GUARDIAN

Jupiter

Jup-Stablecoin

Security Assessment

November 28th, 2025

Summary

Audit Firm Guardian

Prepared By Minato Namikazi, Alex Lazar, Jakub Zmyslowski, Bounty MO

Client Firm Jupiter

Final Report Date November 28, 2025

Audit Summary

Jupiter engaged Guardian to review the security of their Jup-Stablecoin Codebase. From the 17th of November to the 24th of November, a team of 4 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the lack of critical issues detected and minimal code changes following the main review, Guardian assigns a Confidence Ranking of 5 to the protocol. Guardian advises the protocol to consider periodic review with future changes. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. **Recommendation:** Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only Low or Medium-severity issues were discovered. Design patterns are sound, and test coverage is reasonable. Small changes, such as modifying rounding logic, may introduce new vulnerabilities and should be carefully reviewed. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0 High/Critical findings. Varied Low/Medium severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1 High finding and ≥ 3 Medium. Varied Low severity findings.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities (~2/week). Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Overview

Project Summary

Project Name	Jupiter
Language	Solidity
Codebase	https://github.com/jup-ag/jup-stablecoin
Commit(s)	Main Review commit: a7fbb3be291c83124854f3f00e5c8f4c587aa1b7 Remediation Review commit: 7d4f3efa28c8a75fb1d68df579adc8298a0303c2

Audit Summary

Delivery Date	November 28, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● High	0	0	0	0	0	0
● Medium	0	0	0	0	0	0
● Low	7	0	0	1	0	6
● Info	7	0	0	1	0	6

Audit Scope & Methodology

```
/jup-stablecoin-main/programs/jup-stable/src/instructions/user.rs
[ Blank 60 ]  [ Comment 9 ]  [ Code 354 ]
/jup-stablecoin-main/programs/jup-stable/src/state/vault.rs
[ Blank 55 ]  [ Comment 1 ]  [ Code 248 ]
/jup-stablecoin-main/programs/jup-stable/src/instructions/vault.rs
[ Blank 33 ]  [ Comment 1 ]  [ Code 217 ]
/jup-stablecoin-main/programs/jup-stable/src/instructions/benefactor.rs
[ Blank 27 ]  [ Comment 2 ]  [ Code 148 ]
/jup-stablecoin-main/programs/jup-stable/src/state/benefactor.rs
[ Blank 36 ]  [ Comment 0 ]  [ Code 137 ]
/jup-stablecoin-main/programs/jup-stable/src/instructions/custodian.rs
[ Blank 29 ]  [ Comment 3 ]  [ Code 132 ]
/jup-stablecoin-main/programs/jup-stable/src/instructions/init.rs
[ Blank 10 ]  [ Comment 3 ]  [ Code 126 ]
/jup-stablecoin-main/programs/jup-stable/src/oracle.rs
[ Blank 19 ]  [ Comment 3 ]  [ Code 123 ]
/jup-stablecoin-main/programs/jup-stable/src/state/config.rs
[ Blank 25 ]  [ Comment 0 ]  [ Code 118 ]
/jup-stablecoin-main/programs/jup-stable/src/state/operator.rs
[ Blank 19 ]  [ Comment 0 ]  [ Code 95 ]
/jup-stablecoin-main/programs/jup-stable/src/instructions/admin.rs
[ Blank 12 ]  [ Comment 0 ]  [ Code 88 ]
/jup-stablecoin-main/programs/jup-stable/src/state/common.rs
[ Blank 22 ]  [ Comment 6 ]  [ Code 83 ]
/jup-stablecoin-main/programs/jup-stable/src/error.rs
[ Blank 1 ]  [ Comment 0 ]  [ Code 72 ]
/jup-stablecoin-main/programs/jup-stable/src/lib.rs
[ Blank 16 ]  [ Comment 2 ]  [ Code 71 ]
/jup-stablecoin-main/programs/jup-stable/src/instructions/operator.rs
[ Blank 12 ]  [ Comment 1 ]  [ Code 70 ]
/jup-stablecoin-main/programs/jup-stable/src/instructions/mod.rs
[ Blank 1 ]  [ Comment 0 ]  [ Code 14 ]
/jup-stablecoin-main/programs/jup-stable/src/state/mod.rs
[ Blank 0 ]  [ Comment 0 ]  [ Code 5 ]

TOTALS
[ Blank 377 ]  [ Comment 31 ]  [ Code 2101 ]
```

