HashEye

// SMART CONTRACT SECURITY AUDIT

Shape TokenLock

Security assessment by HashEye · prepared for Shape Factory

HASHEYE AUDITED

PROJECT	Shape TokenLock
CLIENT	Shape Factory
CATEGORY	Blockchain
PUBLISHED	March 1, 2026
REPORT ID	research-shape-tokenlock-2026-03-01-1x5gat

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-tokenlock-2026-03-01-1x5gat.

Shape: TokenLock Security Assessment (Summary Report)

March 19, 2026

Prepared for: Jacob Shape Factory

Prepared by: David Pokora

HashEye

PUBLIC

Table of Contents Table of Contents 1

PUBLIC Security Assessment

Project Summary Contact Information The following project manager was associated with this project: Tara Goodwin-Ruffus, Project Manager tara.goodwin-ruffus@hasheye.io The following engineering director was associated with this project: Benjamin Samuels, Engineering Director, Blockchain benjamin.samuels@hasheye.io The following consultants were associated with this project: David Pokora, Consultant david.pokora@hasheye.io Project Timeline The significant events and milestones of the project are listed below. Date Event March 2, 2026 Pre-project kickoff call March 6, 2026 Delivery of report draft March 9, 2026 Report readout meeting March 19, 2026 Delivery of final summary report

PUBLIC Security Assessment

Project Targets The engagement involved reviewing and testing the following target. unofficial-pfp Repository https://github.com/shape-network/unofficial-pfp/ Version e62d8f16c30d42bf3da92aa3cbed2cba4c5a7a24 Type Solidity Platform Ethereum

HashEye 3 Shape: TokenLock

PUBLIC Security Assessment

Executive Summary Engagement Overview Shape Factory engaged HashEye to review the security of its TokenLock Solidity contract. Shape's TokenLock contract allows users to lock ERC20 tokens for an administrator-defined time period and claim a proportionate amount of ERC721 NFTs for the funds they had locked. It provides functionality for administrators to maintain ownership and set configuration, such as time lock periods and units per token. One consultant conducted the review from March 4 to March 5, 2026, for a total of two engineer-days of effort. With full access to source code and documentation, we performed static and dynamic testing of the latest commit at the start of the engagement, using automated and manual processes. Observations and Impact The review focused on assessing whether the TokenLock contract handles ERC20 and ERC721 tokens appropriately and is not prone to trapping or loss of account funds. This included concerns such as but not limited to: access controls for operator methods, arithmetic during transfers, data validation across all methods, consideration for gas limits, and upgradability proxy patterns. The review uncovered one low-severity issue pertaining to the lack of support for ERC20 tokens with fees-on-transfer (TOB-SHAPE-LOCK-1). Additional non-security-related recommendations are noted in appendix B. Recommendations Based on the codebase maturity evaluation and findings identified during the security review, HashEye recommends that Shape Factory take the following steps before deployment: • Remediate the findings disclosed in this report. Addressing both the security-related finding and non-security-related findings in this report will increase compatibility and minimize operator error. All findings should be addressed before any production deployment. • Expand upon the existing test suite. The existing test suite covers the most critical functionality of the system. The test suite should simply be expanded to ensure that uncovered helper methods and dependency functions are also tested, in order to prevent future regressions. This includes Solady's Ownable methods, testing authorization for the setBaseURI method, a test for double-claims, and tests to ensure that the system works after setToken is used to change the token.

HashEye 4 Shape: TokenLock

PUBLIC Security Assessment

Fix Review Summary On March 6, 2026, the Shape Factory team addressed the single security-related finding raised within the report.

The results of the investigation can be found in appendix C.

HashEye 5 Shape: TokenLock

PUBLIC Security Assessment

Summary of Findings The table below summarizes the findings of the review, including type and severity details. ID Title Type Severity 1 TokenLock does not support tokens with fee-on-transfer

Data Validation Low

HashEye 6 Shape: TokenLock

PUBLIC Security Assessment

Detailed Findings 1. TokenLock does not support tokens with fee-on-transfer Severity: Low Difficulty: Low Type: Data Validation Finding ID: TOB-SHAPE-LOCK-1 Target: packages/contracts/src/TokenLock.sol

Description The TokenLock.lockTokens method records the intended transfer amount in its _totalLocked variable before performing an ERC20 transfer of the provided token. It does so using the safeTransferFrom function, which accounts for improper implementations of ERC20.

However, it does not account for ERC20 tokens that charge a fee-on-transfer, meaning that for any requested transfer amount, a portion of the request will go towards a fee, resulting in the actual transfer being less than the requested amount. For TokenLock, this would break internal accounting. $._totalLocked += amount;

[...]

$._token.safeTransferFrom(msg.sender, address(this), amount);

emit LockCreated(lockId, msg.sender, amount, block.timestamp + $._unitTime);

return lockId; Figure 1.1: The lockTokens method does not explicitly check balances and trusts safeTransferFrom. (packages/contracts/src/TokenLock.sol#L241-L267) The following token conditions may be problematic to the system (as further indicated in appendix B) if they involve arbitrary accounting, or block TokenLock's state transitions. They should be evaluated on a case-by-case basis: • Transferring amounts differing from the amount requested. • Changing user balances without interaction, or the system being able to track proportionality.

