HashEye

// SMART CONTRACT SECURITY AUDIT

Radius Technology EVMAuth

Security assessment by HashEye · prepared for Radius Technology

HASHEYE AUDITED

PROJECT	Radius Technology EVMAuth
PERFORMED BY	HashEye Security Review Team - Sergey, Brian
RESEARCH	Sergey, Brian
CLIENT	Radius Technology
CATEGORY	Ethereum/EVM
PUBLISHED	October 1, 2025
REPORT ID	research-radius-technology-evmauth-2025-10-01-1kyeno

This report was produced by HashEye's security review team under the HashEye review process, combining source review, deployment-state checks, and senior manual verification. Full report detail is available at hasheye.io/audits/research-radius-technology-evmauth-2025-10-01-1kyeno.

Radius Technology EVMAuth Security Assessment October 3, 2025

Prepared for: Neil Smithline & Kevin Karwaski Radius Technology

Prepared by: Quan Nguyen

HashEye

PUBLIC

Table of Contents Table of Contents 1

Project Summary 2

Executive Summary 3

Project Goals 5

Project Targets 6

Project Coverage 7

Codebase Maturity Evaluation 9

Summary of Findings 11

Detailed Findings 12

1. Incorrect account assignment in token burning logic 12

2. Expired token groups not synchronized with ERC1155 balance tracking 14

3. Missing debt validation in group transfer function 16

4. Missing debt validation in group burn function 18

5. Unbounded group array growth causes gas limit exceeded 20

6. Inconsistent implementation between blacklist functions and batch versions 22

7. TTL validation missing when updating burnable status 24

A. Vulnerability Categories 26

B. Code Maturity Categories 28

C. Code Quality Recommendations 30

D. Fix Review Results 31

E. Fix Review Targets 33

F. Fix Review Status Categories 34

About HashEye 35

Notices and Remarks 36

PUBLIC Security Assessment

Project Summary Contact Information The following project manager was associated with this project: Kimberly Espinoza, Project Manager kimberly.espinoza@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: Quan Nguyen, Consultant quan.nguyen@hasheye.io Project Timeline The significant events and milestones of the project are listed below. Date Event July 30, 2025 Pre-project kickoff call August 11, 2025 Delivery of report draft August 11, 2025 Report readout meeting September 22, 2025 Completion of fix review (appendix D) September 24, 2025 Delivery of final comprehensive report October 3, 2025 Delivery of final comprehensive report with updated fix review

HashEye 2 Radius Technology EVMAuth

PUBLIC Security Assessment

Executive Summary Engagement Overview Radius Technology engaged HashEye to review the security of the EVMAuth smart contract. This codebase implements an ERC-1155-based authentication token system with time-based expiration, grouped balance accounting, role-based access control, blacklist administration, and purchasable issuance. A team of one consultant conducted the review from August 4 to August 8, 2025, for a total of one engineer-week of effort. Our testing efforts focused on a comprehensive assessment of ERC-1155 compliance and state synchronization, token lifecycle and expiration mechanics, grouped accounting design and transfer semantics, access control and blacklist workflows, metadata configuration (TTL and burnability), and gas/denial of service (DoS) characteristics of group operations. With full access to source code and documentation, we performed static and dynamic testing of the EVMAuth smart contract, using automated and manual processes. Observations and Impact The contracts present a layered ERC-1155 design for authentication tokens with time-based semantics, clear role separation, and modular feature composition. However, we identified several issues with meaningful security and operational impact: burning assigns the wrong account during burn operations (TOB-RADIUS-1); expired token group cleanup does not synchronize with ERC-1155 balances, enabling transfers and use of expired tokens (TOB-RADIUS-2); group transfers and burns skip expired entries without ensuring that the remaining debt is cleared, causing divergence between ERC-1155 balances and group state (TOB-RADIUS-3 & TOB-RADIUS-4); unbounded group arrays and ordered insertions create gas-heavy paths that can exceed block limits and DoS user actions (TOB-RADIUS-5); TTL can remain configured when burnability is toggled, leading to inconsistent lifecycle configuration (TOB-RADIUS-7); and batch blacklist operations omit validations and events, allowing zero address to be blacklisted, which can disrupt mint/burn semantics (TOB-RADIUS-6). Recommendations Based on the codebase maturity evaluation and findings identified during the security review, HashEye recommends that Radius Technology take the following steps: • Remediate the findings disclosed in this report. We recommend immediate remediation by fixing burn account selection, synchronizing expiration cleanup with ERC1155 balances or block transfers of expired amounts, cap or merge groups to avoid linear hot paths, enforce transfer and burn completeness by requiring debt to be cleared, validate or reset TTL when toggling burnability, and align batch blacklist behavior with single-account functions, including checks and events.

