HashEye

// SMART CONTRACT SECURITY AUDIT

Tact Compiler

Security assessment by HashEye · prepared for TON Studio

HASHEYE AUDITED

PROJECT	Tact Compiler
PERFORMED BY	HashEye Security Review Team - Sergey, Brian
RESEARCH	Sergey, Brian
CLIENT	TON Studio
CATEGORY	TON
PUBLISHED	January 1, 2025
REPORT ID	research-tact-compiler-2025-01-01-10qi1u

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-tact-compiler-2025-01-01-10qi1u.

TON Studio Tact Compiler Security Assessment January 23, 2025 Prepared for: Anton Trunov TON Studio Prepared by: Tarun Bansal, Guillermo Larregay, and Bo Henderson

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 8 Summary of Findings 10 Detailed Findings 11 1. The Tact compiler does not support FunC files with .func extension 11 2. Circular dependencies in traits would crash the Tact compiler 13 3. Symbolic links can be used to bypass path restrictions 15 4. Tact grammar does not handle Unicode correctly 17 5. No validation of shift operator arguments 19 6. Incorrect use of the JavaScript map function for executing side eff ects 20 7. Ohm library limitation for nested expressions 22 A. Vulnerability Categories 23 B. Code Maturity Categories 25 C. Code Quality Findings 26 D. Fix Review Results 28 Detailed Fix Review Results 29 E. Fix Review Status Categories 30 About HashEye 31 Notices and Remarks 32 HashEye 1 TON Studio Tact Compiler PUBLIC Security Assessment

Project Summary Contact Information The following project manager was associated with this project: Jeff Braswell , Project Manager jeff .braswell@hasheye.io The following engineering director was associated with this project: Josselin Feist , Engineering Director, Blockchain josselin.feist@hasheye.io The following consultants were associated with this project: Tarun Bansal , Consultant Guillermo Larregay , Consultant tarun.bansal@hasheye.io guillermo.larregay@hasheye.io Bo Henderson , Consultant bo.henderson@hasheye.io Project Timeline The significant events and milestones of the project are listed below. Date Event September 12, 2024 Pre-project kickoff call September 27, 2024 Status update meeting #1 October 8, 2024 Status update meeting #2 October 15, 2024 Status update meeting #3 October 21, 2024 Delivery of report draft October 21, 2024 Report readout meeting January 23, 2025 Delivery of final comprehensive report January 28, 2025 Addition of fix review appendix HashEye 2 TON Studio Tact Compiler PUBLIC Security Assessment

Executive Summary Engagement Overview The TON Studio engaged HashEye to review the security of the Tact compiler. The Tact language is a high-level programming language for the TON virtual machine. It is compiled into FunC and bytecode that runs on the TON virtual machine. A team of three consultants conducted the review from September 18 to October 18, 2024, for a total of eight engineer weeks of eff ort. Our testing eff orts focused on identifying flaws that could cause incorrect semantic analysis or enable arbitrary code execution or key extraction. With full access to source code and documentation, we performed static and dynamic testing of the codebase using automated and manual processes. Observations and Impact The Tact compiler codebase is broken into well-defi ned components and is easy to navigate. However, some files include a lot of code with a complex code flow involving multiple nested control flow structures, such as the program writer component. Such a complex codebase makes it difficult to follow the data flow and understand the security properties of the project. One medium-severity issue highlights the test suite’s ineffi ciency in capturing logic issues. A comprehensive and effi cient test suite could also discover other low- and informational-severity issues. Recommendations Based on the codebase maturity evaluation and findings identified during the security review, HashEye recommends that the TON Studio 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. • Improve and expand the test suite. A complex system such as the Tact compiler should be thoroughly tested, considering normal usage flows as well as specific or edge cases. The fuzz test suite can also be expanded to test the compiler grammar, optimizer, and writers with well-defi ned invariants. A strong test suite included as part of the CI/CD pipeline will also help detect bugs earlier. HashEye 3 TON Studio Tact Compiler 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 2 Informational 4 Undetermined 0 CATEGORY BREAKDOWN Category Count Access Controls 1 Data Validation 6 HashEye 4 TON Studio Tact Compiler PUBLIC Security Assessment

