

Solang Code Generation

Security assessment by HashEye · prepared for Solana

HASHEYE AUDITED

PROJECT	Solang Code Generation
CLIENT	Solana
CATEGORY	Solana
PUBLISHED	November 1, 2023
REPORT ID	research-solang-code-generation-2023-11-01-17k59n

This report was produced under HashEye's layered review process – **automated detection**, **pattern correlation**, and **senior manual verification** – with every finding signed off by a human reviewer. Full findings detail and on-chain attestation are available on the report page at hasheye.io/audits/research-solang-code-generation-2023-11-01-17k59n.

Solang Code Generation, Part 1 Security Assessment (Summary Report)

November 17, 2023

Prepared for: Sean Young Solana Labs

Prepared by: Anders Helsing

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 Solang 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 Solana under the terms of the project statement of work and has been made public at Solana'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 Solang Security Assessment

PUBLIC

Table of Contents About HashEye 1 Notices and Remarks 2 Table of Contents 3 Executive Summary 4 Project Summary 6 Project Goals 7 Project Targets 8 Project Coverage 9 Summary of Findings 10 Detailed Findings 12 1. Incorrect pointer comparison 12 2. Mixed uses of 32-bit and 64-bit values

14 3. The tests for solana.c cannot be built 16 4. Hard-coded number of accounts in SolParameters 17 5. Out-of-bounds read in the entrypoint function 18 6. Out-of-bounds read in the external_call function 20 7. Out-of-bounds write in the external_call function 22 8. Missing checks ensuring the first account passed to the contract is the state 24 10. Incorrect account deserialization 25 12. Array size changes can result in null pointer dereferences 27 13. Large array accesses cause an instruction to fail 29 14. Array access trips a llvm::checkGEPType assertion 30 A. Vulnerability Categories 31 B. Non-Security-Related Findings 33

HashEye 3 Solang Security Assessment

PUBLIC

Executive Summary Engagement Overview Solana Labs engaged HashEye to review the security of the Solang emit crate and the Solana standard library C code. From April 24 to May 5, 2023, one consultant conducted a security review of the client-provided source code, with two person-weeks of effort. Details of the project’s timeline, test targets, and coverage are provided in subsequent sections of this report. Project Scope Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the target system, including access to the source code and documentation. We performed static and dynamic testing of the target system and its codebase, using both automated and manual processes. Summary of Findings The audit uncovered issues of high severity that could impact system confidentiality, integrity, or availability. A summary of the findings is provided on the next page. Summary of Recommendations

In addition to addressing the findings in this report, HashEye recommends taking the following steps:

- Update and increase the test coverage for the C code in the stdlib folder, and run the tests with address sanitization and undefined behavior sanitization enabled. Incorporate the tests into the CI process to ensure they are run regularly.
- While the architecture of emitting LLVM IR code for both the Solana and Substrate target is elegant, and allows for a relatively high degree of common code between the targets, it comes with a price.

Creating loads and stores to application memory is intrinsic to the process of generating code using LLVM IR. However, for each such operation, the correct address needs to be calculated. The emit crate does this by calling the build_gep function from the inkwell crate. According to Inkwell’s documentation, “GEP is very likely to segfault if indexes are used incorrectly, and is therefore an unsafe function.” The code under test contains 78 calls to the function, each of which needs to correctly calculate the address in order to avoid memory corruption.

The complexity of the process of emitting LLVM IR, coupled with the specialized

HashEye 4 Solang Security Assessment

PUBLIC

traits needed for both supported targets, makes verification of the correctness of the generated code difficult, which increases the risk that bugs are not uncovered.

For the Solana target, smart contracts are typically developed in Rust, relying on the Solana Rust API or the Anchor framework. Among the benefits of using Rust is the guarantee of memory safety, and by leveraging the Anchor framework, the security considerations of contracts can be simplified.