HashEye 3 Radius Technology EVMAuth

PUBLIC Security Assessment

• Address gas scalability. Coalesce or cap per-account groups, replace array-shifting structures with more efficient alternatives such as time buckets or head-indexed lists, and avoid linear reads in balance queries by caching non-expired totals. • Strengthen the test suites. We recommend expanding unit test coverage for lifecycle and min/burn/transfer scenarios, including gas benchmarks and budgets, and introducing CI gates for coverage thresholds and static analysis. Finding Severities and Categories

The following tables provide the number of findings by severity and category. EXPOSURE ANALYSIS Severity Count High 4 Medium 1 Low 1 Informational 1 Undetermined 0

CATEGORY BREAKDOWN Category Count Configuration 1 Data Validation 3 Denial of Service 3

HashEye 4 Radius Technology EVMAuth

PUBLIC Security Assessment

Project Goals The engagement was scoped to provide a security assessment of the Radius Technology EVMAuth ERC1155 token contract. Specifically, we sought to answer the following non-exhaustive list of questions: • Does the token expiration mechanism correctly manage token lifecycles without allowing expired tokens to remain usable or create inconsistencies between balance tracking systems? • Can an attacker manipulate the group array management system to cause gas limit issues that prevent legitimate users from performing token operations? • Do the smart contracts handle edge cases appropriately when tokens are transferred between accounts with mixed valid and expired token groups? • Are there appropriate safeguards to prevent unauthorized access through manipulation of authentication token expiration and burning mechanisms? • Is the EVMAuth token implementation compliant with the ERC-1155 specification while maintaining proper expiration tracking? • Do the smart contracts properly implement the intended authentication token design features with appropriate access controls and blacklist management? • Does the codebase conform to industry best practices for gas efficiency, data consistency, and security validation? • Are the batch operations consistent with their single-operation counterparts in terms of validation, event emission, and error handling?

HashEye 5 Radius Technology EVMAuth

PUBLIC Security Assessment

Project Targets The engagement involved reviewing and testing the following target. evmauth-core

Repository github.com/evmauth/evmauth-core Version 63835cf772c8b95e6a1bd69cdf6f47834c356eca Type Solidity Platform EVM

HashEye 6 Radius Technology EVMAuth

PUBLIC Security Assessment

Project Coverage This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following: • Architecture and inheritance model review. We examined the EVMAuth contract hierarchy and its extensions (notably EVMAuthExpiringERC1155), mapping how ownership, access control, blacklisting, TTL, and grouped-expiration logic compose over OpenZeppelin ERC-1155. • Expiration and grouped accounting analysis. We performed a deep review of the grouped-expiration subsystem, assessing FIFO semantics, insertion ordering, pruning conditions, and invariants between ERC-1155 balances and group arrays. • ERC-1155 integration and accounting model. We reviewed how standard ERC-1155 balance updates and events integrate with the system’s domain logic, including hooks, overrides, and state storage patterns that support authentication semantics. • Transfer, mint, and burn flow tracing. We traced end-to-end flows through internal update pipelines to evaluate parameter handling, address semantics (including the zero address), event emission, and consistency of state transitions across operations. • Gas efficiency and DoS analysis. We analyzed the gas costs of group operations and time-dependent paths through targeted scenario construction and micro-benchmarks, focusing on workload patterns likely to occur in production deployments. • Access control and blacklist evaluation. We evaluated role-gated entrypoints and blacklist operations for consistency, validating single vs. batch behaviors, zero-address handling, and event emission. • Metadata and lifecycle configuration. We reviewed configuration paths related to TTL, burnability, and transferability to understand lifecycle policy expression and how configuration changes propagate through the system. Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

HashEye 7 Radius Technology EVMAuth

PUBLIC Security Assessment

• Deployment and operational procedures. Deployment scripts, operational runbooks, and upgrade/migration strategies (if any) received limited attention relative to on-chain logic. • External integration review. This engagement did not assess off-chain systems that consume emitted events (e.g., monitoring pipelines) and third-party integrators.

