Casper Ledger

Security Assessment

July 19, 2021

Prepared For:
Medha Parlikar | *Casper*
medha@casperlabs.io

Prepared By:
Jim Miller | *Trail of Bits*
james.miller@trailofbits.com

Brad Larsen | *Trail of Bits*
brad.larsen@trailofbits.com

Rory Mackie | *Trail of Bits*
rory.mackie@trailofbits.com

Changelog:

July 19, 2021	Initial report delivered
July 28, 2021	Fix review (Appendix D) added

Executive Summary

From July 6 to July 16, 2021, Casper engaged Trail of Bits to review the security of the Casper Ledger integration, produced by the Zondax team, and the Casper web wallet. Trail of Bits performed this assessment over four person-weeks, with two engineers working from commit hash `db71d06` of the `ledger-casper` repository and commit hash `fd01664` of the `casper-explorer` repository.

We began the assessment by reviewing the Ledger integration. We built the Ledger integration codebase and verified that its existing tests would run. We analyzed the test coverage and the provided fuzz tests, identified some testing issues (TOB-CLL-004), and expanded the test suite to increase code coverage. We also ran the `clang-tidy` and `cppcheck` static analyzers and used their results to further survey the codebase. In addition to this automated testing, we manually reviewed the codebase, looking for issues like insufficient input validation, buffer overflows, and the improper handling of sensitive data and errors.

After reviewing the Ledger integration, we reviewed the Casper web wallet. We ran TypeScript linters, including the TSLint rules developed by Microsoft, to identify issues and opportunities to improve the code quality. Appendix C lists code our quality recommendations based on the results of this linting. We also manually reviewed the codebase, looking for issues like logic errors, improper input validation, and improper error handling.

The assessment resulted in four findings. The one low-severity finding pertains to a bug in the web wallet utilities (TOB-CLL-006). The three remaining issues are of informational severity and concern the risk of a denial of service (TOB-CLL-002) and opportunities to improve the parsing code fuzzer (TOB-CLL-004).

The Ledger integration has a large number of hand-written tests. The coverage of the provided fuzz tests is somewhat limited (TOB-CLL-004), but the expanded fuzzers we built did not identify any security issues. The web wallet codebase, on the other hand, contains minimal testing. Both the Ledger and the web wallet codebases would benefit from increased documentation, as a more comprehensive high-level specification and inline comments would help make the codebases easier to review and maintain. In addition, we recommend integrating our guidance on improving fuzzing coverage into the Ledger codebase. For the web wallet, we recommend developing more tests and using the Microsoft TSLint rules.

Update: During the week of July 26, 2021, Trail of Bits reviewed fixes implemented for the issues in this report. A detailed review of the current status of each issue is provided in Appendix D.

Project Dashboard

Application Summary

Name	Casper Ledger and web wallet
Versions	ledger-casper, commit db71d06e casper-explorer, commit fd01664
Type	Hardware Wallet Integration
Platform	Ledger

Engagement Summary

Dates	July 6–16, 2021
Method	Full Source Code Access
Consultants Engaged	3
Level of Effort	4 person-weeks

Vulnerability Summary

Total High-Severity Issues	0	
Total Medium-Severity Issues	0	
Total Low-Severity Issues	1	■
Total Informational-Severity Issues	3	■ ■ ■
Total Undetermined-Severity Issues	0	
Total	4	

Category Breakdown

Data Validation	1	■
Denial of Service	1	■
Testing	1	■
Undefined Behavior	1	■
Total	4	

Code Maturity Evaluation

Category Name	Description
Access Controls	**Satisfactory.** The Ledger integration codebase demonstrates strong data validation, as our extended fuzzing efforts did not result in any crashes. Strong data validation helps prevent unauthorized access to sensitive components.
Arithmetic	**Moderate.** The web wallet uses a big-number library to prevent arithmetic overflows. However, this arithmetic is not tested well and could contain logic errors (TOB-CLL-006). In addition, we found one arithmetic-related issue in the Ledger integration that could cause a denial of service (TOB-CLL-002).
Assembly Use	**Not applicable.**
Centralization	**Not applicable.**
Upgradeability	**Not applicable.**
Function Composition	**Satisfactory.** Although the code has sparse documentation, it is largely well composed, with functions that serve one purpose.
Front-Running	**Not applicable.**
Key Management	**Satisfactory.** The few areas of the Ledger code that deal with private keys carefully follow the key management guidelines outlined in the Developing Secure Ledger Apps documentation.
Monitoring	**Not applicable.**
Specification	**Moderate.** The code has sparse high-level and inline documentation. Some of the inline documentation either lacks detail or is unclear.
Testing & Verification	**Moderate.** The Ledger integration code has a large number of handwritten test cases and an existing fuzzer for its parsing code. However, both the effectiveness of the fuzzer (TOB-CLL-004) and the number of web wallet tests are limited.