Project Goals The engagement was scoped to provide a security assessment of the TON Studio’s Tact compiler. Specifically, we sought to answer the following non-exhaustive list of questions: • Is it possible to generate a malicious Tact file that crashes the compiler or generates FunC code that is not compilable? • Is it possible to alter the host’s filesystem outside the current project root directory? Can a malicious project leak data from the host system? • Is it possible to generate an incorrect or invalid program that passes syntactic and semantic checks? • Are there bugs in the Tact compiler that could enable arbitrary code execution? • Is it possible to write visibly correct Tact code that compiles to unexpected FunC code? • Can the optimizer introduce bugs or change the

code behavior? • Does the compiler write correct FunC code files? HashEye 5 TON Studio Tact Compiler PUBLIC Security Assessment

Project Targets The engagement involved a review and testing of the following target. Tact Compiler Repository https://github.com/tact-lang/tact/ Version 0106ea14857bcf3c40dd10135243d0de96012871 Type TypeScript Platform TON HashEye 6 TON Studio Tact Compiler 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: • Manual review and analysis of the provided grammar and semantics • Manual review of the complete compilation flow, from Tact to FunC, from FunC to Fift, and the bag of cells representation of the output • Manual review of the different parts of the Tact abstract syntax tree (AST) generation, analysis and optimization, and translation to FunC • Basic automated fuzz testing of the optimizer and the FunC interpreter • Static analysis of the compiler's TypeScript code files 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: • The optimizer is not yet integrated into the main compiler code. The changes to the compiler code that are needed to add the optimizer to the data flow could affect the code generation or introduce bugs. • The individual AST and code generation functions were reviewed and analyzed, but since the code relies on recursive traversal of structures, it was not tested with all possible edge cases. • The FunC compiler was not part of the scope of the audit. Therefore, any bugs or unexpected behaviors that could be present in the FunC compilation were out of scope. • Third-party libraries used as dependencies were out of scope. • The standard library code and any other Tact files were not reviewed. HashEye 7 TON Studio Tact Compiler 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. Defi ciencies 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 project makes minimal use of arithmetic operations for constant evaluation. We identified no issues in relation to arithmetic overflows, division by zero, or precision loss. Satisfactory Auditing The Tact compiler emits informative log messages at appropriate places. The set of error messages is large, and the information they contain is descriptive. However, we found the use of "internal compiler errors" to be confusing; we recommend documenting when an error is meant to be an internal compiler error and what impact it has on the code. For example, if the Tact code references an undefi ned type, then the compiler throws an internal compiler error instead of a normal error. Moderate Complexity Management The codebase is generally well organized, with modular components and a clear file structure. Many of the source files are short and easy to read. However, some files show several instances of code repetition. Larger modules could be refactored into several files or additional functions to improve legibility and reduce code repetition. Some patterns, such as the recursive processing of the AST nodes, are prone to errors and can be more challenging for a new collaborator to understand. Running static analysis tools on the code could also help detect and fix code smells, as mentioned in the Code Quality Findings section. Moderate Documentation The documentation available on the project's website is Weak HashEye 8 TON Studio Tact Compiler PUBLIC Security Assessment

targeted toward Tact developers. Other than a high-level description of the system in CONTRIBUTING.md , there is no specific documentation for the compiler modules and compilation stages. The code could use more comments, particularly in the bigger files and complex functions. Testing and Verification The provided test suite consists of 895 unit test cases. However, given the size and complexity of the compiler codebase, it is not suitable for the project. The test suite can be improved to cover all the features, as well as more Tact code examples and edge cases, such as the ones described in the Detailed Findings section. The coverage report generation component is broken, so there is no measurable information about the test suite's effi ciency. Weak HashEye 9 TON Studio Tact Compiler PUBLIC Security Assessment

