COSMOS APPLICATION SECURITY AUDIT

Zondax

07 January 2026
Version: 1.0

Document Owner: Olivier Kopp

Kudelski Labs - Nagravision sàrl
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

DOCUMENT PROPERTIES

Version:	1.0
File Name:	20260107_Ledger_Zondax_Cosmos_CR_Klabs.pdf
Publication Date:	07 January 2026
Confidentiality Level:	Confidential
Document Owner:	Olivier Kopp
Document Recipient:	Sourajyoti Gupta
Document Status:	Draft

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY 5
1.1 Context and Scope 5
1.2 Main outcomes 5
1.3 Decision 5
1.4 Ledger Checklist 6
2. AUDIT SPECIFICATION 7
2.1 Application privileges 7
2.1.1 Application flags 7
2.1.2 Derivation paths 7
2.1.3 Cryptographic curves 8
2.1.4 Chain ID parameter 8
2.2 Signing 9
2.2.1 Clear signing 9
2.2.2 Blind signing 9
2.3 Compilation 10
2.3.1 Compilation warnings 10
2.3.2 Silenced warnings 10
2.4 Continuous integration 11
2.5 Tests 12
2.5.1 Unit tests 12
2.5.2 Integration tests 12
2.6 Fuzzing 13
2.7 Static analysis 14
2.7.1 Scan-build 14
2.7.2 CppCheck 14
2.7.3 Semgrep 16
2.7.4 CodeQL 16
2.8 Vulnerabilities and weaknesses 17
2.8.1 Vulnerabilities 17
2.8.2 Weaknesses 18
2.9 Cryptography 20

2.9.1 Secrets management 20
2.9.2 Cryptographic operations 20
3. CONCLUSION 21

1. EXECUTIVE SUMMARY

1.1 Context and Scope

This report describes the results of the security evaluation of the following application:

- Cosmos application (2edfc87)

As the last review was performed a long time ago, we decided to fully review the application. Only the swap part will not be reviewed as it has already been checked by Ledger.

The evaluation was performed by Kudelski Labs in Cheseaux-sur-Lausanne, Switzerland, between 22 December 2025 and 7 January 2026 with a total effort of 3.5 labour-days.

1.2 Main outcomes

The audit identified the following outcomes:

Security vulnerabilities: ☑ **0 vulnerability.**

ID	SEVERITY	DESCRIPTION

Weaknesses: ⚠ 3 **weaknesses.**

WEAKNESS	DESCRIPTION
Potential null pointer dereference	Validity of pointers should be checked before manipulating them.
Magic number	Hard coded constants lead to a harder to understand and maintain code base. Magic numbers should be replaced by variables or macros for better maintainability.
Missing bound checks	Every function that modifies or accesses a buffer should check whether the changes made or the accessed data are within the bounds of said buffer.

Unresolved tests: ☑ **0 failed integration tests** (out of 135).

☑ **0 failed unit tests** (out of 141).

1.3 Other Observations

- Some warnings have been silenced but removing them does not trigger any additional warnings during the compilation.

1.4 Decision

We consider that this application **is fit for production.** This decision is based on the checklist in the next section.

1.5 Ledger Checklist

☑	Application flags, derivation paths and curves are correctly set.
☑	Clear signing correctly displays information and asks for user confirmation.
☑	Blind signing is disabled by default or does not exist (if it exists, it does apply for complex transactions but not to coin transfers or swaps).
☑	App workflow is considered secure and snapshots cover the nominal and error cases of app features.
☑	Boilerplate Makefiles and SDK are used to compile the app. If it's a new app, it has been developed over the C/Rust boilerplates.
☑	The app compiles without errors or warnings (no errors/warnings are silenced using compilation flags or pragma directives)
☑	Guidelines enforcer and build workflows are integrated and passed.
☑	Unit tests, functional tests and fuzzing tests are passing.
—	For app-plugins, the generic fuzzing target is integrated.
☑	scan-build/CodeQL or clippy succeed without returning vulnerabilities
☑	Dependencies are needed, maintained and secure (no known impactful vulnerability for the used version).
☑	The app does not contain any medium or high severity vulnerabilities.
☑	The review process looked into technical vulnerabilities as well as logical ones and weaknesses.
☑	Secrets are correctly erased from memory, not exported or shown to the user.
—	User supplied messages use a prefix and are validated before signature.
☑	No cryptographic primitives are re-implemented if they already exist in BOLOS or the SDK.
☑	The client is not able to freely manipulate the device keypairs.
☑	An authenticated encryption scheme is used for data being cached on the client.