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High	Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.
Medium	A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.
Low	Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High	The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.
Medium	An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.
Low	Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
L-01	Slot-timestamp Mismatch Breaks Price Fetching	DoS	● Low	Resolved
L-02	SetStatus(Disabled) Requires A Valid Oracle	Unexpected Behavior	● Low	Resolved
L-03	Last Admin Removal Allow Permanent Protocol Halt	Validation	● Low	Resolved
L-04	Lack Of PegManager Access Check	Access Control	● Low	Resolved
L-05	Restricted Custodian Deposits/Withdrawals	Logical Error	● Low	Resolved
L-06	Majority Of Oracles Instead Of All	Oracle	● Low	Acknowledged
L-07	Switchboard Max_staleness Uses Seconds Not Slots	Oracle	● Low	Resolved
I-01	Needlessly Public OraclePrice Functions	Best Practices	● Info	Resolved
I-02	Enabled Vault Can Be Left With Zero Oracles	Unexpected Behavior	● Info	Resolved
I-03	From_pyth_v2() Can DOS Minting And Redeeming	DoS	● Info	Resolved
I-04	Missing Decimal Check At Pool Creation	Validation	● Info	Resolved
I-05	Admins Can't Clear Roles Once Set	Informational	● Info	Resolved
I-06	Vault.bump Field Is Never Initialized	Best Practices	● Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-07	Unsafe Downcasting	Math	● Info	Acknowledged

L-01 | Slot-timestamp Mismatch Breaks Price Fetching

Category	Severity	Location	Status
DoS	● Low	oracle.rs: 52	Resolved

Description

There is an integration bug in how PullFeedAccountData get_value is used, the code calls get_value with clock.unix_timestamp.

```
let price = price_feed.get_value(clock.unix_timestamp as u64, stalesness_threshold, 1, true)
```

But the external library expect that argument to be clock_slot not a Unix timestamp. s.slot values are in slot units around ~381M on Solana mainnet, while clock.unix_timestamp is seconds since Unix epoch ~1.7B.

It filters submissions via s.slot = clock_slot - max_staleness. Due the bug, the filter check becomes

```
s.slot >= (unix_timestamp_seconds) - (max_staleness)
```

The code send max_staleness = 300 as default , so it will equals ~ 381_194_828 > ~ 1_763_589_000 - 300
That will always be false, no submission ever passes the filter, submissions.len() = 0.

So the later submissions.len() < min_samples will always revert with "NotEnoughSamples". On every user mint / redeem call, parse_oracles is being called

Which collect() all the prices from Pyth, Doves and SwitchboardOnDemand oracles, save them into Result<Vec<OraclePrice>>. If every oracle returns Ok(price), the code take minimum oracle price of that vector and proceed to the min() with the 1:1 path in mint/redeem.

But if any oracle returned Err() for any reason the whole mint/redeem tx reverts. When we collect() an iterator of Result<T, E>, we get Result<Vec<T>, E>,. This works as Build a vector of all the Ok values, but if you see a single Err, stop and return that Err instead.

All users mint / redeem operations on vaults which have a SwitchboardOnDemand oracle will revert.
After DoS, a vault manager can set the Switchboard oracle to None via update_oracle.

But this will still cause blocking the usage of Switchboard until the code be mitigated, since the current integration is broken.

Recommendation

Use clock.slot instead of clock.unix_timestamp in get_value.

Resolution

Jupiter Team: Resolved.

L-02 | SetStatus(Disabled) Requires A Valid Oracle

Category	Severity	Location	Status
Unexpected Behavior	● Low	vault.rs: 174	Resolved

Description

In manage_vault handler in the SetStatus action, the code always checks whether the vault has at least one oracle configured (at least one oracle entry is non-empty) and reverts with a NoValidOracle error if all oracle slots are empty.