With this in mind, our recommendation is to investigate the feasibility of drawing on these benefits by generating Rust code (to be compiled into the eBPF ELF binary) from the AST. This process could replace the current process of emitting LLVM IR altogether, or it could be done alongside it to aid in the verification of the generated code. The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS Severity Count High 3 Medium 0 Low 0 Informational 5 Undetermined 4

CATEGORY BREAKDOWN Category Count Authentication 1 Testing 1 Undefined Behavior 10

HashEye 5 Solang Security Assessment

PUBLIC

Project Summary Contact Information The following 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 engineer was associated with this project: Anders Helsing, Consultant anders.helsing@hasheye.io Project Timeline The significant events and milestones of the project are listed below. Date Event April 25, 2023 Project kickoff call May 1, 2023 Delivery of report draft May 8, 2023 Report readout meeting November 17, 2023 Delivery of final report

HashEye 6 Solang Security Assessment

PUBLIC

Project Goals The engagement was scoped to provide a security assessment of the Solang emit crate and Solang standard library C code. Specifically, we sought to answer the following non-exhaustive list of questions: • Are there issues related to memory safety in the stdlib C code? • Is the received contract data correctly decoded into account data, program input, and contract storage? • Does the emitted LLVM IR accurately represent the intention of the Solidity contract code? • Are there any security issues related to how contract storage is modeled? • Are the existing testing strategies sufficient, or could they be extended?

HashEye 7 Solang Security Assessment

PUBLIC

Project Targets The engagement involved a review and testing of the following targets. Solang stdlib Repository <https://github.com/hyperledger/solang/tree/main/stdlib> Version 3910219ed0d536dfdd017f6eb4b2828e75e99855 Type C Platform Solana Solang emit Crate Repository <https://github.com/hyperledger/solang/tree/main/src/emit> Version 3910219ed0d536dfdd017f6eb4b2828e75e99855 Type Rust Platform Solana Solang codegen Crate Repository <https://github.com/hyperledger/solang/tree/main/src/codegen> Version 3910219ed0d536dfdd017f6eb4b2828e75e99855 Type Rust Platform Solana

HashEye 8 Solang 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: • Code review: ◦ We reviewed the Solana-relevant C code in the stdlib folder. ◦ We reviewed the complete code path from the contract entrypoint function to the function dispatch. ◦ We reviewed the emit crate, with a focus on code relevant to Solana. • Fuzzing: ◦ We built a limited fuzzer for the custom Solana storage heap, simulating extended heap operations of different sizes. Coverage Limitations Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. Due to time constraints during this project, we could not achieve complete coverage of the emit crate, which may warrant further review: 1. We did not identify the root cause of three bugs in the generated LLVM IR code. 2. We performed only a cursory review of the code other than the contract storage.

HashEye 9 Solang 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 Incorrect pointer comparison Undefined Behavior Undetermined 2 Mixed uses of 32-bit and 64-bit values Undefined Behavior Undetermined 3 The tests for solana.c cannot be built Testing Informational 4 Hard-coded number of accounts in SolParameters Undefined Behavior High 5 Out-of-bounds read in the entrypoint function Undefined Behavior Informational 6 Out-of-bounds read in the external_call function Undefined Behavior High 7 Out-of-bounds write in the external_call function Undefined Behavior High 8 Missing checks ensuring the first account passed to the contract is the state

Authentication Undetermined 10 Incorrect account deserialization Undefined Behavior Undetermined 12 Array size changes can result in null pointer dereferences Undefined Behavior Informational 13 Large array accesses cause an instruction to fail Undefined Behavior Informational

HashEye 10 Solang Security Assessment

PUBLIC

14 Array access trips a llvm::checkGEPType assertion Undefined Behavior Informational

HashEye 11 Solang Security Assessment

PUBLIC

Detailed Findings 1. Incorrect pointer comparison Severity: Undetermined Difficulty: High Type: Undefined Behavior Finding ID: TOB-SOLEMIT-1 Target: solang/stdlib/stdlib.c