HashEye 8 Radius Technology EVMAuth

PUBLIC Security Assessment

Codebase Maturity Evaluation HashEye uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs. Category Summary Result Arithmetic The contracts use Solidity 0.8.x's checked arithmetic and perform only straightforward balance updates. There is no custom fixed-point math or unchecked arithmetic, and we did not observe nonstandard arithmetic patterns. This reduces the risk of overflow/underflow and simplifies reasoning about state transitions. Satisfactory Auditing The system emits lifecycle events (e.g., ExpiredTokensBurned) that aid off-chain monitoring. However, batch blacklist operations omit event emissions present in their single-account counterparts (TOB-RADIUS-6). This reduces operational visibility and complicates incident response or audit trails, particularly when large batch changes must be traced. Moderate Authentication / Access Controls Role-based access control is consistently used to gate administrative actions. That said, batch blacklist functions lack zero-address validation, enabling accidental blacklisting of address(0) (TOB-RADIUS-6), which can disrupt mint and burn semantics that rely on the zero address. Aligning validations between single and batch flows would strengthen the access-control posture. Moderate Complexity Management The inheritance structure cleanly layers features on top of OpenZeppelin ERC-1155, keeping most feature sets separated and easier to navigate. The grouped-expiration subsystem, however, introduces dual accounting (ERC-1155 balances vs. group arrays) and cross-cutting concerns (pruning, transfers, insertion order) that meaningfully increase complexity. This design has already produced inconsistencies such as pruning not syncing ERC-1155 balances (TOB-RADIUS-2) and missing debt validation (TOB-RADIUS-3 and TOB-RADIUS-4). Moderate

HashEye 9 Radius Technology EVMAuth

PUBLIC Security Assessment

Documentation High-level documentation describes the system and roles, but detailed specifications of the group-array expiration model are limited. The codebase would benefit from explicit documentation of invariants between ERC-1155 balances and group arrays, pruning semantics, and how TTL interacts with the burnability setting. Clearer guidance would improve maintainability and reduce the likelihood of regressions. Moderate Low-Level Manipulation No inline assembly or low-level call patterns are used. Not Applicable Testing and Verification The test suite has insufficient coverage, leaving core functionalities untested and allowing preventable issues like TOB-RADIUS-1, TOB-RADIUS-2, TOB-RADIUS-3, and TOB-RADIUS-4 to emerge. These vulnerabilities could have been caught with basic unit tests. The codebase requires expanded test coverage across all functionalities, comprehensive scenario testing, and gas usage measurement tests to ensure system reliability and security. Weak Transaction Ordering We did not identify ordering-dependent behaviors that could be exploited to manipulate balances or system invariants. Satisfactory

HashEye 10 Radius Technology EVMAuth

PUBLIC Security Assessment

Summary of Findings The table below summarizes the findings of the review, including details on type and severity. ID Title Type Severity 1 Incorrect account assignment in token burning logic Denial of Service High 2 Expired token groups not synchronized with ERC1155 balance tracking Data Validation High 3 Missing debt validation in group transfer function Data Validation High 4 Missing debt validation in group burn function Data Validation High 5 Unbounded group array growth causes gas limit exceeded Denial of Service Medium 6 Inconsistent implementation between blacklist functions and batch versions Denial of Service Low 7 TTL validation missing when updating burnable status Configuration Informational

HashEye 11 Radius Technology EVMAuth

PUBLIC Security Assessment

Detailed Findings 1. Incorrect account assignment in token burning logic Severity: High Difficulty: Low Type: Denial of Service Finding ID: TOB-RADIUS-1 Target: src/base/EVMAuthExpiringERC1155.sol

