

wasmCloud

Security assessment by HashEye · prepared for Open Source Technology Improvement Fund

HASHEYE AUDITED

PROJECT	wasmCloud
CLIENT	Open Source Technology Improvement Fund
CATEGORY	Blockchain
PUBLISHED	September 1, 2023
REPORT ID	research-wasmcloud-2023-09-01-dxafdy

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 hashey.io/audits/research-wasmcloud-2023-09-01-dxafdy.

wasmCloud Security Assessment October 13, 2023 Prepared for: wasmCloud Open Source Technology Improvement Fund Prepared by: Francesco Bertolaccini, Artur Cygan, Spencer Michaels

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

Notices and Remarks Copyright and Distribution © 2023 by HashEye, Inc. All rights reserved. HashEye hereby asserts its right to be identified as the creator of this report in the United Kingdom. This report is considered by HashEye to be public information; it is licensed to wasmCloud under the terms of the project statement of work and has been made public at wasmCloud's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of HashEye. The sole canonical source for HashEye publications is the HashEye Publications page . Reports accessed through any source other than that page may have been modified and should not be considered authentic. 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. HashEye 2 wasmCloud Security Assessment PUBLIC

Table of Contents About HashEye 1 Notices and Remarks 2 Table of Contents 3 Project Summary 4 Executive Summary 5 Project Goals 7 Project Targets 8 Project Coverage 9 Codebase Maturity Evaluation 10 Summary of Findings 12 Detailed Findings 13 1. Out-of-bounds crash in extract_claims 13 2. Stack overflow while enumerating containers in blobstore-fs 15 3. Denial of service in blobstore-s3 using malicious actor 17 4. Unexpected panic in validate_token 18 5. Incorrect error message when starting actor from file 19 A. Vulnerability Categories 20 B. Code Maturity Categories 22 C. Fix Review Results 24 Detailed Fix Review Results 25 D. Fix Review Status Categories 26 HashEye 3 wasmCloud Security Assessment PUBLIC

Project Summary Contact Information The following managers were associated with this project: Dan Guido , Account Manager jeff.braswell@hasheye.io jeff .braswell@hasheye.io The following engineers were associated with this project: Francesco Bertolaccini , Consultant francesco.bertolaccini@hasheye.io artur.cygan@hasheye.io Spencer Michaels , Consultant spencer.michaels@hasheye.io Project Timeline The significant events and milestones of the project are listed below. Date Event August 18, 2023 Pre-project kickoff call August 28, 2023 Status update meeting #1 September 13, 2023 Delivery of report draft September 13, 2023 Report readout meeting October 13, 2023 Delivery of comprehensive report HashEye 4 wasmCloud Security Assessment PUBLIC

Executive Summary Engagement Overview The Open Source Technology Foundation engaged HashEye to review the security of wasmCloud, a runtime and deployment platform for distributed WASM application development. A team of three consultants conducted the review from August 21 to September 1, for a total of six engineer-weeks of effort. Our testing efforts focused on reviewing critical components of the wasmCloud platform, with a particular emphasis on capability providers, such as Rust micro-applications that proxy access to external services (like databases) and HTTP servers. We supplemented our code review with fuzzing wherever feasible, except in the case of the WASM runtime itself, which has already undergone substantial fuzz testing and thus was not prioritized for fuzzing in this audit. With full access to source code and documentation, we performed static and dynamic testing of numerous wasmCloud components, using automated and manual processes. In cases where the codebase diverged into two major versions (notably due to a Rust reimplementation of legacy Elixir components which is still in development), we focused on the stable legacy code, with which the newer code will be backwards compatible. Observations and Impact wasmCloud's capability providers are generally implemented using widely used, well-vetted third-party libraries to interact with the services they are backing. Potentially error-prone operations, such as string transformation, are relatively rare. No issues were discovered in wasmCloud's use of JWTs for authentication; tokens were appropriately validated for all observed authenticated endpoints. The wasmCloud OTP host uses native Rust code, where we found a user-triggerable crash; however, this issue is mitigated by the Rustler library's error handling. Otherwise, the OTP host appears to comply with Erlang/OTP best practices. Recommendations Based on the codebase maturity evaluation and findings identified during the security review, HashEye recommends that the wasmCloud team take the following steps. • 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. • Ensure that documentation is kept up with the pace of development. Document new functionality as it is implemented, especially provider settings that HashEye 5 wasmCloud Security Assessment PUBLIC