2. AUDIT SPECIFICATION

This section follows Ledger's public audit specification that was established to perform a security audit following Ledger's standards.

2.1 Application privileges

- *Application flags, derivation paths and curves are correctly set.*

2.1.1 Application flags

The applications flags used are defined in the `app/pkg/installer_flex.sh`, `app/pkg/installer_s2.sh`, `app/pkg/installer_stax.sh`, `app/pkg/installer_x.sh` files (generated during compilation), either as:

```
LOAD_PARAMS=(... --appFlags 0x800 ...)
```

which enables the following privilege for the Nano S Plus:

1. APPLICATION_FLAG_LIBRARY (0x800)

This flag makes the application behaves (or rather also behave) as a library that exports functions that can be reached from other applications.

or as:

```
LOAD_PARAMS=(... --appFlags 0xa00 ...)
```

for the Flex, Apex, Stax, and Nano X devices, which also enables the following privilege:

1. `APPLICATION_FLAG_BOLOS_SETTINGS (0x200)`

This flag enables the application to read and modify system parameters such as the device's name.

Note that, according to Ledger's specification[1], the application flags should have been defined in the application's Makefile and any flag other than the bolos_settings should be justified in the Makefile.

2.1.2 Derivation paths

The derivation path is specified in the file `app/Makefile` as:

```
APPPATH =  "44'/118'" "44'/60'"
$(info PATHS LIST = $(APPPATH))
PATH_APP_LOAD_PARAMS = $(APPPATH)
```

- The coin type of Cosmos in BIP44[2] is indeed 118 (ATOM).

The coin type of Ethereum in BIP44 is 60.

[1] According to https://developers.ledger.com/docs/device-app/references/app-permissions
[2] https://github.com/satoshilabs/slips/blob/master/slip-0044.md

2.1.3 Cryptographic curves

The curves used are correctly defined in the file `app/Makefile` as:

```
CURVE_APP_LOAD_PARAMS = secp256k1
```

2.1.4 Chain ID parameter

According to the security guideline of Ledger[3], if the application derives keys on the hardened path 44'/60', then the chain ID parameter must be different from `0` or `1`.

This is check inside the application in two different places, first in *app/src/parser.c*:

```
if (container.screen.titlePtr != NULL &&
      container.screen.contentPtr != NULL) {
    if (!strncmp(container.screen.titlePtr, "Chain id",
                 container.screen.titleLen)) {
      if (!strncmp(container.screen.contentPtr, "0",
                   container.screen.contentLen) ||
          !strncmp(container.screen.contentPtr, "1",
                   container.screen.contentLen)) {
        return parser_unexpected_chain;
      }
    }
  }
```

And a second time in *app/src/tx_display.c*:

```
if (strcmp(outVal, COIN_DEFAULT_CHAINID) == 0) {
    // If we don't match the default chainid, switch to expert mode
    display_cache.is_default_chain = true;
    zemu_log_stack("DEFAULT Chain ");
  } else if ((outVal[0] == 0x30 || outVal[0] == 0x31) && strlen(outVal) == 1) {
    zemu_log_stack("Not Allowed chain");
    return parser_unexpected_chain;
  } else {
    zemu_log_stack("Chain is NOT DEFAULT");
  }
```

In both places, the chain id is checked against the values 0 and 1, and an error is issued if one of them is used.

[3] https://developers.ledger.com/docs/device-app/integration/requirements/security

2.2 Signing

2.2.1 Clear signing

- *Clear signing correctly displays information and asks for user confirmation.*

The cosmos application display several information during the signature process. Typically, we find:

- The chain ID
- The account ID
- The type of transaction
- The Fee
- The Gas cost
- The delegator/validator
- Optionally an ascii message

2.2.2 Blind signing

- *Blind signing is disabled by default or does not exist (if it exists, it does apply for complex transactions but not to coin transfers or swaps).*

We did not see any blind signing in the application, apart from the one used with swap.

2.3 Compilation

- *Boilerplate Makefiles and SDK are used to compile the app. If it's a new app, it has been developed over the C/Rust boilerplates.*
- *The app compiles without errors or warnings (no errors/warnings are silenced using compilation flags or pragma directives)*

2.3.1 Compilation warnings

No warnings were raised during the compilation for any of the architecture.