Description The token burning logic incorrectly assigns the wrong account address when burning tokens. When tokens are being burned, the code incorrectly assigns address _account = to instead of

using the from address, causing the contract to attempt burning tokens from the zero address. The _update function is responsible for handling token minting, burning, and transferring operations. When burning tokens, the logic should burn tokens from the from address (the token holder), but the current implementation incorrectly sets _account = to, which is address(0) (the zero address). This means that the contract attempts to burn tokens from the zero address instead of the actual token holder, which will fail since the zero address has no tokens to burn. // Burning else if (to == address(0)) { address _account = to; _burnGroupBalances(_account, _id, _amount); _pruneGroups(_account, _id); } Figure 1.1: Incorrect account assignment in burning logic in _update function Exploit Scenario Alice has authentication tokens that grant access to a premium service. The service relies on an admin to manually burn tokens when Alice ends her subscription. When the admin attempts to burn Alice's authentication tokens to revoke access, the burn transaction keeps reverting. This means Alice's authentication tokens remain in their account even after the admin's burn operation, allowing her to continue accessing premium services she should no longer have access to. Recommendations Short term, correct the account assignment in the burning logic in the _update function.

HashEye 12 Radius Technology EVMAuth

PUBLIC Security Assessment

Long term, implement comprehensive unit tests that cover all token operations (mint, burn, transfer) with various edge cases, including burning tokens from different accounts and with different token IDs.

HashEye 13 Radius Technology EVMAuth

PUBLIC Security Assessment

2. Expired token groups not synchronized with ERC1155 balance tracking Severity: High Difficulty: Low Type: Data Validation Finding ID: TOB-RADIUS-2 Target: src/base/EVMAuthExpiringERC1155.sol

Description The _pruneGroups function removes expired token groups from the custom group tracking system but fails to update the underlying ERC1155 balance tracking. This creates a data inconsistency where expired tokens can still be transferred despite being removed from the expiration tracking system. The contract maintains two separate balance tracking systems: the standard ERC1155 _balances mapping and a custom _group array system for expiration management. When tokens expire, the _pruneGroups function correctly removes expired groups from the _group array and emits an ExpiredTokensBurned event, but it does not call the parent contract's burn function to update the ERC1155 balance tracking. This means that the underlying ERC1155 balance remains unchanged, allowing expired tokens to be transferred to other addresses. function _pruneGroups(address account, uint256 id) internal { Group[] storage groups = _group[account][id]; uint256 _now = block.timestamp;

// Shift valid groups to the front of the array uint256 index = 0; uint256 expiredAmount = 0; for (uint256 i = 0; i < groups.length; i++) { bool isValid = groups[i].balance > 0 && groups[i].expiresAt > _now; if (isValid) { if (i != index) { groups[index] = groups[i]; } index++; } else { expiredAmount += groups[i].balance; } }

// Remove invalid groups from the end of the array while (groups.length > index) { groups.pop(); }

HashEye 14 Radius Technology EVMAuth

PUBLIC Security Assessment

// If any expired groups were removed, emit an event with the total amount of expired tokens if (expiredAmount > 0) { emit ExpiredTokensBurned(account, id, expiredAmount); } } Figure 2.1: ERC1155 balance is not updated in _pruneGroups function Exploit Scenario Alice has 100 authentication tokens with a 60-second expiration time. After the tokens expire, the _pruneGroups function correctly removes the expired token groups, but the underlying ERC1155 balance tracking still shows 100 tokens. Alice can then transfer 50 of these "expired" tokens to Bob. After the transfer, both Alice and Bob have 50 tokens in their ERC1155 balance, but the group tracking system shows 0 tokens for both accounts. Recommendations Short term, modify the _pruneGroups function to call the parent contract's burn function when expired tokens are detected, ensuring that both balance tracking systems remain synchronized. Long term, implement comprehensive unit tests that cover all token operations with various expiration scenarios, including edge cases where tokens expire during

transfers, and ensure that both balance tracking systems remain consistent throughout all operations.