are configurable by users, and record any caveats regarding components contributed by third parties (e.g. the SQL capability providers). Finding Severities and Categories The following tables provide the number of findings by severity and category. EXPOSURE ANALYSIS Severity Count High 0 Medium 0 Low 2 Informational 2 Undetermined 1 CATEGORY BREAKDOWN Category Count Data Validation 3 Error Reporting 2 HashEye 6 wasmCloud Security Assessment PUBLIC

Project Goals The engagement was scoped to provide a security assessment of the wasmCloud platform. Specifically, we sought to answer the following non-exhaustive list of questions: • Does the wasmCloud runtime appropriately sandbox user-provided code? • Do the wasmCloud capability providers limit actors' access to only the intended capabilities? Are there breakouts or loopholes that can circumvent these intended limitations? • Can an attacker with full or partial control over the NATS agent use it to attack wasmCloud actors or capability providers? • Can RPC messages between actors and capability providers be spoofed or modified without detection? • Is it possible to deny service to the host or to take control of the execution environment? HashEye 7 wasmCloud Security Assessment PUBLIC

Project Targets The engagement involved a review and testing of the targets listed below. wasmCloud Repository <https://github.com/wasmCloud/wasmCloud> Version 33ef4f34a5748e445f01148ec7d00bb0f01c1606 Type Rust Platform Native wasmCloud-otp Repository <https://github.com/wasmCloud/wasmCloud-otp> Version 1e9076ae8786168c23e7c28003e2212689d10948 Type Elixir, Rust Platform BEAM, Native wascap Repository <https://github.com/wasmCloud/wascap> Version a1299cc722a122cbde590047bcad9d3edb57d6c2 Type Rust Platform Native capability-providers Repository <https://github.com/wasmCloud/capability-providers> Version 8446e10f93badf7db0a961e595143f3c42a3a6c8 Type Rust Platform Native nats-server Repository <https://github.com/nats-io/nats-server> Version d720a6931c71a83aa8df8715b7dc0f87d5b0f527 Type Go Platform Native HashEye 8 wasmCloud Security Assessment PUBLIC

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: • Manual code review of all extant wasmCloud capability providers • Fuzzing select critical functionality within the wasmCloud codebase • Review of RPC message signing and integrity-checking code • Review of the OTP host Elixir and Rust code 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: • Due to time constraints, our manual code review and fuzzing of the wasmCloud core codebase was not exhaustive, being necessarily limited to only certain subsets of functionality. Most notably, fuzzing was limited to the protocol parsing routines in nats-server, while the entirety of the capability providers was manually reviewed. • Engineers were unable to fuzz the capability providers extensively over their RPC interfaces, as this proved too cumbersome in the limited time available given the infrastructure setup required. In these cases, we instead focused on manual

code review of the providers, and were able to achieve full coverage. HashEye 9 wasmCloud Security Assessment PUBLIC

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 wasmCloud does not perform critical arithmetic operations. Not Applicable Auditing The OTP host extensively uses the standard Elixir Logger to log any important information. We found a minor error in one of the log messages. Satisfactory Authentication / Access Controls wasmCloud's capability provider model ensures that applications can invoke only functionality that they are explicitly authorized to provide. Strong Complexity Management The wasmCloud codebase is generally well-organized, divided by functionality across a variety of distinct repositories and crates. Satisfactory Configuration wasmCloud's capability providers interact with their respective backing services via well-known, widely used libraries. Engineers did not note any issues in the providers' use of the relevant third-party APIs. Satisfactory Cryptography and Key Management RPC messages between actors and capability providers are appropriately validated and protected from spoofing. Satisfactory Data Handling RPC message data is safely handled and transformed into corresponding calls to the relevant back-end services. Satisfactory Documentation wasmCloud's high-level documentation is generally comprehensive, but documentation pertaining to detailed features (such as capability provider settings) or Moderate HashEye 10 wasmCloud Security Assessment PUBLIC

