HashEye

// SMART CONTRACT SECURITY AUDIT

Shape Gasback

Security assessment by HashEye · prepared for Shape

HASHEYE AUDITED

PROJECT	Shape Gasback
CLIENT	Shape
CATEGORY	Blockchain
PUBLISHED	January 1, 2025
REPORT ID	research-shape-gasback-2025-01-01-111gw5

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-shape-gasback-2025-01-01-111gw5.

Shape Gasback Security Assessment (Summary Report) January 17, 2025 Prepared for: Jacob Gabelhouse Shape Prepared by: Simone Monica and Damilola Edwards

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 repositorie s at https://github.com/hasheye-io . To engage us directly, visit our "Contact" pag e at https://www.hasheye.io/contact , or email us at info@hasheye.io . HashEye, Inc. 497 Carroll St., Space 71, Seventh Floor Brooklyn, NY 11215 https://www.hasheye.io info@hasheye.io

 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.

Table of Contents

Project Summary Contact Information The following project manager was associated with this project: Mike Shelton , Project Manager mike.shelton@hasheye.io The following engineering director was associated with this project: Josselin Feist , Engineering Director, Blockchain josselin.feist@hasheye.io The following consultants were associated with this project: Simone Monica , Consultant Damilola Edwards , Consultant simone.monica@hasheye.io damilola.edwards@hasheye.io Project Timeline The significant events and milestones of the project are listed below. Date Event September 23, 2024 Pre-project kickoff call September 27, 2024 Delivery of report draft October 1, 2024 Report readout meeting January 17, 2025 Delivery of final summary report with fix review

Project Targets The engagement involved a review and testing of the following target. Gasback Repository https://github.com/shape-network/gasback Version 639667a Type Solidity Platform EVM

Executive Summary Engagement Overview Shape engaged HashEye to review the security of the Gasback smart contracts, which allow smart contracts deployed on the chain to register to get back a part of the gas spent through them. A team of two consultants conducted the review from September 23 to September 27, 2024, for a total of two engineer-weeks of eff ort. With full access to source code and documentation, we performed static and dynamic testing of the Gasback codebase, using automated and manual processes. Observations and Impact Our review focused on the Gasback main contract; our primary concerns were assessing whether only the owner of the NFT can withdraw the gas refunded; whether the recipient or the delegated address has a Gitcoin Passport with KYC; and whether tokens can be locked in the smart contracts without a way to get them back. We found that the delegation process does not allow users to interfere with other users' delegation. The process of registering and assigning a smart contract to a specific NFT allows the owner to withdraw the refunded gas; in that case, a smart contract should be registered to one and only one NFT, since otherwise it would be possible to retrieve the gas refunded multiple times. Finally, we found that the distribution of gas refunded function is callable only by the chain's owner, and that the owner correctly assigns the tokens to each NFT and does not lock tokens in the contract. Overall, the codebase is in a good state; however, TOB-SHAPE-1 highlights an edge case that would allow anyone to cause a denial of service to the system at the user-interface level for any user. Recommendations We recommend that the Shape team perform the following steps before deployment: • Remediate the findings disclosed in this report. These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations. • Integrate Slither scan into the project's continuous integration pipeline, pre-commit hooks, or build scripts. • Consider documenting the system's core invariants, and extend the current test suite to include fuzz testing. HashEye 6 Shape Gasback Security Assessment PUBLIC

Summary of Findings The table below summarizes the findings of the review, including type and severity details. ID Title Type Severity 1 _delegateToDelegators may contain incorrect data for any user Data Validation Medium 2 Smart contracts can be assigned to any arbitrary existing NFT Access Controls Informational HashEye 7 Shape Gasback Security Assessment PUBLIC