Description The vector_new function receives a pointer to initial values with which to populate the vector. Our impression is that, because the zero address is valid in WASM, the function casts the pointer to an int and compares it to -1 to decide whether it should be used to populate the vector. WASM is outside of the scope of this engagement, but it should be noted that dereferencing a null pointer in C is undefined behavior. However, on Solana, the architecture is 64-bit, and casting the pointer to int and comparing it to -1 has the effect that if the pointer's lower 32 bits has the value 0xffffffff, the initialization will be omitted. 129 // Create a new vector. If initial is -1 then clear the data. This is done since a null pointer valid in wasm 130 struct vector *vector_new(uint32_t members, uint32_t size, uint8_t *initial) 131 { 132 struct vector *v; 133 size_t size_array = members * size; 134 135 v = __malloc(sizeof(*v) + size_array); 136 v->len = members; 137 v->size = members; 138 139 uint8_t *data = v->data; 140 141 if ((int)initial != -1) Figure 1.1: stdlib/stdlib.c#L129-L141

Exploit Scenario Mallory notices this issue and finds a contract that is affected by it. She proceeds to operate the contract outside of its intended use by triggering the bug.

HashEye 12 Solang Security Assessment

PUBLIC

Recommendations Short term, have the function cast the value to a type with a width equal to the pointer side for the target (e.g., ptrdiff_t). This will ensure that the pointer is compared to the full 64-bit -1 value on Solana, which is not a legal pointer value. Long term, use conditional compile structures to create different implementations for the targets in order to handle the architectural differences. This will reduce the code complexity by removing the need to come up with technical workarounds that fit both targets.

HashEye 13 Solang Security Assessment

PUBLIC

2. Mixed uses of 32-bit and 64-bit values Severity: Undetermined Difficulty: High Type: Undefined Behavior Finding ID: TOB-SOLEMIT-2 Target: solang/stdlib/

Description In several places, values that have widths of 64 bits and 32 bits are used interchangeably. For example, the ka_num variable is a uint64_t, but it is regularly passed to functions with an int-type argument (figure 2.1). This is quite harmless because, in practice, a transaction cannot contain enough accounts to overflow an int value. However, the __malloc function uses uint32_t for the size argument but is called in several places with a size_t value. The same is true for the len and size fields of the vector struct. Also, the __realloc function implicitly truncates the size of an allocated chunk from size_t to uint32_t. Should any of these values be larger than the maximum value for uint32_t, memory corruption will occur. 111 return sol_invoke_signed_c(&instruction, params->ka, params->ka_num, NULL, 0); Figure 2.1: The harmless truncation of ka_num (stdlib/solana.c#L111) 187 struct vector *concat(uint8_t *left, uint32_t left_len, uint8_t *right, uint32_t right_len) 188 { 189 size_t size_array = left_len + right_len; 190 struct vector *v = __malloc(sizeof(*v) + size_array); 191 v->len = size_array; 192 v->size = size_array; Figure 2.2: The truncation of the __malloc and vector fields (stdlib/stdlib.c#L187-L192) 125 void *__realloc(void *m, size_t size) 126 { 127 struct chunk *cur = m; 128

HashEye 14 Solang Security Assessment

PUBLIC

```
129 cur--; 130 131 struct chunk *next = cur->next; 132 133 if (next && !next->allocated && size <=
(cur->length + next->length + sizeof(struct chunk))) 134 { 135 // merge with next 136 cur->next =
next->next; 137 if (cur->next) 138 cur->next->prev = cur; 139 cur->length += next->length +
sizeof(struct chunk); 140 // resplit .. 141 shrink_chunk(cur, size); 142 return m; 143 } 144 else
145 { 146 // allocate new area and copy old data 147 uint32_t len = cur->length; Figure 2.3: The
truncation of the chunk length ( stdlib/heap.c#L125-L147 ) Recommendations Short term, fix all
instances of inconsistent uses of type sizes that relate to memory management operations. Long
term, use type sizes consistently. Avoid implicit casts, and where a narrowing cast is intended,
use an explicit cast and add a comment explaining why the cast is safe. Regularly build the stdlib
targets with -Wshorten-64-to-32 or -Wconversion to uncover new conversion issues.
```

HashEye 15 Solang Security Assessment

PUBLIC

3. The tests for solana.c cannot be built Severity: Informational Difficulty: High Type: Testing Finding ID: TOB-SOLEMIT-3 Target: solang/stdlib/solana.c

Description Building the tests using clang -DTEST -DSOL_TEST -O3 -Wall solana.c stdlib.c -o test results in several errors. It appears that the code has been updated, but the tests have not.