recently implemented functionality is lacking. In addition, code comments are generally sparse. Memory Safety and Error Handling With a handful of simple, one-line exceptions, no unsafe code is used anywhere in the wasmCloud codebase. Errors are handled appropriately using Rust's native error semantics. Satisfactory Testing and Verification Unit tests exist for most major functionality, although there are no fuzz tests for external input handling (such as in capability providers). Moderate HashEye 11 wasmCloud 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 Out-of-bounds crash in extract_claims Data Validation Low 2 Stack overflow while enumerating containers in blobstore-fs Data Validation Low 3 Denial of service in blobstore-s3 using malicious actor Data Validation Undetermined 4 Unexpected panic in validate_token Error Reporting Informational 5 Incorrect error message when starting actor from file Error Reporting Informational HashEye 12 wasmCloud Security Assessment PUBLIC

Detailed Findings 1. Out-of-bounds crash in extract_claims Severity: Low Difficulty: Low Type: Data Validation Finding ID: TOB-WACL-1 Target: wascap/src/wasm.rs Description The strip_custom_section function does not sufficiently validate data and crashes when the range is not within the buffer (figure 1.1). The function is used in the extract_claims function and is given an untrusted input. In the wasmCloud-otp, even though extract_claims is called as an Erlang NIF (Native Implemented Function) and potentially could bring down the VM upon crashing, the panic is handled gracefully by the Rustler library, resulting in an isolated crash of the Elixir process. if let Some ((id, range)) = payload.as_section() { wasm_encoder::RawSection { id, data: & buf [range], } .append_to(& mut output); } Figure 1.1: wascap/src/wasm.rs#L161-L167 We found this issue by fuzzing the extract_claims function with cargo-fuzz (figure 2.1). #![no_main] use libfuzzer_sys::fuzz_target; use getrandom::register_custom_getrandom; // TODO: the program won't compile without this, why? fn custom_getrandom (buf: & mut [u8]) → Result <(), getrandom::Error> { return Ok (()); } register_custom_getrandom!(custom_getrandom); fuzz_target!(|data: & [u8]| { HashEye 13 wasmCloud Security Assessment PUBLIC

let _ = wascap::wasm::extract_claims(data); }); Figure 1.2: A simple extract_claims fuzzing harness that passes the fuzzer-provided bytes straight to the function After fixing the issue (figure 1.3), we fuzzed the function for an extended period of time; however, we found no additional issues. if let Some ((id, range)) = payload.as_section() { if range.end ≤ buf.len() { wasm_encoder::RawSection { id, data: & buf [range], } .append_to(& mut output); } else { return Err (errors::new(ErrorKind::InvalidCapability)); } } Figure 1.3: The fix we applied to continue fuzzing extract_claims. The code requires a new error value because we reused one of the existing ones that likely does not match the semantics. Exploit Scenario An attacker deploys a new module with invalid claims. While decoding the claims, the extract_claims function panics and crashes the Elixir process. Recommendations Short term, fix the strip_custom_section function by adding the range check, as shown in the figure 1.3. Add the extract_claims fuzzing harness to the wascap repository and run it for an extended period of time before each release of the library. Long term, add a fuzzing harness for each Rust function that processes user-provided data. References • Erlang - NIFs HashEye 14 wasmCloud Security Assessment PUBLIC