Detailed Findings 1. _delegateToDelegators may contain incorrect data for any user Severity: Medium Difficulty: Low Type: Data Validation Finding ID: TOB-SHAPE-1 Target: src/Gasback.sol Description The _delegateToDelegators variable contains a set of delegators for an address. It is modified with the initiateDelegation and removeDelegation functions. However, it is possible for anyone to incorrectly add its address to the target _delegateToDelegators variable by calling the initiateDelegation function twice. 408 function initiateDelegation (address _delegateAddress) public { 409 if (msg.sender == _delegateAddress) revert CannotDelegateToSelf(); 410 if (_delegatorToDelegateData[msg.sender].delegateAddress == _delegateAddress) revert AlreadyInitiated(); 411 if (_delegateToDelegators[_delegateAddress].contains(msg.sender)) revert AlreadyInitiated(); 412 413 emit DelegationInitiated(msg.sender , _delegateAddress); 414 415 _delegatorToDelegateData[msg.sender].delegateAddress = _delegateAddress; 416 _delegatorToDelegateData[msg.sender].confirmed = false ; 417 _delegateToDelegators[_delegateAddress].add(msg.sender); 418 } Figure 1.1: The initiateDelegation function (src/Gasback.sol#L413-L425) The initiateDelegation function allows the msg.sender to delegate its possible gas rewards to the _delegateAddress . After some data validation, the msg.sender is added to the _delegateToDelegators[_delegateAddress] variable. The problem is that if msg.sender again calls initiateDelegation with a different _delegateAddress value, it passes the data validation, but the msg.sender will still be present in the _delegateToDelegators variable of the _delegateAddress in the first call. This is wrong, as msg.sender no longer delegates to that address. The first consequence is that, if a user first calls initiateDelegation with address A then calls it again with address B and finally reconsiders and delegates again to address A, this HashEye 8 Shape Gasback Security Assessment PUBLIC

last call to initiateDelegation will fail because _delegateToDelegators[_delegateAddress] already contains the user's address. The second consequence is that the getDelegators , getConfirmedDelegators , and getTokensOwnedByConfirmedDelegators functions will permanently revert. 232 function getDelegators (address _delegateAddress) public view returns (DelegatorData[] memory) { 233 EnumerableSet.AddressSet storage delegators = _delegateToDelegators[_delegateAddress]; 234 DelegatorData[] memory delegatorData = new DelegatorData[](delegators.length()); 235 236 for (uint256 i = 0 ; i < delegators.length(); i++) { 237 address delegatorAddress = delegators.at(i); 238 DelegateData memory delegateData = _delegatorToDelegateData[delegatorAddress]; 239 240 assert(delegateData.delegateAddress == _delegateAddress); 241 242 delegatorData[i] = 243 DelegatorData({delegatorAddress: delegatorAddress, delegateConfirmed: delegateData.confirmed}); 244 } 245 246 return delegatorData; 247 } Figure 1.2: The getDelegators function (src/Gasback.sol#L237-L252) The getDelegators function tries to get all of the delegators of the address passed in as the argument; it does this by iterating the _delegateToDelegators variable. This function has an assertion that verifies that each iterated address has the target address as

the delegated address. This assertion can fail and make the function revert; getConfirmedDelegators and getTokensOwnedByConfirmedDelegators functions also revert, as they call getDelegators . Additionally, calling initiateDelegation with address(0) as an argument makes it impossible to call removeDelegation , as address(0) is used as the sentinel value to represent that no delegation is currently present. However, the _delegateToDelegators for address(0) may contain incorrect data. 421 function removeDelegation () public { 422 address delegateAddress = _delegatorToDelegateData[msg.sender].delegateAddress; 423 if (delegateAddress == address (0)) revert NoDelegateAddress(); 424 emit DelegationRemoved(msg.sender , delegateAddress); 425 426 _delegatorToDelegateData[msg.sender].delegateAddress = address (0); HashEye 9 Shape Gasback Security Assessment PUBLIC

427 _delegatorToDelegateData[msg.sender].confirmed = false ; 428 429 _delegateToDelegators[delegateAddress].remove(msg.sender); 430 } Figure 1.3: The removeDelegation function (src/Gasback.sol#L428-L437) Exploit Scenario Eve calls the initiateDelegation function with Alice's address. She then calls it again with Bob's address. As a result, when the getDelegators , getConfirmedDelegators , and getTokensOwnedByConfirmedDelegators functions are called for Alice's address, they permanently revert, which makes the UI unusable for Alice and eventually for other third-party integration contracts. Recommendations Short term, change the following check in the initiateDelegation function from if (_delegatorToDelegateData[msg.sender].delegateAddress == _delegateAddress) revert AlreadyInitiated(); to revert if the _delegatorToDelegateData[msg.sender].delegateAddress is different than address(0) . Also add a check to revert if _delegateAddress is equal to address(0) . Long term, improve the testing suite by checking that initiateDelegation function cannot be called while a current delegation is in progress. Additionally, test the invariant that the _delegateToDelegators variable contains only actual delegators, ideally with fuzzing. HashEye 10 Shape Gasback Security Assessment PUBLIC