Recommendations Short term, update the test code to match the updated code. Long term, regularly run the tests to ensure that failing tests, including build fails, are uncovered.

HashEye 16 Solang Security Assessment

PUBLIC

4. Hard-coded number of accounts in SolParameters Severity: High Difficulty: High Type: Undefined Behavior Finding ID: TOB-SOLEMIT-4 Target: solang/stdlib/solana_sdk.h

Description The ka field of the SolParameters struct is hard-coded to be 10 elements. However, in the sol_deserialize function, the ka_num field of the same struct is set to the total number of received accounts. As a result, any function that iterates over the ka array using ka_num as the high bound (such as the functions cited in findings TOB-SOLEMIT-5, TOB-SOLEMIT-6, and TOB-SOLEMIT-7) will perform an out-of-bounds access if the number of accounts is greater than 10. Exploit Scenario Mallory discovers an out-of-bounds condition and finds a way to either leak back the read data or modify the contract state.

Recommendations Short term, either have the relevant code dynamically allocate the number of entries in the ka array, or limit the ka_num field to reflect the number of entries in the array. Note that if the number of usable accounts is limited to 10, this should be reflected in the documentation. Long term, keep the number of hard-coded arrays to a minimum.

HashEye 17 Solang Security Assessment

PUBLIC

5. Out-of-bounds read in the entrypoint function Severity: Informational Difficulty: High Type: Undefined Behavior Finding ID: TOB-SOLEMIT-5 Target: stdlib/solana.c

Description The entrypoint function iterates over the ka array in the params struct with the ka_num field as the upper bound. This will result in an out-of-bounds read if more than 10 accounts are passed to the contract. 18 uint64_t 19 entrypoint(const uint8_t *input) 20 { 21 SolParameters params; 22 23 uint64_t ret = sol_deserialize(input, ¶ms); 24 if (ret) 25 { 26 return ret; 27 } 28 29 params.ka_clock = NULL; 30 params.ka_instructions = NULL; 31 32 for (int account_no = 0; account_no < params.ka_num; account_no++) 33 { 34 const SolAccountInfo *acc = ¶ms.ka[account_no]; 35 36 if (SolPubkey_same(&clock_address, acc->key)) 37 { 38 params.ka_clock = acc; 39 } 40 else if (SolPubkey_same(&instructions_address, acc->key)) 41 { 42 params.ka_instructions = acc; 43 } 44 }

HashEye 18 Solang Security Assessment

PUBLIC

```
45 46 __init_heap(); 47 48 return solang_dispatch(&params); 49 } Figure 5.1: stdlib/solana.c#L18-L49
```

Because it is unlikely that the data that would be read out-of-bounds will contain the public key address of the `clock_address` or `instructions_address`, the severity of this finding is set to informational. Recommendations Short term, apply the short-term recommendation offered for finding TOB-SOLEMIT-4.

HashEye 19 Solang Security Assessment

PUBLIC

6. Out-of-bounds read in the `external_call` function Severity: High Difficulty: High Type: Undefined Behavior Finding ID: TOB-SOLEMIT-6 Target: `stdlib/solana.c`

Description The `external_call` function iterates over the `ka` array in the `params` struct with the `ka_num` field as the upper bound. This will result in an out-of-bounds read if more than 10 accounts are passed to the contract. 59 // Calls an external function when 'program_id' is NULL or 60 // creates a new contract and calls its constructor. 61 uint64_t external_call(uint8_t *input, uint32_t input_len, SolPubkey *address, 62 SolPubkey *program_id, const SolSignerSeeds *seeds, 63 int seeds_len, SolParameters *params) 64 { 65 SolAccountMeta metas[10]; 66 SolInstruction instruction = { 67 .program_id = program_id, 68 .accounts = metas, 69 .account_len = params->ka_num, 70 .data = input, 71 .data_len = input_len, 72 }; 73 74 int meta_no = 1; 75 int new_address_idx = -1; 76 77 for (int account_no = 0; account_no < params->ka_num; account_no++) 78 { 79 SolAccountInfo *acc = ¶ms->ka[account_no];

Figure 6.1: `stdlib/solana.c#L59-L89`

HashEye 20 Solang Security Assessment

PUBLIC

Exploit Scenario Mallory discovers the out-of-bounds read condition and finds a way to leak back the read data.