This check runs regardless of the target status, so it also blocks SetStatus(Disabled) on a vault with no valid oracles. By contrast, the dedicated Disable action (requiring the VaultDisabler role) does not perform this oracle check and can successfully disable such a vault.

This creates an inconsistency in how disabling works - a VaultManager using SetStatus(Disabled) may be unable to disable a misconfigured vault that a VaultDisabler can disable.

As a result, an operator with only this VaultManager role cannot call SetStatus(Disabled) on a vault that currently has no valid oracles, even though disabling such a vault might be a reasonable emergency action, leading to confusing expectations about which role is responsible for safely disabling vaults.

Recommendation

Only require a valid oracle when enabling the vault. In manage_vault instruction allow SetStatus(Disabled) action to proceed regardless of oracle state.

Resolution

Jupiter Team: Resolved.

L-03 | Last Admin Removal Allow Permanent Protocol Halt

Category	Severity	Location	Status
Validation	● Low	config.rs: 63-69	Resolved

Description

In `Config::remove_admin`

There is no check that at least one admin remains, or prevents an admin from removing themselves.

`psm::manage_config` lets an admin pause the protocol and then remove all admins including themselves, leaving `is_paused == true` and zero admins.

Since `manage_config` requires an existing admin to run, there is no on chain way to unpause after this, and the protocol (including all redemptions) is permanently blocked

That's an info issue, but add a check as precaution

Recommendation

Disallow removing the last admin

```
require!(self.num_admins() > 1, PSmError::NotAllowed);
```

Resolution

Jupiter Team: Resolved.

L-04 | Lack Of PegManager Access Check

Category	Severity	Location	Status
Access Control	● Low	admin.rs: 86	Resolved

Description

The protocol defines a dedicated `PegManager` role, intended to be used in `ConfigManagementAction::SetPegPriceUSD`.

There is a missed `operator.is(OperatorRole::PegManager)?;` in the code. `SetPegPriceUSD` is guarded by Admin, and the intention is that it has its dedicated role, as confirmed in the meeting call.

Currently, the protocol cannot delegate peg operations to a specialized multisig without giving it full admin privileges

Recommendation

Implement the `PegManager` access check

Resolution

Jupiter Team: Resolved.

L-05 | Restricted Custodian Deposits/Withdrawals

Category	Severity	Location	Status
Logical Error	● Low	custodian.rs: 33	Resolved

Description

The program’s data model separates the token program for:

• the LP mint (stored in `Config.token_program`),

• the vault/collateral mint (stored in `Vault.token_program`).

Both Mint and Redeem instructions respect this split, check if `config.token_program == lp_token_program.key()` and `vault.token_program == vault_token_program.key()` and uses `lp_token_program` for LP and `vault_token_program` for collateral operations.

However, the Deposit and Withdraw instructions enforce that both the LP token program and the vault token program are the same as the single `token_program` account passed in.

These instructions only move collateral and never touch the LP mint, so the `config.token_program` equality is irrelevant and conflicts with the design that allows LP and collateral to live on different token program IDs (e.g. LP on Token-2022, collateral on SPL).

Consequently, if LP and collateral use different token program IDs (which Mint and Redeem allow), Deposit and Withdraw will revert with `InvalidTokenProgram` error.

Recommendation

Remove the `config.token_program == token_program.key()` constraint from both Deposit and Withdraw so they only validate and use `vault.token_program`, allowing LP and collateral to use different token program IDs. Optionally, if the LP and collateral are always intended to share the same token program, enforce that consistently.

Resolution

Jupiter Team: Resolved.

L-06 | Majority Of Oracles Instead Of All

Category	Severity	Location	Status
Oracle	● Low	parse_oracles()	Acknowledged

Description

As described in the SwitchboardOnDemand issue, the code collect() all the prices from Pyth, Doves, and SwitchboardOnDemand oracles, and saves them into Result<Vec<OraclePrice>>.

If every oracle returns Ok(price), the code takes the minimum oracle price of that vector and proceeds. But if any oracle returns Err() for any reason, the whole mint/redeem tx reverts.