HashEye 15 Radius Technology EVMAuth

PUBLIC Security Assessment

3. Missing debt validation in group transfer function Severity: High Difficulty: Low Type: Data Validation Finding ID: TOB-RADIUS-3 Target: src/base/EVMAuthExpiringERC1155.sol

Description The _transferGroups function fails to complete group transfers when expired tokens are present, creating an inconsistency between the ERC1155 balance tracking and the group array tracking. The function skips over expired token groups during transfer but does not verify that the transfer was completed successfully. The _transferGroups function iterates through the sender's token groups in the FIFO order to transfer the requested amount. However, it skips over expired token groups without accounting for them in the transfer calculation. When the loop completes, there is no verification that the debt variable has reached zero, meaning that the transfer may be incomplete. This creates a mismatch where the ERC1155 balance tracking correctly reflects the transfer, but the group arrays contain inconsistent data. function _transferGroups(address from, address to, uint256 id, uint256 amount) internal { // Exit early if the transfer is to the same account or if the amount is zero if (from == to || amount == 0) return;

Group[] storage groups = _group[from][id]; uint256 _now = block.timestamp; uint256 debt = amount;

// First pass: Reduce balances from sender's groups (FIFO order) for (uint256 i = 0; i < groups.length && debt > 0; i++) { // Skip token groups that are expired or have no balance if (groups[i].expiresAt <= _now || groups[i].balance == 0) { continue; }

if (groups[i].balance > debt) { // Transfer partial token group _upsertGroup(to, id, debt, groups[i].expiresAt); groups[i].balance -= debt; debt = 0; } else { // Transfer entire token group

HashEye 16 Radius Technology EVMAuth

PUBLIC Security Assessment

_upsertGroup(to, id, groups[i].balance, groups[i].expiresAt); debt -= groups[i].balance; groups[i].balance = 0; } }

// Clean up from account token groups that are expired or have zero balance _pruneGroups(from, id); } Figure 3.1: Missing debt validation in the _transferGroups function Exploit Scenario Alice has five authentication tokens, including two expired tokens and three tokens that have not expired yet. Alice attempts to transfer four tokens to Bob. The ERC1155 balance tracking correctly updates to show Alice has one token and Bob has four tokens. However, the group transfer moves only three non-expired token groups from Alice to Bob, leaving the transfer incomplete. The group arrays now contain inconsistent data, where Alice's group array shows fewer tokens than her ERC1155 balance, and Bob's group array shows fewer tokens than his ERC1155 balance. Recommendations Short term, add a check at the end of the _transferGroups function to verify that debt equals zero, ensuring that the transfer was completed successfully. Long term, implement comprehensive unit tests that cover transfer scenarios with mixed valid and expired tokens, and consider redesigning the transfer logic to properly account for expired tokens during the transfer process.

HashEye 17 Radius Technology EVMAuth

PUBLIC Security Assessment

4. Missing debt validation in group burn function Severity: High Difficulty: Medium Type: Data Validation Finding ID: TOB-RADIUS-4 Target: src/base/EVMAuthExpiringERC1155.sol

Description The _burnGroupBalances function fails to validate that the burn operation completes successfully when expired tokens are present, potentially creating inconsistencies between ERC1155 balance tracking and group array tracking. The function skips over expired token groups during burning but does not verify that the burn was completed successfully. The _burnGroupBalances function iterates through the account's token groups in the FIFO order to burn the requested amount. However, it skips over expired token groups without accounting for them in the burn calculation. When the loop completes, there is no verification that the debt variable has reached zero, meaning that the burn may be incomplete. function _burnGroupBalances(address account, uint256

id, uint256 amount) internal { Group[] storage groups = _group[account][id]; uint256 _now = block.timestamp; uint256 debt = amount;

uint256 i = 0; while (i < groups.length && debt > 0) { if (groups[i].expiresAt ⩽ _now) { i++; continue; }

if (groups[i].balance > debt) { // Burn partial token group groups[i].balance -= debt; debt = 0; } else { // Burn entire token group debt -= groups[i].balance; groups[i].balance = 0; } i++; } }

HashEye 18 Radius Technology EVMAuth

PUBLIC Security Assessment

Figure 4.1: Missing debt validation in the _burnGroupBalances function Exploit Scenario Alice has five authentication tokens distributed across two groups: • Group A: Three tokens (expires at t=100, current time t=50) • Group B: Two tokens (expires at t=200, current time t=50) Time advances to t=150, making Group A expired. The token burner attempts to burn four of Alice's tokens. The ERC1155 balance tracking correctly updates to show that Alice has one token remaining. However, the group burn processes only two non-expired tokens from Group B, leaving the burn incomplete. The group arrays now contain inconsistent data, where Alice's group array shows fewer tokens than her ERC1155 balance. Recommendations Short term, add a check at the end of the _burnGroupBalances function to verify that debt equals zero, ensuring the burn was completed successfully. Long term, implement comprehensive unit tests that cover burn scenarios with mixed valid and expired tokens, and consider redesigning the burn logic to properly account for expired tokens during the burn process.