2. Smart contracts can be assigned to any arbitrary existing NFT Severity: Informational Difficulty: High Type: Access Controls Finding ID: TOB-SHAPE-2 Target: src/Gasback.sol Description The assign function in the Gasback contract allows users to associate additional smart contracts with an existing NFT (_tokenId). Once assigned, any Gasback rewards generated by the assigned smart contract are exclusively linked to that NFT. Only the owner of the NFT can later withdraw these rewards. 343 function assign (uint256 _tokenId , address _smartContract) public onlyUnregistered(_smartContract) { 344 if (!addressHasCode(_smartContract)) revert NoCodeAtAddress(_smartContract); 345 346 if (_smartContract ≠ msg.sender) { 347 if (_staticcallOwner(_smartContract) ≠ msg.sender) revert NotAnOwner(); 348 } 349 350 if (_tokenId == 0 || _ownerOf(_tokenId) == address (0)) revert InvalidTokenId(); 351 352 emit Assign(_smartContract, _tokenId); 353 354 _contractToContractData[_smartContract] = 355 ContractData({tokenId: _tokenId, balanceUpdatedBlock: block.number - 1 , totalEarned: 0 }); 356 357 _tokenIdToTokenData[_tokenId].registeredContracts.push(_smartContract); 358 } Figure 2.1: The assign function (src/Gasback.sol#L343-L358) Currently, the function permits any user to assign smart contracts (that they own) to any existing token, including tokens they do not own. As a result, smart contracts can be accidentally assigned to NFTs belonging to others. In such cases, the rewards generated by the assigned smart contract will be redirected to the rightful owner of the NFT, leaving the assignor unable to access their rewards. The assignor has no recourse to reverse the assignment or recover the lost rewards, eff ectively forfeiting the benefi ts generated by the contract. There is also a risk that malicious actors will purposefully use this avenue to HashEye 11 Shape Gasback Security Assessment PUBLIC

assign spam contracts to NFTs they do not own. Exploit Scenario Bob accidentally assigns a smart contract that he controls to Alice's NFT (tokenId). All Gasback rewards generated by the contract will now accrue only to Alice, and Bob has no way to reclaim the smart contract assignment, even if he notices the mistake immediately. Recommendations Short term, consider implementing a two-step smart contract assignment process: • Step 1: The NFT owner should first indicate the intention to assign a smart contract by registering the contract for the tokenId . • Step 2: When the assign function is invoked, it checks if the smart contract has previously registered to be assigned to the NFT before proceeding. Alternatively, consider introducing a function that enables the protocol owner (or an admin) to remove incorrectly assigned smart contracts. This would provide an administrative failsafe in case manual interventions are required to correct mistakes. HashEye 12 Shape Gasback Security Assessment PUBLIC

A. 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 Undefi ned behavior triggered within the system HashEye 13 Shape Gasback Security Assessment 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 14 Shape Gasback Security Assessment PUBLIC

C. Fix Review Results 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. On January 16, 2025, HashEye reviewed the fixes and mitigations implemented by the Shape team for the issues identified in this report. We reviewed each fix to determine its eff ectiveness in resolving the associated issue. In summary, of the issues described in this report, Shape has resolved all two issues. For additional information, please see the Detailed Fix Review Results below. ID Title Status 1 _delegateToDelegators may contain incorrect data for any user Resolved 2 Smart contracts can be assigned to any arbitrary existing NFT Resolved HashEye 15 Shape Gasback Security Assessment PUBLIC

Detailed Fix Review Results TOB-SHAPE-1: _delegateToDelegators may contain incorrect data for any user Resolved in PR 1 . The initiateDelegation function reverts if the _delegatorToDelegateData[msg.sender].delegateAddress is different from address(0) and if _delegateAddress is equal to address(0) . TOB-SHAPE-2: Smart contracts can be assigned to any arbitrary existing NFT Resolved in PR 2 . The process of smart contracts assignment is now in two steps, as recommended. First, the user has to call the initiateAssign function to indicate that he wants to assign a smart contract to a specific tokenId . Finally, he has to call the assign function to finalize the assignment. HashEye 16 Shape Gasback Security Assessment PUBLIC