SolyzerX
BLOCKCHAIN SECURITY

SHIBONK

Security Assesment

MARCH 2023

SolyzerX

SolyzerX.com

Prepared for:

Bonktoshi Bonkamoto
SHIBONK Founder

Prepared by:

Aiman Asyraf
SolyzerX Engineer

Introduction

Auditing Firm	SolyzerX
Client Firm	SHIBONK
Methodology	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Language	Rust
Contract	H1G6sZ1WDoMmMCFqBKAbg9gkQPCo1sKQtaJWz9dHmqZr
Blockchain	Solana
Commit	6ab15b340e74735b7cd47d75a80f531152ce0a0e
Centralization	Active Ownership
Website	https://shibonkcoin.com/
Discord	https://discord.gg/shibonk
Telegram	https://t.me/SHIBONK_SOL
Twitter	https://twitter.com/SHIBONKSOL
Report Date	March 23, 2023

Verify the authenticity of this report on our website: https://solyzerx.com/projects/shibonk

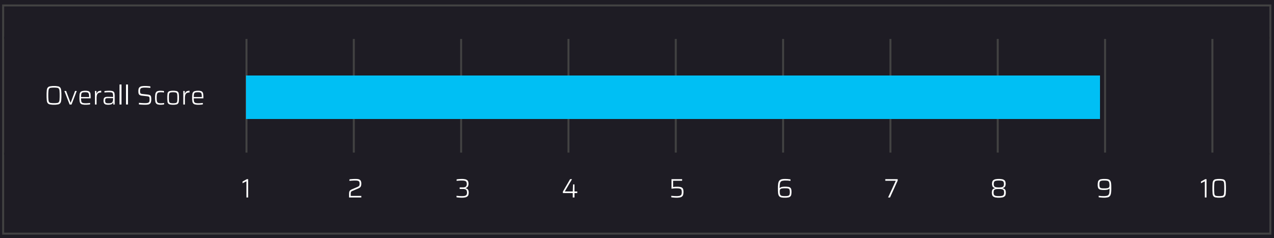
Executive Summary

SolyzerX has performed the automated and manual analysis of rust codes. Rust codes were reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Severity	High	Medium	Low	Informational	Optimization
Count	1	1	0	2	0

Category	Denial of service	Data Validation	Arithmetic	Auditing and Logging	Undefined Behavior
Count	0	1	0	1	2

SHIBONK smart contract source codes have achieved the following score: 8.9



❗ Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

❗ Please note that centralization priviledges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.

SolyzerX

Table of Contents

Scope of Work

SolyzerX volunteered to conduct a SHIBONK (SBONK) smart contract audit of their rust source codes.

The audit scope of work is strictly limited to mentioned rust file(s) only:

○ SHIBONK-SBONK

! If source codes are not deployed on the main net, they can be modified or altered before main-net deployment. Verify the contract's deployment status below:

Public Contract Link	
https://solscan.io/token/H1G6sZ1WDoMmMCFqBKAbg9gkQPCo1sKQtaJWz9dHmqZr	
Repository	https://github.com/Sperlo64/SHIBONK-SBONK.git
Type	Solana SPL token

Audit Methodology

Smart contract audits are conducted using a set standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of SolyzerX's auditing process and methodology:

Connect

- The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

Audit

- Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:
  - Soteria by Sec3
  - RUSTSEC
- Simulations are performed to identify centralized exploits causing contract and/or trade locks.
- A manual line-by-line analysis is performed to identify contract issues and centralized privileges. We may inspect below mentioned common contract vulnerabilities, and centralized exploits:

Common Vulnerabilities Exploits	• Missing signer checks • Missing ownership checks • Missing rent exemption checks • Signed invocation of unverified programs • Solana account confusions • Re-initiation with cross-instance confusion • Arithmetic overflow/underflows • Numerical precision errors • Loss of precision in calculation • Incorrect calculation

General issues in Solana and Rust programs	• Depth of Solana cross-program invocation • Reentrancy • Unsafe Rust code • Outdated dependencies • Redundant code • Do not follow security best practices

Report

○ The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.

○ The client's development team reviews the report and makes amendments to rust codes.

○ The auditing team provides the final comprehensive report with open and unresolved issues.

Publish

○ The client may use the audit report internally or disclose it publicly.

! It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of rust codes.

Risk Categories

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized here for the reader to view:

Risk Type	Definition
High	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Medium	These risks are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity.
Low	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Informational	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Undetermined	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed to mitigate the risk uncertainty.