Recommendations Short term, apply the short-term recommendation offered for finding TOB-SOLEMIT-4. Long term, create unit tests covering the `external_call` function, and run the tests with address sanitization and undefined behavior sanitization enabled.

HashEye 21 Solang Security Assessment

PUBLIC

7. Out-of-bounds write in the `external_call` function Severity: High Difficulty: High Type: Undefined Behavior Finding ID: TOB-SOLEMIT-7 Target: `solang/stdlib/solana.c`

Description The `external_call` function defines a `SolAccountMeta` array with a hard-coded size of 10 elements, and it iterates over this array to write accounts to it, with the `ka_num` field as the upper bound. As a result, the function will perform an out-of-bounds write if more than 10 accounts are passed to the contract. 59 // Calls an external function when 'program_id' is NULL or 60 // creates a new contract and calls its constructor. 61 uint64_t external_call(uint8_t *input, uint32_t input_len, SolPubkey *address, 62 SolPubkey *program_id, const SolSignerSeeds *seeds, 63 int seeds_len, SolParameters *params) 64 { 65 SolAccountMeta metas[10]; 66 SolInstruction instruction = { 67 .program_id = program_id, 68 .accounts = metas, 69 .account_len = params->ka_num, 70 .data = input, 71 .data_len = input_len, 72 }; 73 74 int meta_no = 1; 75 int new_address_idx = -1; 76 77 for (int account_no = 0; account_no < params->ka_num; account_no++) 78 { 79 SolAccountInfo *acc = ¶ms->ka[account_no]; 80 81 // The address for the new contract should go first. Note that there 82 // may be duplicate entries, the order of those does not matter. 83 if (new_address_idx < 0 && SolPubkey_same(address, acc->key))

HashEye 22 Solang Security Assessment

PUBLIC

```
84 { 85 metas[0].pubkey = acc->key; 86 metas[0].is_writable = acc->is_writable; 87
metas[0].is_signer = acc->is_signer; 88 new_address_idx = account_no; 89 } 90 else 91 { 92
```

```
metas[meta_no].pubkey= acc→key; 93 metas[meta_no].is_writable = acc→is_writable; 94
metas[meta_no].is_signer = acc→is_signer; 95 meta_no += 1; 96 } 97 }
```

Figure 7.1: stdlib/solana.c#L59-L89 Exploit Scenario Mallory discovers the out-of-bounds write condition and finds a way to modify the contract state.

Recommendations Short term, apply the short-term recommendation offered for finding TOB-SOLEMIT-4. Long term, apply the long-term recommendation offered for finding TOB-SOLEMIT-6.

HashEye 23 Solang Security Assessment

PUBLIC

8. Missing checks ensuring the first account passed to the contract is the state Severity: Undetermined Difficulty: High Type: Authentication Finding ID: TOB-SOLEMIT-8 Target: Various files

Description The contract functionality implements state storage using a dedicated account, which is intended to be the first of the accounts passed to the contract. However, apart from checks to ensure that magic is set to the intended value, there are no checks to ensure that the first passed account is an actual state account for the contract (i.e., that it has the intended owner and address values). Exploit Scenario Mallory finds a contract that, if a certain state in the state account is met, uses CPI to transfer funds from a PDA vault under the contract's control. By passing in an account under her control, she fakes the necessary state and transfers funds to herself. Recommendations Short term, for each account used by the contract, implement checks to ensure that the address and owner of the account are the intended ones. Long term, create integration tests, with test cases for attacks such as the one described in the exploit scenario, to ensure similar issues are uncovered if future bugs are introduced.