Engagement Goals

The engagement was scoped to provide a security assessment of the Casper Ledger integration, developed by Zondax, and the Casper web wallet.

Specifically, we sought to answer the following questions:

- Does the Ledger code violate the security guidelines set out in the Developing Secure Ledger Apps documentation?
- Could malformed or malcrafted data cause the web wallet or hardware wallet to exhibit unintended behavior?
- Do third-party dependencies contain known vulnerabilities?
- Are there untested parts of the code?

Coverage

Casper Ledger. We manually reviewed the code, focusing on issues like insufficient input validation, buffer overflows, improper handling of sensitive data, and improper error handling. We also examined and ran the existing transaction parser fuzzer (TOB-CLL-004) and reviewed the results of the static analyzers that we ran over the codebase.

Web wallet. We manually reviewed the code, focusing on issues like logic errors, improper input validation, and improper error handling. We also ran various linters on the codebase and reviewed their results (Appendix C).

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

❑ **Change the type of the index variable in the `for` loop in `parser_runtimeargs_getData` from a `uint8_t` to a `uint32_t` so that it is the same type as the loop bound variable.** Alternatively, add an error-checking code path that returns early if the loop bound variable is greater than 255. Either of these approaches should prevent the parser from looping infinitely. TOB-CLL-002

❑ **Revise the `addr_getItem` function so that `zemu_log_stack` is called as intended.** TOB-CLL-003

❑ **Eliminate the nonfunctional duplicates of the fuzzing support scripts, add the missing comma to the `run-fuzzers.py` script, and create a seed corpus of well-formed transaction inputs for the transaction parser fuzz target.** The developers could add extra logic to the `run-fuzzers.py` script to extract the inputs from the existing manual tests into a seed corpus prior to fuzzing. Implementing these recommendations will improve the effectiveness of the existing fuzzing infrastructure. TOB-CLL-004

❑ **Correct the typo in the `if` statement in `divBigNumbersWithPrecision` so that it checks whether the `convertedNumerator` is greater than or equal to the `convertedDenominator`.** TOB-CLL-006

Long term

❑ **Regularly review the results from static analysis tools such as `clang-tidy` and `cppcheck`; these tools can reveal problems like infinite loops and mismatches between integer types in the code.** In particular, the `bugprone-too-small-loop-variable` check from `clang-tidy` identifies the code problem at the root of this issue. TOB-CLL-002

❑ **Run the `run-fuzzers.py` script for a few minutes in the continuous integration system, and periodically review source-level code coverage results from fuzzing to identify any coverage gaps.** That way, the fuzzing infrastructure provides ongoing value. TOB-CLL-004

Findings Summary

#	Title	Type	Severity
1	Malformed data could cause the transaction parser to loop infinitely	Denial of Service	Informational
2	Use of uninitialized memory when ZEMU_LOGGING is defined	Undefined Behavior	Informational
3	Limited effectiveness of the transaction parser fuzz target	Testing	Informational
4	Bug in divBigNumbersWithPrecision function	Data Validation	Low

1. Malformed data could cause the transaction parser to loop infinitely

Severity: Informational
Type: Denial of Service
Target: `ledger-casper/app/src/parser.c`

Difficulty: N/A
Finding ID: TOB-CLL-002

Description

The internal `parser_runtimeargs_getData` function could loop infinitely if provided with malformed data.

