HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SerenityShield
Date: April 6, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for SerenityShield
Approved By	Yevheniy Bezuhlyi | SC Audits Head at Hacken OU
Type	Secret Store
Platform	Secret Network
Language	Rust
Methodology	Link
Website	https://serenityshield.io
Changelog	14.03.2023 - Initial Review 06.04.2023 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by SerenityShield (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project includes review and security analysis of the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/serenityshield/strongbox-contract
Commit	b919434f50bf0ae0e0fa057d5edb54917ecbffa2
Whitepaper	Link
Functional Requirements	Not provided
Technical Requirements	./Publishing.md ./Importing.md ./Developing.md ./README.md
Contracts	File: ./src/contract.rs SHA3: 1bfbf8d18db3d6b237e70c7e873f000fc7a9758da68c0c455233645e37e87662 File: ./src/lib.rs SHA3: 8c667e801a8acb3277ae534611c5d4211462c3f4a04cbe56a2744a5135182ad9 File: ./src/msg.rs SHA3: f180222e65af92cdfd780f30515b6056e72c3c86df99308f5ff32b82e9ae9f66 File: ./src/state.rs SHA3: 240d11227aa876c58b426e6b753f28cb1360081c0d9506ce34c5f8ff29da145b File: ./src/utils.rs SHA3: ec55d563141968054b065f6bfad927236dd6c01b995173a765c39a2ff0b48c59 File: ./src/viewing_key.rs SHA3: 72f95227d45daae288d1895d7544704905d89b2822db0fb33caffc86c1a19ae6

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Second review scope

Repository	https://github.com/serenityshield/strongbox-contract
Commit	65ed704631c4e001e04763ce2516fe7787ca686e
Whitepaper	Link
Functional Requirements	./README.md
Technical Requirements	./Install.md
Contracts	File: ./src/contract.rs SHA3: 4cd77c6df507b7dafc3daafa18c62a5225823dd3e11058e2727e7b7f1bb3e527 File: ./src/lib.rs SHA3: 947e1d1fec864f8af181628fce77e2ffea2fdb1afd5d961172102be0d05f5e39 File: ./src/msg.rs SHA3: 30da6837c13448634480d0347f5fd47f888001fd89091e6da2b44208d50defc0 File: ./src/state.rs SHA3: 0c2dfa763531b14d414bcc85a956fa561aa8e4ed5b1a749d563eb84e23907112 File: ./src/viewing_key.rs SHA3: 17336a4d20eac1335a03e352500541ee2e30b0a26fc79b878b7fbc13b3b26c5c

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

- There are formatting issues in ./README.md; for example (as it looks on Github): • InstantiateThis function
- ./README.md, as the entry point to the documentation, should refer to ./Install.md, because otherwise, a reader has no way of knowing about that document except searching for all possible documentation files manually.
- ./Install.md refers to undefined *cargo unit-test* and *cargo schema* commands.
- ./Install.md should list *clang* as a prerequisite, because it may not come out of the box on some OS.

Code quality

The total Code Quality score is **8** out of **10**.

- Redundant code was found.
- Readability issues were found.
- There are no documentation comments in the code.
- Development environment is configured.

Test coverage

Code coverage of the project is **92.71%** (line coverage by *cargo-llvm-cov*).

Security score

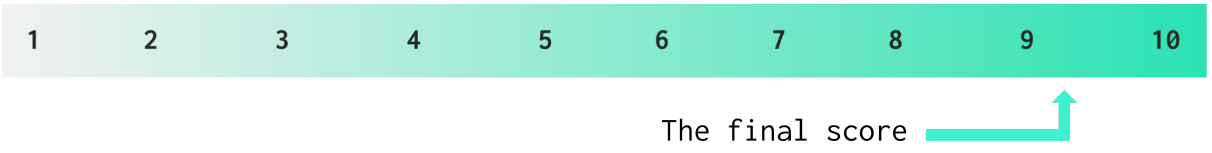
As a result of the audit, the code contains **2** low severity issues. The security score is **10** out of **10**.

All found issues are displayed in the Findings section of the report.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.2**.

The system users should acknowledge all the risks summed up in the Risks section of the report.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
March 14, 2023	7	5	1	0
April 6, 2023	2	0	0	0

Risks

- The validity of the contract is predicated on the following guarantees by Secret Network:
  - Only the contract can decrypt its state.
  - Only the contract and the caller can see the call arguments and the result.
- It is a known Secret Network issue that data stored at a moment in the past can be retrieved with new viewing keys and actual stored data can be retrieved with removed viewing keys.
- If someone manages to deploy a contract (e.g. in an ad-hoc side chain) with the same Secret Network Contract Key as an instance of the Strongbox contract, they would be able to decrypt the Strongbox instance state. It is a known theoretical attack in Secret Network, and is hardly executable.
- The contract owner may allow anyone to read the stored data.
- The contract owner is able to corrupt stored data.
- The contract owner is able to modify viewing keys, and revoke access from the old ones.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

StrongBox is a smart contract on *Secret Network* which allows users to store their private data on-chain safely.

