HashEye

// SMART CONTRACT SECURITY AUDIT

Open Home Foundation SecureTar v3

Security assessment by HashEye · prepared for Open Home Foundation

HASHEYE AUDITED

PROJECT	Open Home Foundation SecureTar v3
PERFORMED BY	HashEye Security Review Team - Sergey, Brian
RESEARCH	Sergey, Brian
CLIENT	Open Home Foundation
CATEGORY	Blockchain
PUBLISHED	March 1, 2026
REPORT ID	research-open-home-foundation-securetar-v3-2026-03-01-ukv4ez

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-open-home-foundation-securetar-v3-2026-03-01-ukv4ez.

Open Home Foundation SecureTar v3 Security Assessment March 2, 2026

Prepared for: Erik Montnemery Open Home Foundation

Prepared by: Scott Arciszewski

HashEye

PUBLIC

Table of Contents Table of Contents 1 Project Summary 2 Executive Summary 3 Project Goals 5 Project Targets 6 Project Coverage 7 Summary of Findings 8 Detailed Findings 9 1. Timing side channel in validation key comparison 9 2. Insecure fallback to legacy protocol version 11 3. Supply-chain attacks are possible with GitHub Actions workflows 13 A. Vulnerability Categories 14 B. Static Analysis Setup 16 C. Code Quality Recommendations 17 D. Fix Review Results 18 Detailed Fix Review Results 18 E. Fix Review Status Categories 19 About HashEye 20 Notices and Remarks 21

HashEye 1 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

Project Summary Contact Information The following project manager was associated with this project: Emily Doucette, Project Manager emily.doucette@hasheye.io The following engineering director was associated with this project: Jim Miller, Engineering Director, Cryptography james.miller@hasheye.io The following consultant was associated with this project: Scott Arciszewski, Consultant scott.arciszewski@hasheye.io Project Timeline The significant events and milestones of the project are listed below. Date Event January 29, 2026 Pre-project kickoff call February 9, 2026 Delivery of report draft February 9, 2026 Report readout meeting February 18, 2026 Completion of fix review March 2, 2026 Delivery of final comprehensive report

HashEye 2 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

Executive Summary Engagement Overview Open Home Foundation engaged HashEye to review the security of SecureTar v3. SecureTar is a Python library for creating and reading password-protected tar archives, used by Home Assistant to create encrypted backups. SecureTar v3 uses best-in-class cryptographic primitives (e.g., Argon2id for password-based key derivation), provided by the Python binding of the libsodium cryptography library (PyNaCl). Encryption is provided through the secretstream API, which implements Rogaway's STREAM construction with XChaCha20-Poly1305. One consultant conducted the review from February 2 to February 6, 2026, for a total of one engineer-week of effort. Our testing efforts focused on identifying any cryptographic weaknesses in SecureTar v3. With full access to the source code and documentation, we conducted static and dynamic testing of SecureTar v3, using both automated and manual processes. Observations and Impact

The review identified only three issues: one medium severity and two informational severity. The most significant consideration for SecureTar users is the continued support for legacy formats (v1, v2), which rely on weak key derivation and lack resistance to chosen-ciphertext attacks (TOB-SECURETAR-2). Also, GitHub Actions are not pinned to commit hashes and have overly broad permissions, opening the system up to supply-chain attacks (TOB-SECURETAR-3). February 18, 2026, we reviewed the fixes and mitigations implemented by the Open Home Foundation team. We found that all issues have been resolved (appendix D). Recommendations Based on the three findings identified during the security review, HashEye recommends that Open Home Foundation take the following steps: • Remediate the findings disclosed in this report. These findings should be addressed through direct fixes or broader refactoring efforts. • Adopt robust testing methodologies. SecureTar v3 is a good candidate for property-based testing, mutation testing, and fuzz testing. All three can be done in Python as part of a CI/CD pipeline. • Change the default version to v3. The current source

code defaults to v2, while the new version offers a significant security improvement. As soon as reasonably possible, v3 should be the new default version.

HashEye 3 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

Finding Severities and Categories

The following tables provide the number of findings by severity and category. EXPOSURE ANALYSIS Severity Count High 0 Medium 1 Low 0 Informational 2 Undetermined 0

CATEGORY BREAKDOWN Category Count Configuration 1 Cryptography 2

HashEye 4 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

Project Goals The engagement was scoped to provide a security assessment of the SecureTar v3 library. Specifically, we sought to answer the following non-exhaustive list of questions: • What cryptographic primitives does the SecureTar library currently use for encryption and key derivation? • What authentication mechanisms exist to verify the integrity of encrypted tar files? • How does SecureTar protect against tampering with or manipulation of encrypted archives? • Are there any known side-channel vulnerabilities in the current implementation? • How does SecureTar ensure that encryption remains effective across different Python versions and environments? • Is the current implementation vulnerable to padding oracle attacks or similar cryptographic weaknesses? • How does SecureTar handle error conditions during encryption or decryption operations? • What protections exist against key extraction through memory analysis? • How are secure random numbers generated and used within the SecureTar library? • Does the current implementation follow modern cryptographic best practices regarding key lengths and algorithm choices? • What performance considerations must be balanced against security improvements in the SecureTar redesign?

