracts have enabled optional compiler optimizations in Solidity. There have been several optimization bugs with security implications. Moreover, optimizations are actively being developed . Solidity compiler optimizations are disabled by default, and it is unclear how many contracts in the wild actually use them. Therefore, it is unclear how well they are being tested and exercised. Security issues due to optimization bugs have occurred in the past . A medium- to high-severity bug in the Yul optimizer was introduced in Solidity version 0.8.13 and was fixed only recently, in Solidity version 0.8.17 . Another medium-severity optimization bug—one that caused memory writes in inline assembly blocks to be removed under certain conditions — was patched in Solidity 0.8.15. A compiler audit of Solidity from November 2018 concluded that the optional optimizations may not be safe . It is likely that there are latent bugs related to optimization and that new bugs will be introduced due to future optimizations. Fix Analysis The issue is not resolved. The Growth Labs team acknowledged the issue and accepted the risk. HashEye 42 GSquared Fix Review PUBLIC

A. Status Categories The following table describes the statuses used to indicate whether an issue has been sufficiently addressed. Fix Status Status Description Undetermined The status of the issue was not determined during this engagement. Unresolved The issue persists and has not been resolved. Partially Resolved The issue persists but has been partially resolved. Resolved The issue has been sufficiently resolved. HashEye 43 GSquared Fix Review PUBLIC

B. Vulnerability Categories The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document. Vulnerability Categories Category Description Access Controls Insufficient authorization or assessment of rights Auditing and Logging Insufficient auditing of actions or logging of problems Authentication Improper identification of users Configuration Misconfigured servers, devices, or software components Cryptography A breach of system confidentiality or integrity Data Exposure Exposure of sensitive information Data Validation Improper reliance on the structure or values of data Denial of Service A system failure with an availability impact Error Reporting Insecure or insufficient reporting of error conditions Patching Use of an outdated software package or library Session Management Improper identification of authenticated users Testing Insufficient test methodology or test coverage Timing Race conditions or other order-of-operations flaws Undefined Behavior Undefined behavior triggered within the system HashEye 44 GSquared Fix Review PUBLIC

Severity Levels Severity Description Informational The issue does not pose an immediate risk but is relevant to security best practices. Undetermined The extent of the risk was not determined during this engagement. Low The risk is small or is not one the client has indicated is important. Medium User information is at risk; exploitation could pose reputational, legal, or moderate financial risks. High The flaw could affect numerous users and have serious reputational, legal, or financial implications. Difficulty Levels Difficulty Description Undetermined The difficulty of exploitation was not determined during this engagement. Low The flaw is well known; public tools for its exploitation exist or can be scripted. Medium An attacker must write an exploit or will need in-depth knowledge of the system. High An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue. HashEye 45 GSquared Fix Review PUBLIC