- `try_update_strongbox` – provides an ability to store some data (only contract owner)
- `try_create_viewing_key` – provides an ability to create keys to view the data (only contract owner)
- `try_transfer_ownership` – provides an ability to transfer contract ownership (only contract owner)
- `try_revoke_viewing_key` – provides an ability to remove a viewing key for a given address (only contract owner)
- `query` – provides an ability to receive stored data using generated keys

Privileged roles

Smart contract owner is able to:

- update data stored at StrongBox
- create, change or revoke a key that allows viewing the data stored at StrongBox
- transfer the ownership to another address
- view the data stored at StrongBox

Viewing key owner:

- view the data stored at StrongBox

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Unchecked Call Return Value	The return value of a message call should be checked.	Passed
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Rust Functions	Deprecated built-in functions should never be used.	Passed
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used.	Not Relevant
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Passed
Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Not Relevant
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Not Relevant
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Not Relevant
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Not Relevant

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Data Consistency	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Not Relevant
Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Compiler Warnings	The code should not force the compiler to throw warnings.	Passed
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	The code should not reference draft contracts, that may be changed in the future.	Passed
Unsafe Rust code	The Rust type system does not check memory safety of unsafe Rust code. Thus, if a smart contract contains any unsafe Rust code, it may still suffer from memory corruptions such as buffer overflows, use after frees, uninitialized memory, etc.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

No critical severity issues were found.

High

H01. Weak Source of Randomness

It is not checked that the entropy / initial seed is secure enough. The functions should check the length of entropy / initial seed to make sure that the generated viewing keys are strong.

This may lead to the data being viewed by an unauthorized user.

In addition, the “human” address of the sender is used as an input material for the viewing key. “Human” representation may not be stable.

Path: src/contract.rs: try_create_viewing_key(), init()

Recommendation: assess the quality of the entropy / initial seed (e.g. check the length). Use the canonical sender address representation as a material for viewing keys.

Found in: b919434

Status: Fixed (Revised commit: 65ed704)

Medium

M01. Denial of Service Vulnerability

The human-readable address representation is used for identity checking of the contract owner.

The human-readable representation may not be stable and larger than the canonical representation. It is safer and more optimal to use the canonical representation.

Path: src/state.rs: struct State

Recommendation: use the `CanonicalAddr` type for storing data in state.

Found in: b919434

Status: Fixed (Revised commit: 65ed704)

M02. Immutable Ownership

The contract is designed in a way that the ownership cannot be transferred.

This may lead to impossibility of updating the owner in critical situations.

Path: src/contract.rs

Recommendation: implement an ability to transfer the contract ownership.

Found in: b919434

Status: Fixed (Revised commit: 65ed704)

M03. Immutable Viewing Keys

The contract is designed in a way that it is impossible to directly revoke / renounce a viewing key. However, the owner is able to recreate a viewing key for the viewer, the number of keys does not decrease and the contract may be overwhelmed with the keys.

This may lead to a lot of permissive viewing keys being registered on the contract.

Path: src/contract.rs

Recommendation: implement an ability to revoke / renounce a viewing key.

Found in: b919434

Status: Fixed (Revised commit: 65ed704)

M04. Weak Source of Randomness

It is possible to recover the length of the initial seed, and the viewing key entropy by analyzing the run-time of the functions *`init()`*, *`try_create_viewing_key()`* respectively. The secrets' length affects the run-times of the functions. The run-times can be compared and the duration-length relation may be inferred.

This may lead to the secrets' length being decoded.

Path: src/contract.rs: init(), try_create_viewing_key()

Recommendation: make the methods' run-time independent of the initial seed / entropy length.

Found in: b919434

Status: Fixed (Revised commit: 65ed704)

M05. Weak Source of Randomness

The owner may reuse the viewing key entropy for different parties (e.g. accidentally), because there is no check that a viewing key entropy has not been used before generating a key from it.

It could also lead to identical viewing keys being generated for different parties if the key creation calls are done in the same block.

Path: src/contract.rs: try_create_viewing_key()

Recommendation: implement a check that a viewing key entropy has not been used before generating a key from it.

