GA GUARDIAN

USDT0

Multihop

Security Assessment

February 14th, 2025

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand

Client Firm USDT0

Final Report Date February 14, 2025

Audit Summary

USDT0 engaged Guardian to review the security of their USDT0 multihop contract. From the 6th of February to the 10th of February, a team of 2 auditors reviewed the source code in scope. All findings have been recorded in the following report.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: N/A

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: N/A

Table of Contents

Project Overview

Project Summary

Project Name	USDT0
Language	Solidity
Codebase	https://github.com/Everdawn-Labs/usdt0-oft-contracts/pull/47
Commit(s)	fa631552d85c9974fdc7b4c5873ec92a77966194

Audit Summary

Delivery Date	February 14, 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	1	0	0	0	0	1
Low	8	0	0	6	0	2
Info	16	0	0	8	1	7

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
M-01	Funds Trapped On Ethereum	DoS	Medium	Resolved
L-01	retrySend Is Unusable For Users With Multiple Failures	Unexpected Behavior	Low	Acknowledged
L-02	Excess Ether Cannot Be Easily Retrieved	Unexpected Behavior	Low	Resolved
L-03	Force Retry With Insufficient Message Value	Censoring	Low	Acknowledged
L-04	Force Retry With Insufficient Gas	Censoring	Low	Acknowledged
L-05	Blacklisted Addresses May Retrieve Funds	Unexpected Behavior	Low	Acknowledged
L-06	Reentrancy Checks Abused For Censoring	Censoring	Low	Acknowledged
L-07	Unexpected Zero Amount Oft Sends	Validation	Low	Acknowledged
L-08	Insufficient Reserved Gas	Warning	Low	Resolved
I-01	Unused Import	Imports	Info	Resolved
I-02	Missing Event	Events	Info	Resolved
I-03	Unused Events	Superfluous Code	Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-04	Lacking Zero Address Checks	Validation	Info	Resolved
I-05	Outdated Documentation	Documentation	Info	Partially Resolved
I-06	Zero Transfer Attempted	Superfluous Code	Info	Acknowledged
I-07	Arbitrary Oft Address	Best Practices	Info	Acknowledged
I-08	Malformed Composed Messages Trap USDT	Trapped Funds	Info	Acknowledged
I-09	Arbitrary Oapp Deployed Through Factory	Warning	Info	Partially Resolved
I-10	SendParam Can Differ From Initial lzCompose	Warning	Info	Acknowledged
I-11	Stargate Only Allows Same-Asset Transfers	Documentation	Info	Acknowledged
I-12	Refunds Are Not Supplied	Documentation	Info	Acknowledged
I-13	Arguments Can Be Calldata	Optimization	Info	Acknowledged
I-14	Lacking setReservedGas Validation	Validation	Info	Acknowledged
I-15	retrySend DoS	DoS	Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-16	Smart Contract Claimers Incompatibility	Compatibility	Info	Resolved

M-01 | Funds Trapped On Ethereum

Category	Severity	Location	Status
DoS	Medium	MultiHopCompserV1.sol	Resolved

Description

The `MultiHopComposerV1` contract uses an interface which returns a boolean value for the `approve` function, however USDT on Ethereum does not return a boolean from the `approve` function.

This results in a revert upon approval which will cause the compose message to be un-executable, thereby trapping the USDT in the `MultiHopComposerV1` contract.

Recommendation

For `MultiHopComposerV1` deployment on Ethereum mainnet use an `IERC20` interface which does not include a boolean return value for the `approve` function.

Resolution

USDT0 Team: Resolved.

L-01 | retrySend Is Unusable For Users With Multiple Failures

Category	Severity	Location	Status
Unexpected Behavior	Low	MultiHopComposerV1.sol: 126	Acknowledged

Description

The `retrySend` function forces the user to send their entire `amountOwed` balances in a single `oft.send` call.

However the user may have had multiple failed hops, in which case the `amountNative` would be too large for a single `oft.send` call and the `amountToken` amount could be larger than the total amount the user wishes to send to any single receiving address.

For example:
• Bob initiates a composed hop A which is to be sent to Alice on Chain 1
• Bob initiates a composed hop B which is to be sent to Carol on Chain 2
• Both hop A and B fail and are stored within the `amountOwed` mapping
• It is impossible for Bob to retry both of these hops individually since they are grouped into the same `amountOwed` entry

Recommendation

Consider allowing users to specify what amount of both tokens they would like to use from their `amountOwed` mapping entry for the `retrySend` call.