Summary of Findings The table below summarizes the findings of the review, including type and severity details. ID Title Type Severity 1 The Tact compiler does not support FunC files with .func extension Data Validation Informational 2 Circular dependencies in traits would crash the Tact compiler Data Validation Medium 3 Symbolic links can be used to bypass path restrictions Access Controls Low 4 Tact grammar does not handle Unicode correctly Data Validation Low 5 No validation of shift operator arguments Data Validation Informational 6 Incorrect use of the JavaScript map function for executing side eff ects Data Validation Informational 7 Ohm library limitation for nested expressions Data Validation Informational HashEye 10 TON Studio Tact Compiler PUBLIC Security Assessment

Detailed Findings 1. The Tact compiler does not support FunC files with .func extension Severity: Informational Difficulty: Low Type: Data Validation Finding ID: TOB-TACT-1 Target: src/imports/resolveLibrary.ts Description The Tact language compiler considers a source file with the .func extension to be a Tact file and appends the .tact extension to the filename. As a result, importing a .func file throws an exception. The Tact language documentation mentions that a .fc or .func file can be imported with the import keyword. However, the resolveLibrary function does not check for the .func extension while deciding whether a source file is a Tact file or a FunC file. It appends the .tact extension to any file imported that does not have a .tact or .fc extension. The file with the .tact extension appended to it does not exist, which results in a project compilation failure with an exception: let importName = args.name; const kind: "tact" | "func" = importName.endsWith(".fc") ? "func" : "tact" ; if (!importName.endsWith(".tact") && !importName.endsWith(".fc")) { importName = importName + ".tact" ; } Figure 1.1: A snippet of the resolveLibrary function (tact/src/imports/resolveLibrary.ts#L56-L60) Additionally, the exception message shows only the actual filename mentioned in the import statement, which does not help the developer understand the reason for the compilation failure. Exploit Scenario Alice creates a Tact project and tries to import an old FunC source file named old.func in her Tact file named main.tact . The project compilation fails with the exception, Could not resolve import "./old.func" in main.tact . Alice sees that the old.func file exists and cannot understand why the compilation is failing. HashEye 11 TON Studio Tact Compiler PUBLIC Security Assessment

Recommendations Short term, update the resolveLibrary function to consider .func files as FunC source files. Long term, expand the unit test cases to test all the edge cases. HashEye 12 TON Studio Tact Compiler PUBLIC Security Assessment

2. Circular dependencies in traits would crash the Tact compiler Severity: Medium Difficulty: Low Type: Data Validation Finding ID: TOB-TACT-2 Target: src/types/resolveDescriptors.ts Description The contract and traits dependency resolver crashes the compiler when there are circular dependencies in traits. The processType function in the resolveDescriptors function checks whether there are any circular dependencies in traits inherited by a contract by keeping track of intermediate traits in the processing variable: function processType(name: string) { // Check if processed if (processed.has(name)) { return ; } if (processing.has(name)) { throwCompilationError(`Circular dependency detected for type " ${ name } "` , types.get(name)!.ast.loc,); } processing.has(name); // Process dependencies first const dependencies = Array . from (types.values()).filter((v) ⇒ v.traits.find((v2) ⇒ v2.name ⩵⩵⩵ name),); for (const d of dependencies) { processType(d.name); } // Copy traits copyTraits(types.get(name)!); // Mark as processed processed.add(name); processing. delete (name); } Figure 2.1: The processType function (tact/src/types/resolveDescriptors.ts#L1773-L1800) HashEye 13 TON Studio Tact Compiler PUBLIC Security Assessment