Found in: b919434

Status: Fixed (Revised commit: 65ed704)

Low

L01. Redundant Code

- Module *contract*:
  - line 33: needless borrow at *&general_purpose::STANDARD.encode(&initial_seed).as_bytes()*
  - line 95: needless borrow at *&entropy.as_bytes()*
  - line 107: needless borrow at *(&entropy)*
  - line 189: needless *return*
  - line 47: redundant *..*
  - line 33: redundant conversion to base64
  - function *query(..)*: the abstraction over *QueryMsg* kinds, provided by *QueryMsg::get_validation_params(..)* is redundant, since there's only one kind of *QueryMsg*, which is *GetStrongbox*. The loop on lines 167-181 could be replaced with its body for the single possible address: *QueryMsg::GetStrongbox::behalf*
- Module *viewing_key*:
  - line 23: needless borrow at *&self.as_bytes()*
  - line 32: needless borrow at *&env.block.time.to_string().as_bytes()*
  - line 42: needless borrow at *&key*
  - line 32: redundant conversion to string at *env.block.time.to_string()*, which causes reallocation of *rng_entropy*, because the length of result string is 20 (as of the time of writing), but *rng_entropy* has only 8 bytes reserved for the value.
- Module *msg*:
  - line 32: redundant *..*

Paths: ./src/contract.rs, ./src/viewing_key.rs, ./src/msg.rs

Recommendation: eliminate the mentioned redundancies.

Found in: b919434

Status: Reported

L02. Vulnerable Dependency

Vulnerability info: https://rustsec.org/advisories/RUSTSEC-2021-0076

Dependency path:

```
libsecp256k1 0.3.5
<- secret-toolkit-crypto 0.1.0
<- secret-toolkit 0.1.0
<- serenty_shield 0.1.0
```

Path: Cargo.toml

Recommendation: upgrade *libsecp256k1* to *>=0.5.0* by changing the version of *secret-toolkit*. For example, *secret-toolkit 0.7.0* does not have vulnerabilities in the dependencies.

Found in: b919434

Status: Fixed (Revised commit: 65ed704)

L03. Multiple Library Versions in Dependency Tree

The issue causes bloating of the size of targets, and can lead to confusing error messages when structs or traits are used interchangeably between different versions of a crate. Because this can be caused purely by the dependencies themselves, it is not always possible to fix this issue.

Cases:

- *base64*: 0.13.1, 0.21.0
- *block-buffer*: 0.9.0, 0.10.4
- *digest*: 0.9.0, 0.10.6
- *rand_core*: 0.5.1, 0.6.4
- *sha2*: 0.9.9, 0.10.6
- *syn*: 1.0.109, 2.0.8

Path: Cargo.toml

Recommendation: for some dependencies, it is possible to fix the duplication, but that would mean downgrading some dependencies. Overall, the choice of dependency versions is conscious; therefore, no action is suggested.

Found in: b919434

Status: Mitigated (the issue is informational and does not require any fixes)

L04. Unformatted Code

cargo fmt yields changes in some files.

Path: src/lib.rs, src/utils.rs, src/viewing_key.rs

Recommendation: format the code using *rustfmt* or equivalent.

Found in: b919434

Status: Fixed (Revised commit: 65ed704)

L05. Possible Typo

It looks like *serenty_seed* should be changed to *serenity_seed*.

Path: src/state.rs: struct State

Recommendation: provide conscious variable names.

Found in: b919434

Status: Fixed (Revised commit: 65ed704)

L06. Late Validation

The signer-is-owner check is executed after heavy viewing key generation.

This increases computation cost in the case of check failure.

Path: src/state.rs: try_create_viewing_key()

Recommendation: perform the signer-is-owner check at the start of execution.

Found in: b919434

Status: Fixed (Revised commit: 65ed704)

L07. Outdated Platform SDK/Tools Versions

secret-toolkit (v0.1.0 used), *cosmwasm-** (v0.10.* used) library versions are significantly outdated.

The docker image for the Secret node (v1.2.6 used) and Secret CLI is outdated.

This may lead to missing important bug fixes. This makes the code and dev environment not in line with the contemporary official documentation.

Path: Cargo.toml, Makefile

Recommendation: consider upgrading the platform dependencies.

Found in: b919434

Status: Fixed (Revised commit: 65ed704)

L08. Confusing Naming

- Module *contract*:

    - line 33: The name *general_purpose::STANDARD* is detached from the meaning. It is recommended to use alias import in such cases, for example:

      use base64::engine::general_purpose::STANDARD as base64_std;

- Module *viewing_key*:
    - line 42: The name *general_purpose::STANDARD* is detached from the meaning. It is recommended to use alias import in such cases, for example:

      use base64::engine::general_purpose::STANDARD as base64_std;

  Path: ./src/contract.rs, ./src/viewing_key.rs

  Recommendation: improve the naming according to the suggestions in the description.

  Found in: 65ed704

  Status: New

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.