HashEye 24 Solang Security Assessment

PUBLIC

10. Incorrect account deserialization Severity: Undetermined Difficulty: High Type: Undefined Behavior Finding ID: TOB-SOLEMIT-10 Target: solang/stdlib/solana_sdk.h

Description The sol_deserialize function iterates over received input bytes and populates the SolParameters array. If the number of accounts passed to the contract exceeds the hard-coded limit of the ka array, the input pointer is advanced, but no data is stored. However, in this case, if the dup_info value is not equal to UINT8_MAX, the pointer is not advanced to account for the 7-byte padding. As a result, the input pointer will reference incorrect data at the end of the loop, causing the input_len, input, and program_id variables to be incorrect. 241 static uint64_t sol_deserialize(242 const uint8_t *input, 243 SolParameters *params) 244 { 245 if (NULL == input || NULL == params) 246 { 247 return ERROR_INVALID_ARGUMENT; 248 } 249 params→ka_num = *(uint64_t *)input; 250 input += sizeof(uint64_t); 251 252 for (int i = 0; i < params→ka_num; i++) 253 { 254 uint8_t dup_info = input[0]; 255 input += sizeof(uint8_t); 256 257 if (i ≥ SOL_ARRAY_SIZE(params->ka)) 258 { 259 if (dup_info == UINT8_MAX) 260 { 261 input += sizeof(uint8_t); 262 input += sizeof(uint8_t); 263 input += sizeof(uint8_t);

HashEye 25 Solang Security Assessment

PUBLIC

```
264 input += 4; // padding 265 input += sizeof(SolPubkey); 266 input += sizeof(SolPubkey); 267
input += sizeof(uint64_t); 268 uint64_t data_len = *(uint64_t *)input; 269 input +=
sizeof(uint64_t); 270 input += data_len; 271 input += MAX_PERMITTED_DATA_INCREASE; 272 input =
(uint8_t *)(((uint64_t)input + 8 - 1) & ~(8 - 1)); // padding 273 input += sizeof(uint64_t); 274 }
275 continue; 276 } ... 330 } 331 332 uint64_t data_len = *(uint64_t *)input; 333 input +=
sizeof(uint64_t); 334 335 params→input_len = data_len; 336 params→input = input; 337 input +=
data_len; 338 339 params→program_id = (SolPubkey *)input; 340 input += sizeof(SolPubkey); 341 342
return 0; 343 }
```

Figure 10.1: stdlib/solana_sdk.h#L227-L343 Exploit Scenario Mallory notices the bug and finds a way to control the values of input_len, input, or program_id to her advantage.

Recommendations Short term, have the function advance the pointer to skip the padding if dup_info is not equal to UINT8_MAX. Long term, create unit tests for sol_deserialize and ensure that all code paths are covered by test cases.

HashEye 26 Solang Security Assessment

PUBLIC

12. Array size changes can result in null pointer dereferences Severity: Informational Difficulty: High Type: Undefined Behavior Finding ID: TOB-SOLEMIT-12 Target: solang/src/emit/instructions.rs

Description Corner cases in the push operation can lead to null pointer dereferences in the emitted call to the `__realloc` function. Due to time constraints, we were unable to find the root cause of this problem.

```
165 let arr = w.vars[array].value; // Corner cases can lead to arr being null here
166 167 let llvm_ty = bin.llvm_type(ty, ns); 168 let elem_ty = ty.array_elem(); 169 170 //
Calculate total size for reallocation 171 let llvm_elem_ty = bin.llvm_field_ty(&elem_ty, ns); 172
let elem_size = llvm_elem_ty 173 .size_of() 174 .unwrap() 175 .const_cast(bin.context.i32_type(),
false); 176 let len = bin.vector_len(arr); 177 let new_len = 178 bin.builder 179
.build_int_add(len, bin.context.i32_type().const_int(1, false), ""); 180 let vec_size = bin 181
.module 182 .get_struct_type("struct.vector") 183 .unwrap() 184 .size_of() 185 .unwrap() 186
.const_cast(bin.context.i32_type(), false); 187 let size = bin.builder.build_int_mul(elem_size,
new_len, ""); 188 let size = bin.builder.build_int_add(size, vec_size, ""); 189
```