However, as shown in the highlighted line in figure 2.1, the trait being processed is not added to the processing list, and because of this, the processType function keeps calling itself recursively indefi nitely if there are circular dependencies in the traits, and the Tact compiler eventually crashes with the Maximum call stack size exceeded error. Exploit Scenario Alice pushes the following Tact code to her remote Git repository, and it crashes her CI/CD pipeline: trait A with B {} trait B with A {} contract Test with A {} Figure 2.2: A contract with a circular trait dependency Recommendations Short term, replace the highlighted line in figure 2.1 with the processing.add(name) statement to track intermediate traits correctly and detect circular dependencies. Long term, improve the test suite to include test cases for all the errors thrown by the compiler. HashEye 14 TON Studio Tact Compiler PUBLIC Security Assessment

3. Symbolic links can be used to bypass path restrictions Severity: Low Difficulty: Low Type: Access Controls Finding ID: TOB-TACT-3 Target: src/vfs/createNodeFileSystem.ts Description The filesystem functions that read and write files do not check for symbolic links. This makes it possible for a user with the right permissions to import files outside the current root path or overwrite arbitrary files. When the readFile and writeFile functions are defi ned in createNodeFileSystem.ts , the default flags are used for fs.readFileSync and fs.writeFileSync calls. This means that when a file is read, the file will be opened, or an exception will be thrown if it does not exist. When a file is written to, the default “w” flag creates the file if it does not exist or opens and truncates the existing file. The only access control check performed in both cases verifies that the file’s location is inside the current root path for the compilation. This check is passed when a symbolic link is placed inside the root path, and since the link destination is not verified, any file could be read or written to. export function createNodeFileSystem(root: string, readonly: boolean = true ,): VirtualFileSystem { let normalizedRoot = path.normalize(root); if (!normalizedRoot.endsWith(path.sep)) { normalizedRoot += path.sep; } return { root: normalizedRoot, exists(filePath: string): boolean { if (!filePath.startsWith(normalizedRoot)) { throw new Error(`Path '${filePath}' is outside of the

root directory '${normalizedRoot}' `,); } return fs.existsSync(filePath); }, resolve(...filePath) { return path.normalize(path.resolve(normalizedRoot, ...filePath)); HashEye 15 TON Studio Tact Compiler PUBLIC Security Assessment

}, readFile(filePath) { if (!filePath.startsWith(normalizedRoot)) { throw new Error(`Path '${filePath}' is outside of the root directory '${normalizedRoot}' `,); } return fs.readFileSync(filePath); }, writeFile(filePath, content) { if (readonly) { throw new Error("File system is readonly"); } if (!filePath.startsWith(normalizedRoot)) { throw new Error(`Path '${filePath}' is outside of the root directory '${normalizedRoot}' `,); } mkdirp.sync(path.dirname(filePath)); fs.writeFileSync(filePath, content); }, }; } Figure 3.1: The createNodeFileSystem function (tact/src/vfs/createNodeFileSystem.ts#L6-L49) Exploit Scenario Bob distributes a malicious Tact project in GitHub, where one of the output files is a symbolic link to ~/.ssh/id_ed25519 . When Alice compiles this project, her default-named SSH private keys are overwritten. Recommendations Short term, either add a check for symbolic links that ensures their target paths are within the project root directory when opening files, or completely disallow symbolic links. Long term, improve the filesystem test suite to include test cases for all platform-specific behavior and exceptional cases mentioned in the Node.js documentation. HashEye 16 TON Studio Tact Compiler PUBLIC Security Assessment