Resolution

USDT0 Team: Acknowledged.

L-02 | Excess Ether Cannot Be Easily Retrieved

Category	Severity	Location	Status
Unexpected Behavior	Low	MultiHopComposerV1.sol: 103	Resolved

Description

In the lzCompose function the refund address in the oft.send call is assigned as the MultiHopComposerV1 contract. As a result excess msg.value sent to the lzCompose function remains in the MultiHopComposerV1 contract.

There is a function for the owner to reclaim native assets from the MultiHopComposerV1 contract, however this function only allows sweeping the entire native balance which includes refunds for failed OFT transfers which should be claimable by those users.

As a result it is unwieldy for the owner to claim this excess Ether.

Recommendation

Consider sending the excess value to a dedicated refund receiver address, or implement logic to track the totality of pending user native refunds and implement a function which allows the owner to withdraw only the excess native tokens which were refunded from the oft.send call.

Resolution

USDT0 Team: Resolved.

L-03 | Force Retry With Insufficient Message Value

Category	Severity	Location	Status
Censoring	Low	MultiHopComposerV1.sol: 77	Acknowledged

Description

The lzCompose function in the LayerZero EndpointV2 contract can be called by anyone and simply validates that the message contents are the same as those which were sent by the OApp.

Therefore any arbitrary address can force a retry by executing the compose message with a value less than the MessagingFee specifies, causing a NotEnoughNative revert and _handleError to be triggered.

A malicious actor can observe that a composed message to the MultiHopComposerV1 contract has been posted, or force it to be posted themselves by executing the parent lzReceive message, and then use their own malicious lzCompose invocation to force the Oft.send call to fail.

Recommendation

Ensure the msg.value matches the fee.

Resolution

USDT0 Team: Acknowledged.

L-04 | Force Retry With Insufficient Gas

Category	Severity	Location	Status
Censoring	Low	MultiHopComposerV1.sol: 77	Acknowledged

Description

The lzCompose function in the LayerZero EndpointV2 contract can be called by anyone and simply validates that the message contents are the same as those which were sent by the OApp.

Therefore any arbitrary address may invoke the lzCompose function for a MultiHopComposerV1 compose call with an amount of gas such that the oft.send function reverts with out of gas, but the rest of the execution completes due to the reservedGas.

A malicious actor can observe that a composed message to the MultiHopComposerV1 contract has been posted, or force it to be posted themselves by executing the parent lzReceive message, and then use their own malicious lzCompose invocation to force the Oft.send call to fail.

Recommendation

Consider adding validation to the lzCompose function that requires that a sufficient amount of gas has been provided by the caller to successfully execute the oft.send call.

Resolution

USDT0 Team: Acknowledged.

L-05 | Blacklisted Addresses May Retrieve Funds

Category	Severity	Location	Status
Unexpected Behavior	Low	MultiHopComposerV1.sol	Acknowledged

Description

Through the lzCompose function USDT tokens may be credited to an arbitrary evmRefundAddress with funds in the amountOwed. The evmRefundAddress may be blacklisted at the time of the lzCompose execution or may be blacklisted in the future.

In either case, the blacklisted address is able to reclaim their funds through either the retrieveFunds or retrySend functions.

Recommendation

Consider adding validation to require that the user is not blacklisted in the retrieveFunds and retrySend functions.

Resolution

USDT0 Team: Acknowledged.

L-06 | Reentrancy Checks Abused For Censoring

Category	Severity	Location	Status
Censoring	Low	Global	Acknowledged

Description

The `send` function in the `StarGatePool` contract invokes the `sendToken` function which uses the `nonReentrantAndNotPaused` modifier.

If any functions in the `StarGatePool` contract with the `nonReentrantAndNotPaused` modifier have been entered into then any subsequent calls to `send` will revert due to the reentrancy check.

This behavior can be leveraged to censor composed hops by causing them to fail on the `Oft.send` function.

A malicious actor can observe that a composed message to the `MultiHopComposerV1` contract has been posted, or force it to be posted themselves by executing the parent `lzReceive` message, and then use their own malicious `lzCompose` invocation to force the `Oft.send` call to fail.