2.3.2 Silenced warnings

A few warnings are silenced in the codebase:

- Misc-no-recursion in app/src/tx_parser.c
- Modernize-deprecated-headers in app/src/tx_parser.h
- Misc-no-recursion in app/src/tx_display.c
- Bugprone-branch-clone in app/src/tx_display.c

Removing these lines does not generate any additional warning.
They should still be removed to avoid missing any new warning generated by a code update.

2.4 Continuous integration

- *Guidelines enforcer and build workflows are integrated and passed.*

Both the guidelines enforcer and the build workflows are integrated and passing successfully.

2.5 Tests

- *Unit tests, functional tests and fuzzing tests are passing.*
- App workflow is considered secure and snapshots cover the nominal and error cases of app features.

2.5.1 Unit tests

A total of 141 unit tests are written across 6 tests suites. All of them are passing successfully:

```
[----------] Global test environment tear-down
[==========] 141 tests from 6 test suites ran. (32 ms total)
[  PASSED  ] 141 tests.
```

2.5.2 Integration tests

The application uses Zemu to execute the integration tests. 135 tests are written across 3 tests suites. All of them are passing successfully:

```
Test Suites: 3 passed, 3 total
Tests:       135 passed, 135 total
Snapshots:   0 total
Time:        330.612 s
Ran all test suites.
Done in 335.88s.
```

2.6 Fuzzing

- *For app-plugins, the generic fuzzing target is integrated.*

Fuzzing is not required for applications.

2.7 Static analysis

- *scan-build/CodeQL or clippy succeed without returning vulnerabilities*
- *Dependencies are needed, maintained and secure (no known impactful vulnerability for the used version).*
- *The app does not contain any medium or high severity vulnerabilities.*

2.7.1 Scan-build

Scan build did not identify any problems:

```
scan-build: Using '/usr/lib/llvm-14/bin/clang' for static analysis
...
scan-build: Analysis run complete.
scan-build: Removing directory '/tmp/scan-build-2025-12-22-084349-15584-1' because it
contains no reports.
scan-build: No bugs found.
```