4. Tact grammar does not handle Unicode correctly Severity: Low Difficulty: Low Type: Data Validation Finding ID: TOB-TACT-4 Target: src/grammar/grammar.ohm Description Unicode is a complex encoding standard. Some control characters, such as all the newline character alternatives, are not correctly handled in the Tact grammar. Additionally, it is possible to alter the source code visualization using Unicode right-to-left and left-to-right overrides, causing users and developers to be unaware of potentially dangerous code. The Tact grammar defi nes several line terminators: CR (\u000D), LF (\u000A), LS (\u2028), and PS (\u2029). lineTerminator = "\n" | "\r" | "\u2028" | "\u2029" Figure 4.1: The lineTerminator definition, lacking some mandatory breaks (tact/src/grammar/grammar.ohm#L406) However, Unicode Standard Annex #14 also defi nes FF (\u000C) and VT (\u000B) as mandatory breaks. Some editors (such as Visual Studio Code; figure 4.2a) show the control characters, but some might not (such as a terminal; figure 4.2b), which can be misleading for the user or developer. a) b) Figure 4.2: An example of code that displays differently depending on the editor used HashEye 17 TON Studio Tact Compiler PUBLIC Security Assessment

The same misleading behavior can be reproduced using right-to-left (\u202E) and left-to-right (\u202D) overrides. Exploiting these text direction changes makes it easier for malicious developers to hide unexpected behavior in clear sight. Figure 4.3 shows an example of a contract that can show different operations depending on the editor used. This is a visualization issue; the compiler is unaffected, and the generated FunC code follows the correct logic. a) b) Figure 4.3: An example of code exploiting right-to-left override and left-to-right override control characters Exploit Scenario Bob wants to deploy a malicious contract that intentionally miscalculates fees to steal jettons from the contract users. To gain people’s trust, he makes a blog post showing the contract code and explaining how it works. Since web browsers can understand and process Unicode strings, the code shown is not what is actually compiled, and people can still download and verify that the output bag of cells matches Bob’s. Recommendations Short term, follow Unicode Technical Standard #55 (“Unicode Source Code Handling”) and ensure that all problematic cases are handled correctly. Long term, improve the test suite to include test cases for Unicode ambiguities. HashEye 18 TON Studio Tact Compiler PUBLIC Security Assessment

5. No validation of shift operator arguments Severity: Informational Difficulty: Low Type: Data Validation Finding ID: TOB-TACT-5 Target: src/types/resolveExpression.ts Description The Tact compiler does not validate that the shift operator argument is less than 257. The FunC compiler throws an error for the use of an invalid shift operator argument. The Tact compiler resolves all the types and ensures that the operands of all the operators are of the correct type. However, it does not check the bounds of the operand values, so it could miss some trivial bugs that would then be caught by the FunC compiler or only at runtime. get fun hello1(src: Int): Int { return src << 34605176784551 & 32769; } Figure 5.1: A sample function with an invalid shift operator argument Recommendations Short term, implement a bounds check for shift operator arguments to ensure that the operation does not result in an unexpected value. Long term, consider having the compiler catch trivial runtime errors to improve the developer experience and security of the codebase. HashEye 19 TON Studio Tact Compiler PUBLIC Security Assessment

6. Incorrect use of the JavaScript map function for executing side effects Severity: Informational Difficulty: Low Type: Data Validation Finding ID: TOB-TACT-6 Target: src/grammar/rename.ts Description The renameModuleItems function of the rename.ts file sorts the attributes in the AST for Tact functions, traits, and contracts with the map JavaScript function, which does not update the objects in place. The renameModuleItems function of the rename.ts file is used to canonicalize

the Tact code item names for comparison of two code files. The comparison function expects the renameModuleItems function to return a sorted array of items with sorted attributes for each item object: public renameModuleItems(items: AstModuleItem []): AstModuleItem[] { // Give new names to module-level elements. let renamedItems = items.map((item) ⇒ this .changeItemName(item)); if (this .sort) { renamedItems.map((item) ⇒ this .sortAttributes(item)); } // Apply renaming to the contents of these elements. renamedItems = renamedItems.map((item) ⇒ this .renameModuleItemContents(item),); return this .sort ? this .sortModuleItems(renamedItems) : renamedItems; } Figure 6.1: The renameModuleItems function in the rename.ts file (tact/src/grammar/rename.ts#L114-L128) However, as shown in the highlighted line in figure 5.1, the returned value of the map function call is not stored in the renamedItems variable, which discards any updates made to the renamedItems array items, resulting in unsorted attributes in the code items. These unsorted attributes could cause the comparator to return false negatives for the same code and could allow users to pass the plagiarism test by changing the order of attributes in the code. HashEye 20 TON Studio Tact Compiler PUBLIC Security Assessment