All category breakdown which are identified in the audit report are categorized here for the reader to review:

Category Breakdown				
Denial of service	Data Validation	Arithmetic	Auditing and Logging	Unmaintained

SolyzerX

Centralized Privileges

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

- Privileged roles can be granted the power to `pause()` the contract in case of an external attack.
- Privileged roles can use functions like, `include()` , and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale and to list on an exchange.

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

The client can lower centralization-related risks by implementing below mentioned practices:

- Ensuring that the contract logic correctly implements the project specifications,
- Examining the code in detail for contract-specific low-level vulnerabilities,
- Ruling out economic attacks,
- Ruling out denial of service attacks
- Checking for instructions that allow front-running or sandwiching attacks,
- Checking for unsafe design which might lead to common vulnerabilities being introduced in the future,
- Checking for any other, as-of-yet unknown classes of vulnerabilities arising from the structure of the Solana blockchain,
- Checking for rug-pull mechanisms or hidden backdoors.

❗ Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.

Automated Analysis Check

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Description	Status
Missing Signer Checks	Case when instruction should only be available to a restricted set of entities, but the program does not verify that the call has been signed by the appropriate entity (e.g., by checking AccountInfo::is_signer).	Passed
Missing Ownership Checks	For accounts that are not supposed to be fully user-controlled, the program does not check the AccountInfo::owner field.	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.	Passed
Signed invocation of unverified programs	The program does not verify the pubkey of any program called via the invoke_signed() API.	Passed
Solana account confusions	The program fails to ensure that the account data has the type it expects to have.	Passed
Redeployment with cross-instance confusion	The program fails to ensure that the wasm code has the code it expects to have.	Passed
Arithmetic overflow/underf lows	If an arithmetic operation results in a higher or lower value, the value will wrap around with two's complement.	Passed
Numerical precision errors	Numeric calculations on floating point can cause precision errors, wich can accumulate.	Passed

Loss of precision in calculation	Numeric calculations on integer types such as division can loss precision.	Passed
Casting truncation	Potential truncation problem with a cast conversion.	Passed
Exponential complexity in calculation	Finding computational complexity in calculations.	Passed
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.	Passed
Insufficient SPL-Token account verification	Finding extra checks that should not exist with the given type of accounts.	Passed
Over/under payment of loans	A loan overpayment is when paying extra towards a loan over and above the agreed monthly repayment. A loan underpayment is when paying less towards a loan over and below the agreed monthly repayment.	Passed
Anti-pattern instruction calls	Calling some anti-pattern instructions specific to Solana blockchain.	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.	Failed
Outdated dependencies	Rust/Cargo makes it easy to manage dependencies, but the dependencies can be outdated or contain known security vulnerabilities. cargo-outdated can be used to check outdated dependencies.	Failed
Redundant code	Repeated code or dead code that can be cleaned or simplified to reduce code complexity.	Passed

Do not follow security best practices	Failing to properly use assertions, check user errors, multisig, etc.	Passed
Project specification implementation check	Ensuring that the contract logic correctly implements the project specifications.	Passed
Contract-specific low-level vulnerabilities	Examining the code in detail for contract-specific low-level vulnerabilities.	Passed
Ruling out economic attacks	Economic rules that can be exploited to steal funds.	Passed
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Front-running or sandwiching	Checking for instructions that allow front-running or sandwiching attacks.	Passed
Unsafe design vulnerabilities	Checking for unsafe design which might lead to common vulnerabilities being introduced in the future.	Passed
As-of-yet solana unknown classes of vulnerabilities	Checking for any other, as-of-yet unknown classes of vulnerabilities arising from the structure of the Solana blockchain.	Passed
Rug-pull mechanisms or hidden backdoors	Checking for rug-pull mechanisms or hidden backdoors.	Passed

Findings Summary

#	Title	Type	Severity
1	Potential segfault in the time crate	Auditing and Logging	Medium
2	reject_remote_clients Configuration corruption	Data Validation	High
3	ansi_term is Unmaintained	Unmaintained	Informational
4	net2 crate has been deprecated; use socket2 instead	Unmaintained	Informational

Detailed Findings

1. Potential segfault in the time crate	
Severity: Medium	Difficulty: High
Type: Auditing and Logging	Finding ID: 0071
Target: Time	

Description

Impact

Unix-like operating systems may segfault due to dereferencing a dangling pointer in specific circumstances. This requires an environment variable to be set in a different thread than the affected functions. This may occur without the user's knowledge, notably in a third-party library.