HashEye 5 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

Project Targets The engagement involved reviewing and testing the following target. SecureTar v3

Repository https://github.com/home-assistant-libs/securetar Version 2bd142637bb1001893d6b7e42b82a4646b5f5f83 Type Library Platform Python

HashEye 6 Open Home Foundation SecureTar v3

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: • Static analysis with Semgrep • Manual review of the SecureTar v3 library with a focus on its use of cryptography • AI-assisted analysis with internal and public Claude Skills and related tooling, including HashEye’ constant-time-analysis skill Coverage Limitations Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review: • Third-party dependencies • Implementation-specific behavior of Python runtime libraries

HashEye 7 Open Home Foundation SecureTar v3

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 Timing side channel in validation key comparison Cryptography Informational 2 Insecure fallback to legacy protocol version Cryptography

Informational 3 Supply-chain attacks are possible with GitHub Actions workflows Configuration Medium

HashEye 8 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

Detailed Findings 1. Timing side channel in validation key comparison Severity: Informational Difficulty: Not Applicable Type: Cryptography Finding ID: TOB-SECURETAR-1 Target: KeyDerivationV3

Description The validation key, derived from a memory-hard password hashing function, is stored in the header and later used to verify that the correct password was provided. This verification uses the ≠ operator (figure 1.1). if validation_key ≠ stored_validation_key: raise InvalidPasswordError("Invalid password") Figure 1.1: securetar/__init__.py#L1531-L1532

When the outputs of cryptographic functions (e.g., hash functions or password-based cryptography) are compared using standard byte- or string-comparison operators, the underlying comparison function will often abort early as soon as a mismatch is detected. (In glibc, comparisons are done word by word; in some runtimes, comparisons are done byte by byte.) If you can measure how long it takes the comparison to fail, you can learn information about what was stored in the first place. Typically, an attacker would need many queries in order to measure the timing difference between different inputs. There are two mitigating factors that prevent this from being a security issue: 1. The values being compared are the outputs of a memory-hard password-hashing algorithm (Argon2id), which means each query will be measured against the timing jitter caused by the GPU-hard iterations of Argon2. 2. The value an attacker might hope to leak from such an attack is stored in the header, unencrypted, thus rendering this kind of cryptanalysis pointless. This is the sort of code smell that security-focused static analysis tools and amateur security researchers catch quickly. Fixing it preemptively will reduce the noise from false positives.

HashEye 9 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

Recommendations Short term, replace the comparison operation with the negation of hmac.compare_digest(). Long term, use security-focused static analysis tooling (e.g., Semgrep, CodeQL) in your CI pipelines to catch these issues during development.

HashEye 10 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

2. Insecure fallback to legacy protocol version Severity: Informational Difficulty: Not Applicable Type: Cryptography Finding ID: TOB-SECURETAR-2 Target: SecureTarHeader

Description The from_bytes method, called on SecureTarHeader, checks three conditions, joined by an or operation (which short-circuits if any are true), and assumes the header is configured for the legacy version without magic (figure 2.1). magic, version, reserved = struct.unpack(SECURETAR_FILE_ID_FORMAT, header) if (magic ≠ SECURETAR_MAGIC or version not in (2, 3) or reserved ≠ SECURETAR_MAGIC_RESERVED): # Assume legacy format without magic cipher_initialization = header version = 1 else: Figure 2.1: securetar/__init__.py#L159-L167

This legacy fallback has the unintended consequence that a single corruption could cause incorrect behavior rather than raising an exception. Automated scripts that use SecureTar might proceed with processing invalid data rather than recognizing a decryption failure. Recommendations Short term, modify the logic so that the magic value is the deciding factor on v1 support, as shown in figure 2.2. if magic == SECURETAR_MAGIC: # Magic matches → this IS a SecureTar file, not legacy if version not in (2, 3): raise ValueError(f"Unsupported SecureTar version: {version}") if reserved ≠ SECURETAR_MAGIC_RESERVED: raise ValueError("Corrupted header: invalid reserved bytes") # ... continue v2/v3 parsing else: # No magic → genuine v1 legacy file cipher_initialization = header version = 1 Figure 2.2: Proposed fix

HashEye 11 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

Long term, remove legacy fallback support once it is no longer needed.