Despite that the 3 oracles will be working correctly, to avoid temporary DoS in some cases, we could check if at least 2 succeed.

Recommendation

For example ethena does

```
if (validOracleCount < minNumberOfOracles) {
// revert
}
```

After filtering, they count validOracleCount. If validOracleCount < minNumberOfOracles, they revert.

Important: If you implemented this change make sure to not mask the other critical errors, the code enforce that all configured oracle accounts are present and match the vault config (owner + pubkey) only treat genuine soft failures (stale price, internal oracle error) as skippable, and only if we have reached the minNumberOfOracles.

Resolution

Jupiter Team: Acknowledged.

L-07 | Switchboard Max_staleness Uses Seconds Not Slots

Category	Severity	Location	Status
Oracle	● Low	oracle.rs: 52	Resolved

Description

Other than clock.unix_timestamp, the code also passes the threshold differently.

The vault stalesness_threshold is defined and used as a time in seconds for Pyth and Doves, but it is passed directly as max_staleness into PullFeedAccountData::get_value for Switchboard:

get_value is implemented to treat max_staleness as a slot count, not seconds.

This becomes ~300 seconds for Pyth/Doves but ~300 slots (~2–3 minutes) for Switchboard, leading to inconsistent freshness behavior across oracles.

This can reject Switchboard prices more aggressively than configured, increasing the chance of NotEnoughSamples for Switchboard while Pyth/Doves still pass

Recommendation

keep stalesness_threshold as a seconds parameter for Pyth/Doves, but convert it to slots before calling Switchboard

Resolution

Jupiter Team: Resolved.

I-01 | Needlessly Public OraclePrice Functions

Category	Severity	Location	Status
Best Practices	● Info	oracle.rs: 16	Resolved

Description

I'd say `from_pyth_v2()` and its counterparts in `oracle.rs` should be private instead of public because they have minimal internal validation (looking at the use of `AccountsInfo` with no manual validation in particular) and are meant to only be used by `parse_oracles()` (which does indeed validate account owners, etc).

This impacts nothing right now because they aren't used anywhere else but in `parse_oracles()` but changing this would prevent somebody mistakenly using them directly in the future.

Recommendation

Make `from_pyth_v2()` and its counterparts private instead of public.

Resolution

Jupiter Team: Resolved.

I-02 | Enabled Vault Can Be Left With Zero Oracles

Category	Severity	Location	Status
Unexpected Behavior	● Info	vault.rs: 178	Resolved

Description

The protocol correctly blocks enabling a vault that has no valid oracle (for the SetStatus(Enabled) action it requires at least one non-empty oracle). However, once a vault is Enabled, an operator can call UpdateOracle and set each oracle slot to None (EmptyOracle) without any guard.

This allows the vault to remain Enabled while having zero configured oracles. In that state, user instructions (mint/redeem) will fail at oracle validation, resulting in an enabled vault that unexpectedly cannot process user flows, confusing integrations and operators.

Recommendation

Consider rejecting updates that would remove the last oracle while vault is enabled or set the vault's status to Disabled during that case.

Resolution

Jupiter Team: Resolved.

I-03 | From_pyth_v2() Can DOS Minting And Redeeming

Category	Severity	Location	Status
DoS	● Info	oracle.rs: 17-40	Resolved

Description

Look at the (price_u64 - price.conf).into(), snippet of the from_pyth_v2() function. Pyth only guarantees conf to be "positive when present" but not smaller than the price.

Albeit highly unlikely, it is possible that stablecoins become distressed (TerraLuna, USDC/DAI depeg after SVG). Therefore a scenario where conf is bigger or equal to price is also unlikely but possible.

In the case where this underflows, because you've set profile.release to have overflow-checks = true, this will panic. It's not caught within parse_oracles() and parse_oracles() is always called with the ? operator. This means the panic will propagate and revert the transaction even if all of the other oracles returned valid prices, i.e.: a DoS in mint() and redeem().

What's more, this line can also return 0 if price == conf. Returning 0 is unexpected and something that is clearly not wanted (since we've checked that the price is not 0 or negative in this function and also for Switchboard we use only_positive=true).