2. Stack overflow while enumerating containers in blobstore-fs Severity: Low Difficulty: High Type: Data Validation Finding ID: TOB-WACL-2 Target: capability-providers/blobstore-fs/src/fs_utils.rs Description The all_dirs function is vulnerable to a stack overflow caused by unbounded recursion, triggered by either the presence of circular symlinks inside the root of the blobstore (as configured during startup), or the presence of excessively nested directory inside the same. Because this function is used by FsProvider::list_containers, this issue would result in a denial of service for all actors that use the method exposed by affected blobstores. let mut subdirs: Vec <PathBuf> = Vec ::new(); for dir in &dirs { let mut local_subdirs = all_dirs(prefix.join(dir.as_path()).as_path(), prefix); subdirs.append(&mut local_subdirs); } dirs.append(&mut subdirs); dirs Figure 2.1: capability-providers/blobstore-fs/src/fs_utils.rs#L24-L30 Exploit Scenario An attacker creates a circular symlink inside the storage directory. Alternatively, an attacker can—under the right circumstances—create successively nested directories with a sufficient depth to cause a stack overflow. blobstore.create_container(ctx, &"a".to_string()). await ?; blobstore.create_container(ctx, &"a/a".to_string()). await ?; blobstore.create_container(ctx, &"a/a/a".to_string()). await ?; ... blobstore.create_container(ctx, &"a/a/a/.../a/a/a".to_string()). await ?; blobstore.list_containers(). await ?; Figure 2.2: Possible attack to a vulnerable blobstore In practice, this attack requires the underlying file system to allow exceptionally long filenames, and we have not been able to produce a working attack payload. However, this does not prove that no such file systems exist or will exist in the future. HashEye 15 wasmCloud Security Assessment PUBLIC

Recommendations Short term, limit the amount of allowable recursion depth to ensure that no stack overflow attack is possible given realistic stack sizes, as shown in figure 2.3. pub fn all_dirs(root: &Path, prefix: &Path, depth: i32) → Vec <PathBuf> { if depth > 1000 { return vec! []; } ... // Now recursively go in all directories and collect all sub-directories let mut subdirs: Vec <PathBuf> = Vec ::new(); for dir in &dirs { let mut local_subdirs = all_dirs(prefix.join(dir.as_path()).as_path(), prefix, depth + 1); subdirs.append(&mut local_subdirs); } dirs.append(&mut subdirs); dirs } Figure 2.3: Limiting the amount of allowable recursion depth Long term, consider limiting the reliance on the underlying file system to a minimum by disallowing nesting containers. For example, Base64-encode all container and object names before passing them down to the file system routines. References • OWASP Denial of Service Cheat Sheet ("Input validation" section) HashEye 16 wasmCloud Security Assessment PUBLIC

3. Denial of service in blobstore-s3 using malicious actor Severity: Undetermined Difficulty: High Type: Data Validation Finding ID: TOB-WACL-3 Target: capability-providers/blobstore-s3/src/lib.rs Description The stream_bytes function continues looping until it detects that all of the available bytes have been sent. It does this based on the output of the send_chunk function, which reports the amount of bytes that have been sent by the call. An attacker could send specially crafted responses that cause stream_bytes to continue looping, causing send_chunk to report that no errors were detected while also reporting that no bytes were sent. while bytes_sent < bytes_to_send { let chunk_offset = offset + bytes_sent; let chunk_len = (self.max_chunk_size() as u64).min(bytes_to_send - bytes_sent); bytes_sent += self .send_chunk (ctx, Chunk { is_last: offset + chunk_len > end_range, bytes: bytes[bytes_sent as usize..(bytes_sent + chunk_len) as usize] .to_vec(), offset: chunk_offset as u64, container_id: bucket_id.to_string(), object_id: cobj.object_id.clone(), },) .await?; } Figure 3.1: capability-providers/blobstore-s3/src/lib.rs#L188-L204 Exploit Scenario An attacker can send a maliciously crafted request to get an object from a blobstore-s3 provider, then send successful responses without making actual progress in the transfer by reporting that empty-sized chunks were received. Recommendations Make send_chunk report a failure if a zero-sized response is received. HashEye 17 wasmCloud Security Assessment PUBLIC