HashEye 12 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

3. Supply-chain attacks are possible with GitHub Actions workflows Severity: Medium Difficulty: High Type: Configuration Finding ID: TOB-SECURETAR-3 Target: .github/workflows/*.yml

Description The GitHub Actions workflows do not pin actions to commit hashes. Additionally, they allow overly broad permissions. For example, see figure 3.1. steps: - uses: actions/checkout@v6 Figure 3.1: . github/workflows/pythonpublish.yml#L14-L15

If a malicious new version of actions/checkout is tagged, every build and Python release could be vulnerable to arbitrary code execution in the build process, which can enable the publication of malware or GitHub token theft. Exploit Scenario Mallory hacks ericsciple (the GitHub user that owns actions/checkout) and publishes v6.0.3 of actions/checkout with a targeted attack for securetar-v3. Her malware modifies the Python code to steal users' passwords right before the next release. As a result, this unauthorized modification is included in the official release. Recommendations Short term, pin all actions to specific hashes. Long term, use zizmor to analyze your workflow configurations.

HashEye 13 Open Home Foundation SecureTar v3

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 14 Open Home Foundation SecureTar v3

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 15 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

B. Static Analysis Setup This appendix describes the setup of Semgrep, the static analysis tool we used during this audit. After installing Semgrep and creating a directory of rules, we ran the following command on the codebase: semgrep scan --config "p/hasheye" --config "p/cwe-top-25" \ --

config "p/python" --config "r/all" --config "p/default" --sarif \ --sarif-output=semgrep.sarif We used a mix of public and private rules and uncovered the two reported informational-severity issues, along with several false positives.

HashEye 16 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

C. Code Quality Recommendations This appendix contains recommendations 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. • Test the unhappy path for SecureTarHeader.from_bytes. We set up a mutation testing framework using mutmut and discovered that all mutants to this code escaped the test suite. • Pass the O_EXCL flag in _open_file. SecureTar supports mode="x" (exclusive create) in its public API (MOD_EXCLUSIVE = "x" at line 114), but _open_file does not distinguish between "w" and "x"; both use O_WRONLY | O_CREAT without O_EXCL. This means mode="x" silently overwrites existing files instead of failing. Additionally, O_TRUNC is missing, so partial writes to existing files leave trailing garbage from the old content. • Bump DEFAULT_CIPHER_VERSION to 3. The improved cryptography should be adopted as soon as possible. • Document bufsize=0 behavior. The parameter is passed to tarfile, which interprets 0 as "use the default buffer size (10240)", but this is not documented and is implementation-dependent. • Address the use of bytes arguments. Multiple API inputs expect bytes arguments. This increases cognitive load and makes the APIs error-prone. Instead, we recommend using NewType or data classes, as shown in figure C.1. from typing import NewType

RootKey = NewType('RootKey', bytes) Salt = NewType('Salt', bytes) ValidationKey = NewType('ValidationKey', bytes)

class KeyDerivationV3(KeyDerivationStrategy): """Key derivation for SecureTar v3."""

stream_factory = SecretStreamFactory

def __init__(self, root_key: RootKey, root_salt: Salt, validation_salt: Salt, validation_key: ValidationKey,): Figure C.1: Updated KeyDerivation init API that uses distinct input types

HashEye 17 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

D. Fix Review Results During a fix review, HashEye assesses 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 a comprehensive analysis of the system. On February 18, 2026, HashEye reviewed the fixes and mitigations implemented by the Open Home Foundation team for the issues identified in this report. In summary, Open Home Foundation has fixed all the reported issues. ID Title Severity Status 1 Timing side channel in validation key comparison Informational Resolved 2 Insecure fallback to legacy protocol version Informational Resolved 3 Supply-chain attacks are possible with GitHub Actions workflows Medium Resolved Detailed Fix Review Results TOB-SECURETAR-1: Timing side channel in validation key comparison Resolved in the patch provided, which follows our recommendation to use hmac.compare_digest to avoid timing side channels. TOB-SECURETAR-2: Insecure fallback to legacy protocol version Resolved in the patch provided, which implements our short-term recommendation. TOB-SECURETAR-3: Supply-chain attacks are possible with GitHub Actions workflows Resolved in the patch provided, which specifies the Git commit SHA-1 hash for each action dependency.

HashEye 18 Open Home Foundation SecureTar v3

PUBLIC Security Assessment

E. 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 19 Open Home Foundation SecureTar v3

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 20 Open Home Foundation SecureTar v3

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 21 Open Home Foundation SecureTar v3

PUBLIC Security Assessment