There is a layer of defense for this situation: the vault min-max limits, but there's no guarantee that they will be sensible (even though the defaults are sensible at $0.5-$1, these can be eventually set to anything).

Now, parse_oracles() returns the smallest given price, so it will return the 0 if it's provided by from_pyth_v2() even if there are other valid prices from the other oracles.

And if the vault min-max limits don't catch it, this would trickle down to compute_mint_amount and make it return 0 as mint_amount which would make require!(mint_amount > 0, JupStableError::ZeroAmount); throw. On the redeeming side, an oracle price of 0 would trickle down to cause a "division by 0" panic in calculate_redeem_amount.

Recommendation

It would be better to check something like price.checked_sub(conf) < 0 rather then the price.price < 0 check we have and the price_u64 - price.conf we do. checked_sub returns none if there's an overflow instead of panicking.

Resolution

Jupiter Team: Resolved.

I-04 | Missing Decimal Check At Pool Creation

Category	Severity	Location	Status
Validation	● Info	pool.rs: 71	Resolved

Description

Because of `require!(diff < 19, PSmError::MathOverflow);`

`normalize_amount` always assumes that

`|decimals - target_decimals| < 19`

But that is never enforced anywhere when the pool is created. The program allows a pool to be created between any two mints, including ones whose decimals differ by 20+

And when a user later calls `redeem`, `normalize_amount` will see `diff > 19` and revert with `PSmError::MathOverflow` before any transfers happen.

So a pool can be configured and look valid on-chain, admins can `supply` and `withdraw`, but no user will ever be able to redeem if the decimals gap is > 19.

Recommendation

Enforce the same require check at pool creation for consistency

Resolution

Jupiter Team: Resolved.

I-05 | Admins Can't Clear Roles Once Set

Category	Severity	Location	Status
Informational	● Info	operator.rs: 78	Resolved

Description

The implementation for Operator (https://github.com/GuardianOrg/jup-stablecoin-team1-1762878643100/blob/a7fbb3be291c83124854f3f00e5c8f4c587aa1b7/programs/jup-stable/src/state/operator.rs#L75) contains both set_role and clear_role.

The set_role function is use in manage_operator (https://github.com/GuardianOrg/jup-stablecoin-team1-1762878643100/blob/a7fbb3be291c83124854f3f00e5c8f4c587aa1b7/programs/jup-stable/src/instructions/operator.rs#L78), but clear_role is used nowhere (not within manage_operator nor another function.

This means the admin cannot clear a role once given.

P.S.: If a role owner becomes malevolent you can still disable them from being an operator, so this is not the worst bug in the world.

Recommendation

Implement clear_role() as part of manage_role() similar to how it was done for set_role() or potentially add another program callable function similar to delete_benefactor()

Resolution

Jupiter Team: Resolved.

I-06 | Vault.bump Field Is Never Initialized

Category	Severity	Location	Status
Best Practices	● Info	vault.rs: 96	Resolved

Description

The Vault account includes a `bump` field, but `create_vault` does not set it and it remains at its default value 0.

That makes the field effectively unused and can cause maintenance issues if later code assumes it contains the correct bump for vault PDA derivations.

Recommendation

Either initialize `vault.bump` in `create_vault` using `ctx.bumps.vault` or remove the field entirely.

Resolution

Jupiter Team: Resolved.

I-07 | Unsafe Downcasting

Category	Severity	Location	Status
Math	● Info	state/benefactor.rs	Acknowledged

Description

The following functions perform downcasting from u128 to u64 using the as keyword, which omits arithmetic overflow checks:

```
pub fn calculate_mint_fee(&self, amount: u64) -> u64 {
(amount as u128 * self.mint_fee_rate as u128 / 10000) as u64 + 1
}
pub fn calculate_redeem_fee(&self, amount: u64) -> u64 {
(amount as u128 * self.redeem_fee_rate as u128 / 10000) as u64 + 1
}
```

Recommendation

It is recommend to use try_into() instead of the as keyword.

Resolution

Jupiter Team: Acknowledged.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. 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. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits