HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: GotBit
Date: 15 June, 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 GotBit
Approved By	Yevheniy Bezuhlyi | SC Audits Head at Hacken OÜ
Type	Bridge
Platform	Solana
Language	Rust
Methodology	Link
Website	gotbit.io
Changelog	18.05.2023 - Initial Review 07.06.2023 - Second Review 15.06.2023 - Third Review

Table of Contents

Introduction

Hacken OÜ (Consultant) was contracted by GotBit (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.

System Overview

GotBit Bridge – is a system that allows cross-chain token transfers. The system contains the smart contract (*bridge_solana*), which was audited.

The smart contract's main purpose is processing incoming and outcoming transfer requests. The system allows creation of several bridge instances for bridging different tokens.

The smart contract has the following entrypoints:

- *initialize* – creates a new bridge instance.
- *set_params* – configures a given existing bridge instance.
- *set_chain_data* – allows whitelisting chains where the bridge instances are deployed.
- *fulfill* – execute a transfer on this end of the bridge.
- *send* – save transaction details to be processed later by the bridge authority.
- *withdraw* – transfer all tokens from bridge associated account to a specified address.

The bridge fully relies on the owner, the authority for processing outcoming transfer requests (distributing funds to receivers on the chain). The system fully trusts the owner and does not require any additional confirmations except for the owner’s will to distribute funds to users.

The system does not allow to cancel transfer requests and does not provide any guarantees that the transfer request will be fulfilled.

Privileged roles

The bridge instance owner is able to:

- update transfer fees up to 99.99%,
- fulfill or ignore transfer requests,
- withdraw all deposited funds,
- configure bridge destination chains.

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 **9** out of **10**.

- Functional requirements are provided.
- Architecture in Solana implementation differs from Ethereum one making the requirements vague.
- The technical description on how to build and test the project is provided.

Code Quality

The total Code Quality score is **9** out of **10**.

- Functions that process common parameters, accept them differently.
- All the code is implemented in one file making navigation harder.
- Development environment is consistent with the requirements.

Test Coverage

Code coverage of the project is about **95%**.

- Unit tests are not provided.
- Integration test coverage is a manual approximation.

Security Score

As a result of the audit, the code does not contain security 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: **9.5**.

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

1
2
3
4
5
6
7
8
9
10
The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
18 May 2023	2	4	1	0
7 June 2023	1	0	0	0
15 June 2023	0	0	0	0

Risks

- Unless the smart contract is deployed with the *--final* parameter, it could be upgraded and its functionality may be changed.
- The owner is able to **withdraw all the deposited funds** to an arbitrary address.
- The owner is able to set an **unlimited fee** for transactions.
- There are **no guarantees** that a transfer request will be fulfilled.
- Anyone is able to initialize a bridge instance. It is strongly recommended to check if the used bridge instance is initialized by a trusted account and the account is not compromised.
- The owner cannot be changed. In an emergency situation, the only option for the owner is to:
    1. Withdraw all the funds from the bridge.
    2. Abandon the bridge and all the data associated with it (nonces, bridge parameters, chain data and transaction history details will be lost).
    3. Create a fresh new bridge instance with a new account.

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 Errors	If a function returns a Result, it should not be ignored.	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.	Passed
Signature Reuse	Signed messages that represent an approval of an action should not be reusable.	Not Relevant
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed
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.	Passed
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed

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 and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract.	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.	Passed
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. The 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 the 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
Missing rent exemption checks	All Solana accounts holding an Account, Mint, or Multisig must contain enough SOL to be considered rent exempt. Otherwise, the accounts may fail to load.	Not Relevant
Missing freeze authority checks	When freezing is enabled but the program does not verify that the freezing account call has been signed by the appropriate freeze_authority.	Not Relevant

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Denial of Service

Impact	Medium
Likelihood	High

According to the implementation, each user has a nonce counter (there could be several transactions with the same nonce). However, during the fulfillment seeds for *empty_account* contain nonce and do not contain the sender.

This may block the ability to process transfers for most users after the first one is processed and *empty_account* with seed nonce *0* is created.

Path: ./anchor/programs/bridge-solana/src/lib.rs: Fulfill

Recommendation: Provide documentation to the protocol, use a pair of seeds (*sender*, *nonce*) for *empty_account* creation.

Found in: 2f76331

Status: Mitigated (The Ethereum bridge implementation takes the logic into account)

■■ Medium

M01. Eager Division

Impact	Medium
Likelihood	Medium

The functions perform division before multiplication during fee calculation.

This may lead to:

- the calculated fee value being lower than expected,
- if a user requests bridging of lower than 10000 tokens, no fee would be charged.

Path: ./anchor/programs/bridge-solana/src/lib.rs: fulfill(), send()

Recommendation: Perform division after multiplication or define the minimal amount for being processed.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

M02. Immutable Ownership

Impact	High
Likelihood	Low

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

This may lead to the impossibility to update the owner in critical situations.

Path: ./anchor/programs/bridge-solana/src/lib.rs

Recommendation: Implement an ability to transfer contract ownership.

Found in: 2f76331

Status: Mitigated (It is a design decision described in requirements)

M03. Highly Permissive Role

Impact	High
Likelihood	Low

The *bridge_token_account* PDA uses the owner as the authority. That allows the owner to transfer funds from the account directly, calling the SPL Token program.

Storing user funds on an account that allows direct transfers authorized by any third party makes the project consistency significantly lower.

This may lead to user funds being stolen by any contract which the owner calls.

Path: ./anchor/programs/bridge-solana/src/lib.rs: Initialize

Recommendation: Use *bridge_token_account* itself as the authority.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

M04. Undocumented Behavior

Impact	Medium
Likelihood	Medium

The *exchange_rate_from* value highly influences the amount during funds bridging. However, the behavior is not documented.

The exchange rate is used in both *send* and *fulfill* functions, which makes it hard to understand the bridging process.

This may lead to incorrect amounts being bridged.

Path: ./anchor/programs/bridge-solana/src/lib.rs: send(), fulfill()

Recommendation: Provide corresponding documentation or remove the misleading functionality.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

Low

L01. Unlimited Fees

Impact	Medium
Likelihood	Low

Owner is able to set fees without any limitations.

This may lead to users losing all funds during interactions with the project.

Path: ./anchor/programs/bridge-solana/src/lib.rs: initialize(), set_params()

Recommendation: Limit the fee percentages to reasonable values.

Found in: 2f76331

Status: Mitigated (It is a design decision described in requirements)

L02. Leak of Authorization

Impact	Low
Likelihood	Medium

Anyone is able to create a bridge instance on the smart contract.

This may lead to project reputation risks due to malefactors may use the smart contract to grind users.

Path: ./anchor/programs/bridge-solana/src/lib.rs: initialize()

Recommendation: Disallow the ability of bridge creation for untrusted accounts.

Found in: 2f76331

Status: Mitigated (The ability of several bridge instances with different owners creation is justified by design)

Informational

I01. Common Used Value

Commonly used values should be represented as constants to prevent typos and make code reusable.

This may lead to new issues appearing during further development.

Path: ./anchor/programs/bridge-solana/src/lib.rs: 10000

Recommendation: Extract the common value into a constant.

Found in: 2f76331

Status: Fixed (Revised commit: b3a3626)

I02. Assert Violation

Assert statements should be used to validate project invariants.

This may lead to the code failing with a panic instead of an error.

Path: ./anchor/programs/bridge-solana/src/lib.rs: initialize(), set_params(), set_chain_data(), fulfill(), send()

Recommendation: Use *require!(..)* macro instead.

Found in: 2f76331

Status: Fixed (Revised commit: b3a3626)

I03. Outdated Package Metadata

The description is a short blurb about the package. However, the current description is *Created with Anchor*, which does not correctly describe the project.

This may lead to an incorrect description shown to interested at crates.io.

Path: ./anchor/programs/bridge-solana/Cargo.toml

Recommendation: Make the description declarative.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

I04. Floating Language Version

It is preferable for a production project, especially a smart contract, to have the programming language version pinned explicitly. This results in a stable build output, and guards against unexpected toolchain differences or bugs present in older versions, which could be used to build the project.

The language version could be pinned in automation/CI scripts, as well as proclaimed in README or other kinds of developer documentation. However, in the Rust ecosystem, it can be achieved more ergonomically via a *rust-toolchain.toml* descriptor (see https://rust-lang.github.io/rustup/overrides.html#the-toolchain-file)

Path: ./rust-toolchain.toml

Recommendation: Pin the language version at the project level.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

I05. Redundant Functionality

The *_current_chain* value is used as a seed for most of the PDAs throughout the project. However, it is not documented or intuitive of how it could be various and what it is responsible for.

Path: ./anchor/programs/bridge-solana/src/lib.rs

Recommendation: Provide corresponding documentation or remove the functionality as redundant.

Found in: 2f76331

Status: Reported

I06. Redundant Statement

The *instructions_sysvar* account is listed in the *Fulfill* structure but is never used.

The *associated_token_program* account in the structure *Initialize* is never used.

Path: ./anchor/programs/bridge-solana/src/lib.rs: Fulfill, Initialize

Recommendation: Remove redundancies.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

I07. Misleading Name

The *empty_account* field in the *Fullfill* struct does not match its purpose. The existence of this account serves as a boolean flag of the successful completion of a given transaction to prevent double execution.

Path: ./anchor/programs/bridge-solana/src/lib.rs: Fulfill

Recommendation: Rename the field to better reflect the purpose (for example, *is_tx_fullfilled*).

Found in: 2f76331

Status: Reported

I08. Invalid Constraint

The constraints *user_t_a.owner == user.key() && user_t_a.mint == _token_mint.key()* is not enough to validate that the account is a token account and the user is its owner.

An arbitrary account could be passed to the function.

Path: ./anchor/programs/bridge-solana/src/lib.rs: Send

Recommendation: Use *token::mint = _token_mint && token::authority = user* validation.

Found in: 2f76331

Status: Fixed (Revised commit: b3a3626)

I09. Inconsistent Code

Some functions receive *_token_mint*/*_owner* accounts as function parameters directly, others - in the form of *token_mint*/*owner* fields in context structures.

Path: ./anchor/programs/bridge-solana/src/lib.rs

Recommendation: Pass all accounts into context structures.

Found in: f205b8a

Status: Reported

I10. Unformatted Code

cargo fmt yields changes in the file.

Path: ./anchor/programs/bridge-solana/src/lib.rs

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

Found in: f205b8a

Status: Fixed (Revised commit: b3a3626)

I11. Used Variables Marked as Unused

The variables *_version* and *_current_chain* are used inside functions but are named as though they are not being used.

Path: ./anchor/programs/bridge-solana/src/lib.rs: fulfill(..), withdraw(..)

Recommendation: Remove “_” prefix from their names.

Found in: f205b8a

Status: Fixed (Revised commit: b3a3626)

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.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

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.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial Review Scope

Repository	https://github.com/GotBit/bridge
Commit	2f7633185cf94c002480c0700942992551492dfe
Technical Requirements	Link
Functional Requirements	Link
Contracts	File: ./anchor/programs/bridge-solana/src/lib.rs SHA3: a7a45221ce6b786c76dbef722cd31fd3f3e1d67635d22dd8265036309653053b

Second Review Scope

Repository	https://github.com/GotBit/bridge
Commit	f205b8a2df5f1fdebc5291a1a61f99689d428374
Technical Requirements	Link
Functional Requirements	Link
Contracts	File: ./anchor/programs/bridge-solana/src/lib.rs SHA3: efcd919dabdf35e18f0631890ee1dd6c1c8be5fede9f86f8fc690ea0ce44e647

Third Review Scope

Repository	https://github.com/GotBit/bridge
Commit	b3a36262ddfd5c28a2da1e4ca175e7010f19babd
Technical Requirements	Link
Functional Requirements	Link
Contracts	File: ./anchor/programs/bridge-solana/src/lib.rs SHA3: 77ff18ada2c27f3d95b0afe3cf654322e474d0404d4983a3a4c7909c3409c0e7