The `for` loop in `parser_runtimeargs_getData`, shown in figure 1.1, uses a `uint8_t` loop index variable but a `uint32_t` loop bound variable. A `uint8_t` variable can represent a maximum value of 255. If the value of the loop index variable exceeds this maximum value, it will wrap around to 0, which is well-defined behavior for unsigned integers in C and C++. If the loop bound variable has a value greater than 255, the loop termination condition will always be false and the loop will execute infinitely.

```
parser_error_t parser_runtimeargs_getData(char *keystr, uint32_t *length, uint8_t *runtype,
uint32_t num_items, parser_context_t *ctx) {
    // ...
    for (uint8_t index = 0; index < num_items; index++) {
        // ...
    }

    return parser_no_data;
}
```

Figure 1.1: The abbreviated definition of the `parser_runtimeargs_getData` *function, showing the mismatch between the loop index and loop bound variable types (*[`ledger-casper/app/src/parser.c:235-282`]*)*

The `parser_runtimeargs_getData` function is called from the `parser_getItem_Transfer` function with a `num_items` argument read from user-controlled transaction data, as shown in figure 1.2.

```
parser_error_t parser_getItem_Transfer(ExecutableDeployItem item, parser_context_t *ctx,
                                       uint8_t displayIdx,
                                       char *outKey, uint16_t outKeyLen,
                                       char *outVal, uint16_t outValLen,
                                       uint8_t pageIdx, uint8_t *pageCount) {
    // ...
    uint32_t num_items = 0;
    CHECK_PARSER_ERR(readU32(ctx, &num_items));

    // ...
    if(!app_mode_expert()){
        if(new_displayIdx == 0) {
```

```
            snprintf(outKey, outKeyLen, "Target");
            CHECK_PARSER_ERR(parser_runtimeargs_getData("target", &dataLength, &datatype,
num_items, ctx))
            return parser_display_runtimeArg(datatype, dataLength, ctx,
                                             outVal, outValLen,
                                             pageIdx, pageCount);
        }
        // ...
    }
    // ...
}
```

Figure 1.2: The abbreviated definition of the `parser_getItem_Transfer` *function,, which calls* `parser_runtimeargs_getData` *in several places (*[`ledger-casper/app/src/parser.c:284-367`]*)*

This issue is exploitable only if the `num_items` argument is greater than 255. However, because user-controlled input is parsed through the `ledger-casper` parsers, this value currently cannot be larger than four.

Recommendations

Short term, change the type of the index variable in the `for` loop in `parser_runtimeargs_getData` from a `uint8_t` to a `uint32_t` so that it is the same type as the loop bound variable. Alternatively, add an error-checking code path that returns early if the loop bound variable is greater than 255. Either of these approaches should prevent the parser from looping infinitely.

Long term, regularly review the results from static analysis tools such as `clang-tidy` and `cppcheck`; these tools can reveal problems like infinite loops and mismatches between integer types in the code. In particular, the `bugprone-too-small-loop-variable` check from `clang-tidy` identifies the code problem at the root of this issue.

References

- Extra Clang Tools 13 Documentation
- danmar/cppcheck
- Developing Secure Ledger Apps Documentation

2. Use of uninitialized memory when ZEMU_LOGGING is defined

Severity: Informational
Type: Undefined Behavior
Target: `ledger-casper/app/src/addr.c`

Difficulty: N/A
Finding ID: TOB-CLL-003

Description

In a build in which the `ZEMU_LOGGING` macro is defined, the `addr_getItem` function calls the `zemu_log_stack` function with uninitialized memory, invoking undefined behavior.

Based on other uses of `zemu_log_stack` in the codebase, it appears that the two lines highlighted in figure 2.1 should be swapped.

```
zxerr_t addr_getItem(int8_t displayIdx,
                     char *outKey, uint16_t outKeyLen,
                     char *outVal, uint16_t outValLen,
                     uint8_t pageIdx, uint8_t *pageCount) {
    char buffer[300];
    zemu_log_stack(buffer);
    array_to_hexstr(buffer, sizeof(buffer), G_io_apdu_buffer, SECP256K1_PK_LEN);

    // ...
}
```