The affected functions from time 0.2.7 through 0.2.22 are:

- time::UtcOffset::local_offset_at
- time::UtcOffset::try_local_offset_at
- time::UtcOffset::current_local_offset
- time::UtcOffset::try_current_local_offset
- time::OffsetDateTime::now_local
- time::OffsetDateTime::try_now_local

The affected functions in time 0.1 (all versions) are:

- at
- at_utc
- now

Non-Unix targets (including Windows and wasm) are unaffected.

Patches

Pending a proper fix, the internal method that determines the local offset has been modified to always return None on the affected operating systems. This has the effect of returning an Err on the try_* methods and UTC on the non-try_* methods.

Users and library authors with time in their dependency tree should perform cargo update, which will pull in the updated, unaffected code.

Users of time 0.1 do not have a patch and should upgrade to an unaffected version: time 0.2.23 or greater or the 0.3 series.

Workarounds

A possible workaround for crates affected through the transitive dependency in chrono, is to avoid using the default oldtime feature dependency of the chrono crate by disabling its default-features and manually specifying the required features instead.

Examples:

Cargo.toml:

```
chrono = { version = "0.4", default-features = false, features = ["serde"] }
```

```
chrono = { version = "0.4.22", default-features = false, features = ["clock"] }
```

Commandline:

```
cargo add chrono --no-default-features -F clock
```

Recommendation

Upgrade to >=0.2.23

2. reject_remote_clients Configuration corruption	
Severity: High	Difficulty: Medium
Type: Data Validation	Finding ID: 0001
Target: Tokio	

Description

On Windows, configuring a named pipe server with pipe_mode will force ServerOptions::reject_remote_clients as false.

This drops any intended explicit configuration for the reject_remote_clients that may have been set as true previously.

The default setting of reject_remote_clients is normally true meaning the default is also overridden as false.

Workarounds

Ensure that pipe_mode is set first after initializing a ServerOptions. For example:

```rust
let mut opts = ServerOptions::new();
opts.pipe_mode(PipeMode::Message);
opts.reject_remote_clients(true);
```

Recommendation

Upgrade to >=1.18.4, <1.19.0 OR >=1.20.3, <1.21.0 OR >=1.23.1

3. ansi_term is Unmaintained	
Severity: Informational	Difficulty: Medium
Type: Unmaintained	Finding ID: 0139
Target: ansi_term	

Description

The maintainer has advised that this crate is deprecated and will not receive any maintenance. The crate does not seem to have much dependencies and may or may not be ok to use as-is. Last release seems to have been three years ago.

Possible Alternative(s)

The below list has not been vetted in any way and may or may not contain alternatives;

- anstyle
- console
- nu-ansi-term
- owo-colors
- stylish
- yansi

Dependency Specific Migration(s)

- structopt, clap2

4. net2 crate has been deprecated; use socket2 instead	
Severity: Informational	Difficulty: Medium
Type: Unmaintained	Finding ID: 0016
Target: net2	

Description

The net2 crate has been deprecated and users are encouraged to considered socket2 instead.

Disclaimers

SolyzerX provides the easy-to-understand audit of rust source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high level of technical risks and uncertainty. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. This audit report could include false positives, false negatives, and other unpredictable results.

Confidentiality

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without SolyzerX's prior written consent.

No Financial Advice

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Technical Disclaimer

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULT OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, SOLYZERX HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, SOLYZERX SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, SOLYZERX MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

Timeliness Of Content

The content contained in this audit report is subject to change without any prior notice. SolyzerX does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

Links To Other Websites

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than SolyzerX. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such website's and social account's owners. You agree that SolyzerX is not responsible for the content or operation of such websites and social accounts and that SolyzerX shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

SolyzerX

About SolyzerX

Founded in 2022 and headquartered in Malaysia, SolyzerX 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.

We provide solidity & rust development, testing, and auditing services. We work on major public blockchains e.g., Ethereum, Binance, Solana, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Velas, Oasis, etc.

SolyzerX is built by engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 4 core members, and 5+ casual contributors.

Website: https://solyzerx.com

Email: support@solyzerx.com

Github: https://github.com/SolyzerX

Telegram (Channel): https://t.me/SolyzerX

Telegram (Foundation): https://t.me/SolyzerXFoundation

solyzerx

solyzerx.com

t.me/solyzerx

t.me/solyzerxfoundation

SMART CONTRACT AUDITS | SOLIDITY & RUST DEVELOPMENT AND TESTING

RELENTLESSLY SECURING PUBLIC AND PRIVATE BLOCKCHAINS