HashEye 7 Shape: TokenLock

PUBLIC Security Assessment

• Pausability/blacklistability or similar behavior where transfers cannot be made by token holders when they may wish to. Examples of affected tokens include, but are not limited to: rebasing, fee-on-transfer, inflationary/deflationary, or pausable/blacklistable tokens. Exploit Scenario The TokenLock operator deploys the contract to be used with an ERC20 token. The ERC20 token enforces fee-on-transfer, resulting in a lower-than-requested amount to be moved into users' actual accounts. As a result, TokenLock believes that it received the full amount, and its internal accounting deviates increasingly as more transfers accrue. This leads to trapped balances and state transitions. Recommendations Short term, encapsulate the safeTransferFrom call with a balanceOf check for the sender before and after the transfer. Ensure that the actual amount transferred is computed from this delta. Alternatively, reject any tokens that enforce fee-on-transfer or document the lack of support for them. Long term, review the token integration checklist to ensure that token assets are appropriately supported by the system. Document any gaps in support to minimize operator error.

HashEye 8 Shape: TokenLock

PUBLIC Security Assessment

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 Undefined behavior triggered within the system

HashEye 9 Shape: TokenLock

PUBLIC Security Assessment

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 Not Applicable This issue is of informational severity and does not pose an immediate risk, so difficulty does not apply. 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 10 Shape: TokenLock

PUBLIC Security Assessment

B. Code Quality Findings This appendix contains findings that do not have immediate or obvious security implications. However, addressing them may enhance the code's readability and may prevent the introduction of vulnerabilities in the future. • Document supported token contracts. The TokenLock contract accepts arbitrary ERC20 tokens. However, it cannot account for ERC20 tokens that: ○ Are pausable and blacklistable (may result in trapped funds). ○ Are re-basing tokens (dynamically adjusting account balances will affect token balance arithmetic). • Add a nonReentrant modifier on the withdrawExcessTokens method. Although this method can only be called by the administrator, the additional gas charge for the re-entrancy guard is negligible, and it ensures intended behavior as the contract logic changes over time. (packages/contracts/src/TokenLock.sol#L222) • Use a variable to avoid recomputing unlockTime within the lockTokens method. The computation "block.timestamp + $._unitTime" exists in two places within this method. Use a variable to avoid redundant computation and minimize developer error. (packages/contracts/src/TokenLock.sol#L252-L264) • Set or document unit restrictions (e.g., setUnitTime). The operator of the TokenLock contract should ensure that they do not configure the contract such that its state is trapped. For instance, setUnitTime allows an operator to set a very large value that will produce an unreachable unlockTime for any Locks made at the time. (packages/contracts/src/TokenLock.sol#L200-L207) • Update documentation to encourage use of paginated functions in off-chain solutions. Although there are unbounded functions, such as getUserLockIds, getAllUserLocks, and getActiveUserLocks, that perform up to two loops across the entire user array, these can conceptually hit gas or RPC limits. • Update outdated NatSpec documentation. The _getTokenLockStorage and getAllUserLocks functions are preceded by outdated comments regarding what is returned.

HashEye 11 Shape: TokenLock

PUBLIC Security Assessment

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 March 6, 2026, HashEye reviewed the fixes and mitigations implemented by the Shape Factory team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue. In summary, Shape Factory has resolved the one security-related issue described in the report. For additional information, please see the Detailed Fix Review Results below. ID Title Severity Status 1 TokenLock does not support tokens with fee-on-transfer

Low Resolved

HashEye 12 Shape: TokenLock

PUBLIC Security Assessment

Detailed Fix Review Results TOB-SHAPE-LOCK-1: TokenLock does not support tokens with fee-on-transfer Resolved in PR #58. The Shape Factory team indicated that they do not intend to integrate tokens that enforce fee-on-transfer, blacklistability, pausability, or rebasing to their system, which should avoid this issue.

To help mitigate the concern of integrating unsupported tokens, the Shape Factory team introduced a check that reverts if fee-on-transfer is detected in the underlying token. They have updated their documentation to highlight unsupported token types.

HashEye 13 Shape: TokenLock

PUBLIC Security Assessment

D. Fix Review 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 14 Shape: TokenLock

PUBLIC Security Assessment

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 assessments, supporting client organizations in the technology, defense, blockchain, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, Uniswap, Solana, Ethereum Foundation, Linux Foundation, and Zoom. To keep up with our latest news and announcements, please follow hasheye on X or LinkedIn 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 15 Shape: TokenLock

PUBLIC Security Assessment

 Test Coverage Disclaimer

HashEye performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan. Security assessment projects are time-boxed and often rely on information 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 software controls and security properties. 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. A project's time and resource constraints also limit their use.

HashEye 16 Shape: TokenLock

PUBLIC Security Assessment