Figure 2.1: The abbreviated definition of the `addr_getItem` *function*
([`ledger-casper/app/src/addr.c:33-59`]*)*

```
void zemu_log_stack(const char *ctx) {
#if defined(ZEMU_LOGGING) && (defined (TARGET_NANOS) || defined(TARGET_NANOX))
#define STACK_SHIFT 20
    void* p = NULL;
    char buf[70];
    snprintf(buf, sizeof(buf), "|SP| %p %p (%d) : %s\n",
            &app_stack_canary,
            ((void*)&p)+STACK_SHIFT,
            (uint32_t)((void*)&p)+STACK_SHIFT - (uint32_t)&app_stack_canary,
            ctx);
    zemu_log(buf);
#else
    (void)ctx;
#endif
}
```

Figure 2.2: The definition of the `zemu_log_stack` *function, which emits log messages when the* `ZEMU_LOGGING` *macro is defined*

Recommendations

Short term, revise the `addr_getItem` function so that `zemu_log_stack` is called as intended.

3. Limited effectiveness of the transaction parser fuzz target

Severity: Informational
Difficulty: N/A
Type: Testing
Finding ID: TOB-CLL-004
Target: `Various in ledger-casper`

Description
The following issues limit the effectiveness of the existing fuzz target for the transaction parser:

1. The `run-fuzzers.py` and `run-fuzz-crashes.py` scripts are duplicated in the source tree, with nonfunctional versions in the root `ledger-casper` directory and working versions in the `ledger-casper/fuzz` subdirectory.
2. A comma is missing from line 33 of the `run-fuzzers.py` script. As a result, the specified maximum input length for the fuzzer is ignored.
3. Because seed corpora are not provided, the fuzzing coverage is poor even after the fuzzers are run for 24 hours.

The Zondax team informed us that it does, in fact, provide seed corpora for their fuzzer internally.

Recommendations
Short term, eliminate the nonfunctional duplicates of the fuzzing support scripts, add the missing comma to the `run-fuzzers.py` script, and create a seed corpus of well-formed transaction inputs for the transaction parser fuzz target. The developers could add extra logic to the `run-fuzzers.py` script to extract the inputs from the existing manual tests into a seed corpus prior to fuzzing. Implementing these recommendations will improve the effectiveness of the existing fuzzing infrastructure.

Long term, run the `run-fuzzers.py` script for a few minutes in the continuous integration system, and periodically review source-level code coverage results from fuzzing to identify any coverage gaps. That way, the fuzzing infrastructure provides ongoing value.

4. Bug in divBigNumbersWithPrecision function

Severity: Low
Type: Data Validation
Target: `Various in ledger-casper`

Difficulty: High
Finding ID: TOB-CLL-006

Description

The `divBigNumbersWithPrecision` function contains a typo. As shown in figure 4.1, the function contains an `if` statement that checks whether the `convertedDenominator` is greater than or equal to the `convertedDenominator`. Because this will always return `true`, the `else` block will never be evaluated. The `if` statement is supposed to check whether the `convertedNumerator` is greater than or equal to the `convertedDenominator`. Because this function is currently used in the `motesToCurrency` function, this bug could impact currency conversion.

```
export const divBigNumbersWithPrecision = (
  numerator: BigNumber | string,
  denominator: BigNumber | string,
  precision: number = 2
): number => {
  const convertedNumerator = BigNumber.from(numerator);
  const convertedDenominator = BigNumber.from(denominator);
  const precisionMultiplier = Math.pow(10, precision);

  let wholeQuotient, fractionQuotient, remainder;

  if (convertedDenominator.gte(convertedDenominator)) {
    ...
  } else {
    ...
};
```

Figure 4.1: In the `divBigNumbersWithPrecision` function, the highlighted `if` statement will always return `true`, and the `else` block will never be reached.

Exploit Scenario

Eve notices this bug in the `divBigNumbersWithPrecision` function. She then crafts an input that triggers the bug and receives more funds than she is owed in the currency conversion operation.

Recommendations

Short term, correct the typo in the `if` statement in `divBigNumbersWithPrecision` so that it checks whether the `convertedNumerator` is greater than or equal to the `convertedDenominator`.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing a system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Testing	Related to test methodology or test coverage
Timing	Related to race conditions, locking, or the order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices or Defense in Depth.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is relatively small or is not a risk the customer has indicated is important.
Medium	Individual users' information is at risk; exploitation could pose reputational, legal, or moderate financial risks to the client.
High	The issue could affect numerous users and have serious reputational, legal, or financial implications for the client.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is commonly exploited; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of a complex system.
High	An attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components
Arithmetic	Related to the proper use of mathematical operations and semantics
Assembly Use	Related to the use of inline assembly
Centralization	Related to the existence of a single point of failure
Upgradeability	Related to contract upgradeability
Function Composition	Related to separation of the logic into functions with clear purposes
Front-Running	Related to resilience against front-running
Key Management	Related to the existence of proper procedures for key generation, distribution, and access
Monitoring	Related to the use of events and monitoring procedures
Specification	Related to the expected codebase documentation
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.)