2.7.2 CppCheck

Cppcheck (V2.7) did not find any vulnerability and only highlighted some false positives:

```
Checking app/src/addr.c ...
1/19 files checked 1% done
Checking app/src/apdu_handler.c ...
app/src/apdu_handler.c:137:7: error: There is an unknown macro here somewhere.
Configuration is required. If ZEMU_LOGF is a macro then please configure it. [unknownMacro]
      ZEMU_LOGF(50, "Chain config not supported for: %s\n", bech32_hrp)
      ^
Checking app/src/apdu_handler.c: DEBUG...
Checking app/src/apdu_handler.c: HAVE_SWAP...
Checking app/src/apdu_handler.c: LEDGER_SPECIFIC...
Checking app/src/apdu_handler.c: TARGET_APEX_P;TARGET_FLEX;TARGET_STAX...
2/19 files checked 8% done
Checking app/src/cbor/cbor_parser_helper.c ...
3/19 files checked 12% done
Checking app/src/chain_config.c ...
4/19 files checked 13% done
Checking app/src/common/actions.c ...
5/19 files checked 14% done
Checking app/src/common/main.c ...
Checking app/src/common/main.c: HAVE_NBGL...
Checking app/src/common/main.c: HAVE_NBGL;HAVE_SWAP...
Checking app/src/common/main.c: HAVE_SWAP...
6/19 files checked 16% done
Checking app/src/common/tx.c ...
Checking app/src/common/tx.c: COMPILE_TEXTUAL...
Checking app/src/common/tx.c: HAVE_SWAP...
Checking app/src/common/tx.c: LEDGER_SPECIFIC...
Checking app/src/common/tx.c: TARGET_APEX_P;TARGET_FLEX;TARGET_NANOS2;TARGET_STAX...
Checking app/src/common/tx.c: TARGET_NANOX...
7/19 files checked 20% done
Checking app/src/crypto.c ...
8/19 files checked 26% done
Checking app/src/json/json_parser.c ...
Checking app/src/json/json_parser.c: APP_TESTING...
Checking app/src/json/json_parser.c: LEDGER_SPECIFIC...
Checking app/src/json/json_parser.c: TARGET_APEX_P;TARGET_FLEX;TARGET_STAX...
9/19 files checked 32% done
Checking app/src/parser.c ...
```

```
app/src/parser.c:580:3: error: There is an unknown macro here somewhere. Configuration is
required. If CHECK_APP_CANARY is a macro then please configure it. [unknownMacro]
  CHECK_APP_CANARY()
  ^
Checking app/src/parser.c: COMPILE_TEXTUAL...
Checking app/src/parser.c: LEDGER_SPECIFIC...
Checking app/src/parser.c: TARGET_APEX_P;TARGET_FLEX;TARGET_STAX...
10/19 files checked 47% done
Checking app/src/parser_impl.c ...
Checking app/src/parser_impl.c: COMPILE_TEXTUAL...
app/src/parser_impl.c:192:5: error: There is an unknown macro here somewhere. Configuration
is required. If CHECK_PARSER_ERR is a macro then please configure it. [unknownMacro]
    CHECK_PARSER_ERR(cbor_check_expert(&data, &container))
    ^
Checking app/src/parser_impl.c: LEDGER_SPECIFIC...
Checking app/src/parser_impl.c: TARGET_APEX_P;TARGET_FLEX;TARGET_STAX...
11/19 files checked 52% done
Checking app/src/secret.c ...
Checking app/src/secret.c: APP_SECRET_MODE_ENABLED...
12/19 files checked 53% done
Checking app/src/swap/handle_check_address.c ...
13/19 files checked 55% done
Checking app/src/swap/handle_get_printable_amount.c ...
14/19 files checked 57% done
Checking app/src/swap/handle_sign_transaction.c ...
15/19 files checked 70% done
Checking app/src/swap/swap_utils.c ...
app/src/swap/swap_utils.c:79:3: error: There is an unknown macro here somewhere.
Configuration is required. If CHECK_ZXERR is a macro then please configure it.
[unknownMacro]
  CHECK_ZXERR(z_str3join(out, out_len, "", PIC(chains[chain_index].ticker)))
  ^
16/19 files checked 73% done
Checking app/src/tx_display.c ...
Checking app/src/tx_display.c: APP_TESTING...
Checking app/src/tx_display.c: LEDGER_SPECIFIC...
Checking app/src/tx_display.c: TARGET_APEX_P;TARGET_FLEX;TARGET_STAX...
17/19 files checked 89% done
Checking app/src/tx_parser.c ...
app/src/tx_parser.c:220:7: error: There is an unknown macro here somewhere. Configuration
is required. If CHECK_APP_CANARY is a macro then please configure it. [unknownMacro]
      CHECK_APP_CANARY()
      ^
Checking app/src/tx_parser.c: LEDGER_SPECIFIC...
Checking app/src/tx_parser.c: TARGET_APEX_P;TARGET_FLEX;TARGET_STAX...
18/19 files checked 96% done
Checking app/src/tx_validate.c ...
Checking app/src/tx_validate.c: LEDGER_SPECIFIC...
Checking app/src/tx_validate.c: TARGET_APEX_P;TARGET_FLEX;TARGET_STAX...
19/19 files checked 100% done
```

2.7.3 Semgrep

Semgrep (V1.146.0) did not identify any vulnerability:

```
✅ Scan completed successfully.
 • Findings: 0 (0 blocking)
 • Rules run: 104
 • Targets scanned: 38
 • Parsed lines: ~98.2%
 • Scan was limited to files tracked by git
Ran 104 rules on 38 files: 0 findings.
```

2.7.4 CodeQL

Long switch case :

- */app/src/tx_display.c*:247
- */app/src/tx_parser.c*:206

Commented-out code :

- */app/src/tx_display.c*:18, 67
- */app/src/tx_parser.h*:20
- */app/src/tx_parser.c*:18

Local variable address stored in non-local memory :

- */app/src/tx_display.c*:117, 224, 519

2.8 Vulnerabilities and weaknesses

- *The app does not contain any medium or high severity vulnerabilities.*
- *The review process looked into technical vulnerabilities as well as logical ones and weaknesses.*
- App workflow is considered secure and snapshots cover the nominal and error cases of app features.

2.8.1 Vulnerabilities

No vulnerabilities were found during the manual review of the C Codebase.