This issue cannot be exploited because of the use of the Array.sort function in the sortAttributes function in the sort.ts file. The Array.sort function sorts the array in place; therefore, the renamedItems array items have sorted attributes. Recommendations Short term, replace the map function with the forEach function. Long term, refactor the codebase to use non-mutating methods to avoid unexpected side eff ects and confusion. HashEye 21 TON Studio Tact Compiler PUBLIC Security Assessment

7. Ohm library limitation for nested expressions Severity: Informational Difficulty: Low Type: Data Validation Finding ID: TOB-TACT-7 Target: src/grammar/grammar.ohm Description The Tact compiler uses the Ohm parsing toolkit to implement the source code parser. The Ohm library uses recursion to parse nested items in expressions and throws a RangeError: Maximum call stack size exceeded error for deeply nested expressions. For example, the expression shown in the following image throws the error and crashes the compiler: ((1)) Figure 7.1: An example expression that would cause a compiler error The above expression is a valid Tact code expression and should be parsed correctly. However, because of the limitations of the Ohm library, the Tact compiler does not support it. Recommendations Short term, document the limitations of the Ohm library for deeply nested expressions. HashEye 22 TON Studio Tact Compiler 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 Undefi ned behavior triggered within the system HashEye 23 TON Studio Tact Compiler 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 24 TON Studio Tact Compiler 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 Complexity Management The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defi ned functions Documentation The presence of comprehensive and readable codebase documentation 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. 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 25 TON Studio Tact Compiler PUBLIC Security Assessment

C. Code Quality Findings The following findings are not associated with any specific vulnerabilities. However, addressing them will enhance code readability and may prevent the introduction of vulnerabilities in the future. • One of the project's dependencies has a known vulnerability. Running npx yarn audit on the repository shows that the ajv-cli package used for validating the linting schema depends on a vulnerable version of the fast-json-patch package. • The TypeScript compiler can be configured to be stricter. The tsconfig.json configuration file does not enforce several options to increase the generated code's robustness and reliability. An example of a stricter tsconfig.json file can be found in the " recommendations for TSConfig bases " repository. The strictest configuration should be used, and all errors and warnings that arise from the process must be addressed. • There are several instances of unused variables and function parameters. Compiling Tact code with the stricter rule set reveals several cases of unused variables in the following files: ○ src/abi/global.ts ○ src/abi/map.ts ○ src/bindings/typescript/serializers.ts ○ src/bindings/typescript/writeStruct.ts ○ src/generator/writers/writeAccessors.ts ○ src/generator/writers/writeContract.ts ○ src/generator/writers/writeSerialization.ts ○ src/types/resolveDescriptors.ts ○ src/types/resolveExpression.ts • Some built-in Ohm rules are redefined in the grammar. The letterAscii , letterAsciiUC , and letterAsciiLC rules are equivalent to the built-in letter , upper , and lower rules, respectively. • There are unused and duplicated rules in the Tact grammar. The letterDigitUnderscore rule is unused and is equivalent to the typeIdPart and idPart rules. HashEye 26 TON Studio Tact Compiler PUBLIC Security Assessment