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed, and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.
Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

C. Code Quality Recommendations

This appendix contains findings that do not have immediate or obvious security implications.

1. The `crypto.c` file in `ledger-casper` does not check the returned value for each of the following operations:
   a. `cx_blake2b_init2`
   b. `cx_hash`
   c. `cx_ecfp_init_private_key`
   d. `cx_ecfp_init_public_key`
   e. `cx_ecfp_generate_pair`
   f. `cx_hash_sha256`
   g. `cx_ecdsa_sign`
2. The `isDict` function in `ledger-casper/js/src/common.js` will not work for all input types. For instance, if the input is a `RegExp`, this function will incorrectly return `true`. The way this function is currently used does not present any problems, but for completeness, consider [strongly typing a dictionary]() or use the following code snippet instead:

```
function isDict(v: any) {
    return v !== undefined && v.constructor == Object
}
```

3. Throughout `casper-explorer`, there are potential uncaught exceptions in calls made to functions such as `PublicKey.fromHex`. Review each of these instances and consider handling these exceptions more gracefully.
4. There are several links in the `casper-explorer` privacy policy scene that contain `http` URLs. Adjust these links to use `https`.
5. There are multiple instances in which an unnecessary local variable is used in `casper-explorer`:
   a. `state` variable in `apps/web/src/app/services/auth-service.tsx:61`
   b. `signature` variable in `apps/web/src/app/services/ledger-service.tsx:157`
   c. `ev` variable in `libs/ui/src/lib/dropdown/dropdown.tsx:104`
6. Use `crypto.randBytes()` instead of `Math.random` in `apps/web/src/app/subscriptions/casper-service-hooks.tsx:23`.

D. Fix Log

After the initial assessment, Casper and Zondax addressed the discovered issues through pull request #137 to `casper-explorer` and pull request #44 to `ledger-casper`. Trail of Bits verified each fix to ensure that it would correctly address the corresponding issue. The results of this verification and additional details on each fix are provided below.

#	Title	Severity	Status
1	Malformed data could cause the transaction parser to loop infinitely	Informational	Fixed
2	Use of uninitialized memory when ZEMU_LOGGING is defined	Informational	Fixed
3	Limited effectiveness of the transaction parser fuzz target	Informational	Partially fixed
4	Bug in divBigNumbersWithPrecision function	Low	Fixed

Detailed fix log

TOB-CLL-002: Malformed data could cause the transaction parser to loop infinitely
Fixed. The type of the `index` variable in the `for` loop has been changed to a `uint32_t`, which prevents the loop from looping infinitely.

TOB-CLL-003: Use of uninitialized memory when ZEMU_LOGGING is defined
Fixed. The `add_getItem` function has been revised so that `zemu_log_stack` is not called with uninitialized memory.

TOB-CLL-004: Limited effectiveness of the transaction parser fuzz target
Partially fixed. The Zondax team indicated that it provides seed corpora internally. However, the typo in `run-fuzzers.py` has not been fixed.

TOB-CLL-006: Bug in divBigNumbersWithPrecision function
Fixed. This portion of the codebase has been refactored, and the function containing the bug has been removed.

Appendix C: Code Quality Recommendations
The following code quality recommendations have been addressed:

- Issue 1: There are now checks on the return values of certain functions listed in this issue. The other functions' return values appear to be meaningless.

- Issue 3: The code now uses `isValidPublicKey` in all instances in which public keys are initialized from external entry points. This ensures that exceptions are not thrown every time `fromHex` is used.
- Issue 4: Casper changed all `http` URLs to `https`.
- Issue 5: All unnecessary local variables have been removed.
- Issue 6: The codebase now uses `crypto.randomInt` instead of `Math.random`.