HashEye 19 Radius Technology EVMAuth

PUBLIC Security Assessment

5. Unbounded group array growth causes gas limit exceeded Severity: Medium Difficulty: Medium Type: Denial of Service Finding ID: TOB-RADIUS-5 Target: src/base/EVMAuthExpiringERC1155.sol

Description The contract's group array management system can cause gas limit issues when users have many token purchases with different expiration times. Each token purchase creates a new Group in the _group[account][tokenId] array, and functions that iterate through this array can exceed the block gas limit when the array becomes too large. The contract maintains a _group array for each account and token ID combination to track token batches with different expiration times. When users make frequent token purchases (e.g., one token every second), this creates a large array that must be iterated through during operations like balanceOf, _transferGroups, and _validGroups. Additionally, the _upsertGroup function performs expensive array insertion operations that shift elements to maintain expiration order. As the array grows, these operations consume increasing amounts of gas, eventually exceeding the block gas limit and making the contract unusable for affected users. function _upsertGroup(address account, uint256 id, uint256 amount, uint256 expiresAt) internal { Group[] storage groups = _group[account][id];

// Find the correct position to insert the group (ordered by expiration, oldest to newest) uint256 insertIndex = groups.length; for (uint256 i = 0; i < groups.length; i++) { // Check if this is an insert or an update if (groups[i].expiresAt > expiresAt) { // Insert the new token group at this position insertIndex = i; break; } else if (groups[i].expiresAt == expiresAt) { // If a token group with same expiration exists, combine the balances and return groups[i].balance += amount; return; } }

// If the new token group expires later than all the others, add it to the end

HashEye 20 Radius Technology EVMAuth

PUBLIC Security Assessment

of the array and return if (insertIndex == groups.length) { groups.push(Group({balance: amount, expiresAt: expiresAt})); return; }

// Shift array elements to make room for the new token group groups.push(Group({balance: 0, expiresAt: 0})); // Add space at the end for (uint256 i = groups.length - 1; i > insertIndex; i--) { groups[i] = groups[i - 1]; }

// Insert the new Group at the correct position groups[insertIndex] = Group({balance: amount, expiresAt: expiresAt}); } Figure 5.1: Gas-intensive array operations in _upsertGroup function Exploit Scenario Bob intentionally transfers small amounts of tokens with different expiration times to Alice’s address 2,000 times, creating 2,000 additional groups in her _group[alice] [tokenId] array. When Alice attempts to transfer her tokens, the _transferGroups function must iterate through all groups to find valid tokens, remove them from Alice’s group array, and add them to the recipient’s group array, consuming excessive gas. The transaction fails due to gas limit, preventing Alice from transferring, burning, or even checking her token balance. This effectively locks Alice’s tokens in the contract, making them unusable. Recommendations Short term, implement a maximum group array size limit and add pagination or batching mechanisms for operations that iterate through large group arrays. Long term, redesign the token expiration management system to use more efficient data structures that avoid linear iteration and reduce gas consumption for large token holdings.

HashEye 21 Radius Technology EVMAuth

PUBLIC Security Assessment

6. Inconsistent implementation between blacklist functions and batch versions Severity: Low Difficulty: High Type: Denial of Service Finding ID: TOB-RADIUS-6 Target: src/base/EVMAuthExpiringERC1155.sol

Description The blacklist management functions have inconsistent implementations between their single-account and batch versions. The batch functions lack important validations and event emissions that are present in their single-account counterparts, creating potential security and transparency issues. The addToBlacklist function includes validation to prevent blacklisting blacklist managers and zero addresses, while addBatchToBlacklist lacks these checks. Similarly, removeFromBlacklist uses delete operation and emits events, while removeBatchFromBlacklist sets the value to false and does not emit events. These inconsistencies can lead to unexpected behavior and reduced transparency in blacklist operations. function addToBlacklist(address account) external onlyRole(BLACKLIST_MANAGER_ROLE) { require(!hasRole(BLACKLIST_MANAGER_ROLE, account), "Account is a blacklist manager"); require(account ≠ address(0), "Account is the zero address");

_blacklisted[account] = true;

emit AddedToBlacklist(account); }

function addBatchToBlacklist(address[] memory accounts) external onlyRole(BLACKLIST_MANAGER_ROLE) { for (uint256 i = 0; i < accounts.length; i++) { _blacklisted[accounts[i]] = true; } } Figure 6.1: Inconsistent blacklist function implementations

HashEye 22 Radius Technology EVMAuth

PUBLIC Security Assessment

Exploit Scenario An admin uses addBatchToBlacklist to blacklist multiple accounts, including the zero address. The batch function will successfully blacklist the zero address without any validation, bypassing the security check present in the single-account version. This will break the mint and burn logic since the zero address is used internally for these operations. Recommendations Short term, update the batch functions to include the same validations and event emissions as their single-account counterparts, ensuring consistent behavior across all blacklist operations. Long term, implement comprehensive unit tests that cover all blacklist scenarios and consider creating a shared internal function to avoid code duplication and ensure consistency.

HashEye 23 Radius Technology EVMAuth

PUBLIC Security Assessment

7. TTL validation missing when updating burnable status Severity: Informational Difficulty: High Type: Configuration Finding ID: TOB-RADIUS-7 Target: src/base/EVMAuthExpiringERC1155.sol

Description The setBaseMetadata function allows updating the burnable status of a token without validating whether a TTL (time-to-live) has been set. This creates a potential inconsistency where a token can be marked as non-burnable while still having an expiration time configured. The contract enforces that TTL can be set only for burnable tokens in the setTTL function, but the reverse validation is missing. When updating a token’s metadata to make it non-burnable, the

function should check if a TTL is currently set and either prevent the change or reset the TTL to zero. Without this validation, tokens can exist in an inconsistent state where they have expiration times but cannot be burned. function setBaseMetadata(uint256 id, bool _active, bool _burnable, bool _transferable) public { require(hasRole(TOKEN_MANAGER_ROLE, _msgSender()), "Unauthorized token manager"); require(id ⩽ nextTokenId, "Invalid token ID");

_metadata[id] = BaseMetadata(id, _active, _burnable, _transferable);

if (id == nextTokenId) { nextTokenId++; } } Figure 7.1: Missing TTL validation in setBaseMetadata function Exploit Scenario An admin sets up an authentication token with a 30-day TTL and burnable status. Later, the admin decides to make the token non-burnable by calling setBaseMetadata with _burnable = false. The token now has an inconsistent state where it has an expiration time but cannot be burned. This could lead to confusion in the system logic where non-burnable tokens can be burnt through the cleanup mechanisms.

HashEye 24 Radius Technology EVMAuth

PUBLIC Security Assessment

Recommendations Short term, add validation in the setBaseMetadata function to check if a TTL is set when making a token non-burnable, and either prevent the change or automatically reset the TTL to zero. Long term, implement comprehensive unit tests that cover metadata update scenarios, and ensure consistency between the burnable status and TTL configuration.

HashEye 25 Radius Technology EVMAuth

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 26 Radius Technology EVMAuth

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 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 27 Radius Technology EVMAuth

PUBLIC Security Assessment

B. Code Maturity Categories The following tables describe the code maturity categories and rating criteria used in this document. Code Maturity Categories Category Description Arithmetic The proper use of mathematical operations and semantics Auditing The use of event auditing and logging to

support monitoring Authentication / Access Controls The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system Complexity Management The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions Cryptography and Key Management The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution Decentralization The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades Documentation The presence of comprehensive and readable codebase documentation Low-Level Manipulation The justified use of inline assembly and low-level calls Testing and Verification The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage Transaction Ordering The system's resistance to transaction-ordering attacks

HashEye 28 Radius Technology EVMAuth

PUBLIC Security Assessment

Rating Criteria Rating Description Strong No issues were found, and the system exceeds industry standards. Satisfactory Minor issues were found, but the system is compliant with best practices. Moderate Some issues that may affect system safety were found. Weak Many issues that affect system safety were found. Missing A required component is missing, significantly affecting system safety. Not Applicable The category does not apply to this review. Not Considered The category was not considered in this review. Further Investigation Required Further investigation is required to reach a meaningful conclusion.

HashEye 29 Radius Technology EVMAuth

PUBLIC Security Assessment

C. Code Quality Recommendations The following recommendations are not associated with specific vulnerabilities. However, implementing them can enhance the code's readability and may prevent the introduction of vulnerabilities in the future. EVMAuth 1. Remove the duplicated authorization check to save gas and simplify logic. The setMetadata function currently requires TOKEN_MANAGER_ROLE before calling setBaseMetadata, which already enforces the same role. This double-check increases gas usage and risks policy drift if the two checks ever diverge. EVMAuthPurchasableERC1155 2. Remove the redundant withdrawal function. The purchase flow forwards proceeds to wallet immediately, so the contract should not normally hold ETH. Keeping the redundant withdraw function adds maintenance overhead without clear benefit.

HashEye 30 Radius Technology EVMAuth

PUBLIC Security Assessment

D. 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 September 22, 2025, HashEye reviewed the fixes and mitigations implemented by the Radius Technology team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue. In summary, Radius Technology has resolved all seven issues. For additional information, please see the Detailed Fix Review Results below.

ID Title Severity Status 1 Incorrect account assignment in token burning logic High Resolved 2 Expired token groups not synchronized with ERC1155 balance tracking High Resolved 3 Missing debt validation in group transfer function High Resolved 4 Missing debt validation in group burn function High Resolved 5 Unbounded group array growth causes gas limit exceeded Medium Resolved 6 Inconsistent implementation between blacklist functions and batch versions Low Resolved 7 TTL validation missing when updating burnable status Informational Resolved

HashEye 31 Radius Technology EVMAuth

PUBLIC Security Assessment

Detailed Fix Review Results

TOB-RADIUS-1: Incorrect account assignment in token burning logic Resolved in commit 54a99e3. The team corrected the account address in the token burning logic. TOB-RADIUS-2: Expired token groups not synchronized with ERC1155 balance tracking Resolved in commit 54a99e3. The team added a _burnPrunedTokens hook to burn expired tokens when the pruneBalanceRecords function is called. TOB-RADIUS-3: Missing debt validation in group transfer function Resolved in commit 54a99e3. The team added a debt validation at the end of the _transferBalanceRecords function that reverts the transaction if debt is non-zero. TOB-RADIUS-4: Missing debt validation in group burn function Resolved in commit 54a99e3. The team added a debt validation at the end of the _deductFromBalanceRecords function that reverts the transaction if debt is non-zero. TOB-RADIUS-5: Unbounded group array growth causes gas limit exceeded Resolved in commit 54a99e3. The team caps the balance record array's length at 100 and groups the expiration time by buckets that cover the TTL duration. TOB-RADIUS-6: Inconsistent implementation between blacklist functions and batch versions Resolved in commit 54a99e3. The team removed the batch blacklist functionality. TOB-RADIUS-7: TTL validation missing when updating burnable status Resolved in commit 54a99e3. The team removed the burnable configuration. The token is now burnable by default unless the TTL value is configured to 0.

HashEye 32 Radius Technology EVMAuth

PUBLIC Security Assessment

E. Fix Review Targets The fix review involved reviewing the following target: evmauth-core

Repository github.com/evmauth/evmauth-core Version 54a99e37f154844c6c4140f3ad1ed4ad14d7b474 Type Solidity Platform EVM

HashEye 33 Radius Technology EVMAuth

PUBLIC Security Assessment

F. 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 34 Radius Technology EVMAuth

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 35 Radius Technology EVMAuth

PUBLIC Security Assessment

Notices and Remarks Copyright and Distribution © 2025 by HashEye, Inc. All rights reserved. HashEye hereby asserts its right to be identified as the creator of this report in the United Kingdom.

HashEye considers this report public information; it is licensed to Radius Technology under the terms of the project statement of work and has been made public at Radius Technology's request. Material within this report may not be reproduced or distributed in part or in whole without HashEye' express written permission. The sole canonical source for HashEye publications is the HashEye Publications page. Reports accessed through sources other than that page may have been modified and should not be considered authentic. 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 36 Radius Technology EVMAuth

PUBLIC Security Assessment