4. Unexpected panic in validate_token Severity: Informational Difficulty: High Type: Error Reporting Finding ID: TOB-WACL-4 Target: wascap/src/jwt.rs Description The validate_token function from the wascap library panics with an out-of-bounds error when input is given in an unexpected format. The function expects the input to be a valid JWT token with three segments separated by a dot (figure 4.1). This implicit assumption is satisfied in the code; however, the function is public and does not mention the assumption in its documentation. /// Validates a signed JWT. This will check the signature, expiration time, and not-valid-before time pub fn validate_token <T> (input: &str) → Result <TokenValidation> where T: Serialize + DeserializeOwned + WascapEntity, { let segments: Vec <& str > = input.split('.').collect(); let header_and_claims = format! ("{}.{}" , segments[0] , segments[1]); let sig = base64::decode_config(segments[2] , base64::URL_SAFE_NO_PAD)?; .. } Figure 4.1: wascap/src/jwt.rs#L612-L641 Exploit Scenario A developer uses the validate_token function expecting it to fully validate the token string. The function receives an untrusted malicious input that forces the program to panic. Recommendations Short term, add input format validation before accessing the segments and a test case with malformed input. Long term, always validate all inputs to functions or document the input

assumptions if validation is not in place for a specific reason. HashEye 18 wasmCloud Security Assessment PUBLIC

5. Incorrect error message when starting actor from file Severity: Informational Difficulty: Low Type: Error Reporting Finding ID: TOB-WACL-5 Target: host_core/lib/host_core/actors/actor_supervisor.ex Description The error message when starting an actor from a file contains a string interpolation bug that causes the message to not include the fileref content (figure 5.1). This causes the error message to contain the literal string ``${fileref}`` instead. It is worth noting that the fileref content will be included anyway as an attribute. Logger `.error("Failed to read actor file from `${fileref}` : #{ inspect(err) } " , fileref : fileref)` Figure 5.1: host_core/lib/host_core/actors/actor_supervisor.ex#L301 Recommendations Short term, change the error message to correctly interpolate the fileref string. HashEye 19 wasmCloud 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 Undefined behavior triggered within the system HashEye 20 wasmCloud 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 21 wasmCloud Security Assessment PUBLIC

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 Configuration The configuration of system components in accordance with best practices Cryptography and Key Management The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution Data Handling The safe handling of user inputs and data processed by the system Documentation The presence of comprehensive and readable codebase documentation Memory Safety and Error Handling The presence of memory safety and robust error-handling mechanisms Testing and Verification The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage 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. HashEye 22 wasmCloud Security Assessment PUBLIC

Missing A required component is missing, significantly affecting system safety. Not Applicable The category is not applicable 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 23 wasmCloud 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 October 4, 2023, HashEye reviewed the fixes and mitigations implemented by the wasmCloud team for the issues

identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue. In summary, wasmCloud has resolved all identified issues. For additional information, please see the Detailed Fix Review Results below. ID Title Status 1 Out of bounds crash in extract_claims Resolved 2 Stack overflow while enumerating containers in blobstore-fs Resolved 3 Denial of Service in blobstore-s3 using malicious actor Resolved 4 Unexpected panic in validate_token Resolved 5 Incorrect error message when starting actor from file Resolved HashEye 24 wasmCloud Security Assessment PUBLIC

Detailed Fix Review Results TOB-WACL-1: Out of bounds crash in extract_claims Resolved in commit 664d9b9 . The missing range validation was added. TOB-WACL-2: Stack overflow while enumerating containers in blobstore-fs Resolved in PR capability-providers/271 . The fix limits the recursion to a maximum of 1,000 calls. TOB-WACL-3: Denial of Service in blobstore-s3 using malicious actor Resolved in PR capability-providers/271 . The missing response emptiness check was added. TOB-WACL-4: Unexpected panic in validate_token Resolved in PR wascap/52 . The missing segments quantity validation was added. TOB-WACL-5: Incorrect error message when starting actor from file Resolved in PR wasmcloud-otp/648 . The mistake in message log string interpolation was fixed. HashEye 25 wasmCloud Security Assessment PUBLIC

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 26 wasmCloud Security Assessment PUBLIC