2.8.2 Weaknesses

Potential null pointer dereference

Validity of pointers should be checked before manipulating them. This occurs in the following files:

- *app/src/addr.c:27,38*
- *app/src/tx_validate.c:42,101*
- *app/src/ secret.c:33*
- *app/src/crypto.c:156*
- *app/src/parser.c:32,52,63,83,109,125,159,189,284,360,560*
- *app/src/parser_impl.c:135*
- *app/src/tx_display.c:443,629*
- *app/src/tx_parser.c:34,78*
- *app/src/tx_parser.h:64,68*
- *app/src/json/json_parser.c:39,88,119,154,186,221,235*

Notable examples:

```
__Z_INLINE bool is_msg_type_field(char *field_name) {
  return strcmp(field_name, "msgs/type") == 0;
}

__Z_INLINE bool is_msg_from_field(char *field_name) {
  return strcmp(field_name, "msgs/value/delegator_address") == 0;
}
```

In this case, if a null pointer is passed to the functions above, the behavior is undefined and other bugs could arise.

```
parser_error_t parser_getNumItems(const parser_context_t *ctx,
                                  uint8_t *num_items) {
  *num_items = 0;
  if (ctx->tx_obj->tx_type == tx_textual) {
[...]
```

In this second case, if any of the parameter is Null, the application will crash when trying to access them.

Magic number

Hard coded constants lead to a harder to understand and maintain code base. Magic numbers should be replaced by variables or macros for better maintainability. This occurs in the following files:

app/src/parser_txdef.h:46

app/src/addr.c:56

*app/src/crypto.c:*156,187,215,245

*app/src/parser.c:*71,72,217

Missing bound checks

Every function that modifies or accesses a buffer should check whether the changes made or the accessed data are within the bounds of said buffer.

Notable examples:

app/src/tx_validate.c :78

app/src/apdu_handler.c:78

```
__Z_INLINE uint8_t extractHRP(uint32_t rx, uint32_t offset) {
  uint8_t hrp_len = 0;
  if (rx < offset + 1) {
    THROW(APDU_CODE_DATA_INVALID);
  }
  MEMZERO(bech32_hrp, MAX_BECH32_HRP_LEN);

  bech32_hrp_len = G_io_apdu_buffer[offset];
[...]
```

No checks are made in the above code to ensure that the offset is within the maximum range of the buffer. A too large offset might result in a buffer overread.

2.9 Cryptography

2.9.1 Secrets management

- *Secrets are correctly erased from memory, not exported or shown to the user.*

All secrets are correctly erased from memory, using the explicit_bzero function that will never be optimized by the compiler.

2.9.2 Cryptographic operations

- *User supplied messages use a prefix and are validated before signature.*
- *No cryptographic primitives are re-implemented if they already exist in BOLOS or the SDK.*
- *The client is not able to freely manipulate the device keypairs.*
- *An authenticated encryption scheme is used for data being cached on the client.*

During the review, we did not see any cryptographic primitive being re-implemented inside the Cosmos application.

All the messages use a specific structure, and no user-provided message can be signed. Only the “memo” part of the message can be arbitrary.

3. CONCLUSION

The cosmos application does not contain any vulnerability, and the automated tools did not highlight any finding. Three weaknesses have been identified in the codebase. Based on these results, the latest Cosmos version can be integrated to the Ledger repository.

DOCUMENT RECIPIENTS

NAME	POSITION	CONTACT INFORMATION
Sourajyoti Gupta	Program Manager	sourajyoti.gupta@zondax.ch
Ainhoa Aldave	Chief Operating Officer	ainhoa.aldave@zondax.ch

KUDELSKI LABS CONTACTS

NAME	POSITION	CONTACT INFORMATION
Olivier Kopp	Senior Security Engineer	olivier.kopp@nagra.com
Stéphane Jullian	Senior Operations Manager	stephane.jullian@nagra.com

ACRONYMS

ACRONYM	DEFINITION

DOCUMENT HISTORY

AUTHOR	STATUS	DATE	VERSION	COMMENTS
Olivier Kopp	Draft	7 January 2026	1.0	
		Select the Date		
		Select the Date		

REVIEWER	POSITION	DATE	VERSION	COMMENTS
Adel Qasem	Sr. Security Engineer	7 January 2026	1.0	
		Select the Date		
		Select the Date		

APPROVER	POSITION	DATE	VERSION	COMMENTS
Stéphane Jullian	Sr. Operations Manager	7 January 2026	1.0	
		Select the Date		
		Select the Date		