• There is a misleading error message in the Tact grammar. The checkVariableName function is used to validate several attribute names in grammar.ts . If the validation fails, the error message mentions "variable name" regardless of the origin of the error. This can be confusing for users. • The return type of the buildFieldDescription function does not explicitly indicate all fields. The FieldDescription struct returned by the function lacks an explicit index key, and the order of the returned fields does not match the defi nition. • The division functions do not check a precondition; instead, they rely on the caller. As mentioned in the comments for the divFloor and modFloor functions, they do not check for cases when the divisor is zero. This is not an issue in the current state of the codebase, but it can introduce issues in the future if a developer adds a new use case where the divisor is not checked. • The buildConstantDescription function does not run its struct field count before traversing the struct. When passed a struct, it loops over all fields before checking for the existence of at least one field. This check should be performed before the iterations. HashEye 27 TON Studio Tact Compiler 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. From January 2 to January 3, 2025, HashEye reviewed the fixes and mitigations implemented by the TON Studio team for the issues identified in this report. We reviewed each fix to determine its eff ectiveness in resolving the associated issue. Several of the fixes also improve the test suite, adding cases with the mentioned exploit scenarios and new examples. The Unicode character handling was not changed; however, the team plans on changing the Unicode support for Tact in the long term. The Ohm library nesting limit (TOB-TACT-7) was not solved, as it is outside of the TON Studio codebase, but the Tact compiler documentation was updated to refl ect the issue. Finally, TOB-TACT-5 requires a better solution for the general case; however, the proposed solution works for the cases in which the bit shift operand is constant or can be calculated as a constant. In summary, of the seven issues described in this report, TON Studio has resolved four issues, has partially resolved one issue, and has not resolved the remaining two issues. For additional information, please see the Detailed Fix Review Results below. ID Title Severity Status 1 The Tact compiler does not support FunC files with .func extension Informational Resolved 2 Circular dependencies in traits would crash the Tact compiler Medium Resolved 3 Symbolic links can be used to bypass path restrictions Low Resolved 4 Tact grammar does not handle Unicode correctly Low Unresolved 5 No validation of shift operator arguments Informational Partially Resolved 6 Incorrect use of the JavaScript map function for executing side eff ects Informational Resolved HashEye 28 TON Studio Tact Compiler PUBLIC Security Assessment

7 Ohm library limitation for nested expressions Informational Unresolved Detailed Fix Review Results TOB-TACT-1: The Tact compiler does not support FunC files with .func extension Resolved in commit abe8746 . The .func extension was added to the filename check, and a new test case was added to ensure that .func type files are imported correctly. TOB-TACT-2: Circular dependencies in traits would crash the Tact compiler Resolved in commit d12cf94 . The current trait is now added to the processing list, and the circular dependency is now detected. A new test case was added to check for circular dependencies. TOB-TACT-3: Symbolic links can be used to bypass path restrictions Resolved in commit 40a6342 . A new function to check filepaths for symbolic links and disallow them was added to createNodeFilesystem.ts . All read and write file accesses in the createNodeFileSystem function are now validated. Additionally, test files and test cases were added to check for symbolic link imports. TOB-TACT-4: Tact grammar does not handle Unicode correctly Unresolved. No changes were made to the project to address this issue. The client provided the following context for this finding's fix status: We acknowledge possible Unicode exploits via text editors and most likely our long term goal will be ban Unicode from Tact source code except comments and string literals. TOB-TACT-5: No validation of shift operator arguments Partially resolved in commit 32dbaa8 . The implemented fix works only for constant expressions, without variables. However, the team stated that proper constant propagation static analysis will be included in the next Tact release. Tests were also added for constant bit shifts. TOB-TACT-6: Incorrect use of the JavaScript map function for executing side effects Resolved in commit 00bf680 . The renameModuleItems function was refactored to fix the issue, and now the renamedItems map is correctly sorted when the flag is set. TOB-TACT-7: Ohm library limitation for nested expressions Unresolved. As this is a limitation in the Ohm library, it has not been resolved. However, the team documented the issue in the Expressions section of Tact main documentation. HashEye 29 TON Studio Tact Compiler 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 30 TON Studio Tact Compiler 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 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 31 TON Studio Tact Compiler 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. This report is considered by HashEye to be public information; it is licensed to TON Studio under the terms of the project statement of work and has been made public at TON Studio'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.