HashEye 27 Solang Security Assessment

PUBLIC

```
190 let realloc_size = if ns.target == Target::Solana { 191 bin.builder 192
.build_int_z_extend(size, bin.context.i64_type(), "") 193 } else { 194 size 195 }; 196 197 //
Reallocate and reassign the array pointer 198 let new = bin 199 .builder 200 .build_call( 201
bin.module.get_function("__realloc").unwrap(), 202 &[arr.into(), realloc_size.into()], 203 "", //
Calling __realloc with null causes a null pointer dereference 204 ) 205 .try_as_basic_value() 206
.left() 207 .unwrap() 208 .into_pointer_value(); 209 w.vars.get_mut(array).unwrap().value =
new.into(); Figure 12.1: solang/src/emit/instructions.rs#L165-L209 function foo() public pure
returns (bytes memory) { bytes b1 = hex"42"; bytes b2 = hex"41";

b2.push(0x41);
```

return (b1); } Figure 12.2: An example corner case that will cause a null pointer dereference
function foo2_func() public pure { uint64[] a; a.push(2); } Figure 12.3: Another example corner
case that will cause a null pointer dereference Recommendations Short term, find the root problem
causing the null pointer, and triage the security implications of the bug from a memory corruption
perspective.

HashEye 28 Solang Security Assessment

PUBLIC

13. Large array accesses cause an instruction to fail Severity: Informational Difficulty: High Type: Undefined Behavior Finding ID: TOB-SOLEMIT-13 Target: solang/src/emit/instructions.rs

Description Modifying the store contract's `get_values3` function to include 935 push operations on a local bytes array causes the instruction to fail, whereas 934 iterations succeed. Due to time constraints, we were unable to find the root cause of this problem.

```
function get_values3() public
view returns (uint256, string memory, bytes memory, bytes4, enum_bar) { bytes b2 = hex"41";

uint i;

for (i=0; i<935; i++) { b2.push(0x41); }
```

return (u256, str, b2, fixedbytes, bar); } Figure 13.1: An example showing that 935 iterations
cause the instruction to fail Recommendations Short term, find the root problem causing this issue
and, if possible, put checks in place that will detect this error at compile time.

HashEye 29 Solang Security Assessment

PUBLIC

14. Array access trips a `llvm::checkGEPType` assertion Severity: Informational Difficulty: High Type: Undefined Behavior Finding ID: TOB-SOLEMIT-14 Target: solang/src/emit/instructions.rs

Description A corner case function that accesses an array trips a `llvm::checkGEType` assertion. Due to time constraints, we were unable to find the root cause of this problem. function `foo_func3()`

```
public view returns (uint256) { uint64[10] a;

a[9] = 0x41; a.push(2);

return (a[9]); }
```

Figure 14.1: An example function that trips the `llvm::checkGEType` assertion

solang: /home/runner/work/solang-llvm/solang-llvm/llvm-project/llvm/include/llvm/IR/Instructions.h:922: `llvm::Type* llvm::checkGEType(llvm::Type*): Assertion `Ty && "Invalid GetElementPtrInst indices for type!"` failed. Aborted (core dumped)

Figure 14.2: The error generated by compiling the function

Recommendations Short term, find the root problem causing this issue and mitigate it.

HashEye 30 Solang Security Assessment

PUBLIC

A. Vulnerability Categories The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability 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
Insufficient test methodology or test coverage	Timing Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

HashEye 31 Solang 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 32 Solang Security Assessment

PUBLIC

B. Non-Security-Related Findings The following finding is not associated with specific vulnerabilities. However, addressing it will enhance code readability and may prevent the introduction of vulnerabilities in the future.

- The documentation of the data field of the `AccountInfo` struct states that "This field can be modified, but use with caution"; however, it is marked as read-only in the code.

HashEye 33 Solang Security Assessment

PUBLIC