The malicious actor can first invoke their own dummy `StarGatePool.send` invocation to trigger the reentrancy guard, and then inside of that function call, during the native refund callback to their address.

The malicious actor can invoke the `lzCompose` function on the `EndpointV2` contract to trigger the `MultiHopComposerV1.lzCompose` function which will then make a call to the `StarGatePool.send` function that fails due to a reentrant call.

The user's send is then censored and they are forced to retry their send with the `retrySend` function.

Recommendation

If the target `oft` address is the `StarGatePool`, consider checking if the public `status` value is `ENTERED`, and if so reverting the `lzCompose` transaction rather than censoring the transaction for the user. Otherwise the `lzCompose` function can validate that the executor is a trusted executor.

Resolution

USDT0 Team: Acknowledged.

L-07 | Unexpected Zero Amount Oft Sends

Category	Severity	Location	Status
Validation	Low	MultiHopComposerV1.sol: 126	Acknowledged

Description

The `retrySend` function does not validate that the sender has a nonzero token balance in the `amountOwed` mapping before triggering the OFT send. As a result any user may call the `retrySend` function and trigger an OFT send from the `MultiHopComposerV1` contract with a zero token amount.

This may be unexpected, particularly since the user can send composed messages from the `MultiHopComposerV1` along with this zero token amount.

Recommendation

Consider validating that the user has a nonzero `amountToken` in the `retrySend` function.

Resolution

USDT0 Team: Acknowledged.

L-08 | Insufficient Reserved Gas

Category	Severity	Location	Status
Warning	Low	MultiHopComposerV1.sol	Resolved

Description

With additional logic inside of the `forceApprove` function and the added gas for an external token approval call, the `reservedGas` is not sufficient for the `_handleError` function and forced approval to occur in the event that the OFT send runs out of gas.

Recommendation

Consider increasing the `reservedGas` by a minor amount to 42,000.

Resolution

USDT0 Team: Resolved.

I-01 | Unused Import

Category	Severity	Location	Status
Imports	Info	MultiHopComposerV1.sol	Resolved

Description

In the `MultiHopComposerV1` the `IOAppCore` interface is imported but not used in the contract code.

Recommendation

Remove the extraneous `IOAppCore` interface import.

Resolution

USDT0 Team: Resolved.

I-02 | Missing Event

Category	Severity	Location	Status
Events	Info	MultiHopComposerV1.sol: 121	Resolved

Description

In the retrieveFunds function there is an event to indicate the retrieval of the underlying token with the LogRetrieveFunds event, but no event nor data entry in the LogRetrieveFunds event to indicate that native value was retrieved from the contract.

Similarly there is no indication of native value retrieved in the retrySend function.

Recommendation

Consider either adding a native value field to the LogRetrieveFunds event or introducing an event to indicate the retrieval of native funds in the retrieveFunds and retrySend function.

Resolution

USDT0 Team: Resolved.

I-03 | Unused Events

Category	Severity	Location	Status
Superfluous Code	Info	MultiHopComposerV1.sol	Resolved

Description

The `MultiHopComposerV1` contract contains the `LogTooHighSendAmount` and `Swapped` events which are declared but never used in the contract.

Recommendation

Implement the use case for the `LogTooHighSendAmount` or `Swapped` events or remove them from the contract.

Resolution

USDT0 Team: Resolved.

I-04 | Lacking Zero Address Checks

Category	Severity	Location	Status
Validation	Info	MultiHopFactoryV1.sol: 17	Resolved

Description

The constructor for the `MultiHopFactoryV1` contract performs no zero address validation on the `_endPoint` address.

Recommendation

Consider performing validation on the `_endPoint` parameter to ensure it is not the zero address.

Resolution

USDT0 Team: Resolved.

I-05 | Outdated Documentation

Category	Severity	Location	Status
Documentation	Info	MultiHopComposerV1.sol	Partially Resolved

Description

The documentation for the `constructor` of the `MultiHopComposerV1` contract references the `StableComposer` contract as the contract which it constructs. However the `MultiHopComposerV1` is the contract which is constructed.

Additionally, the documentation for the contract indicates that `The contract is intended to unwrap USDT0 into native gas tokens`. However this is not the current functionality of the contract.

Recommendation

Update the comment for the `constructor` to reflect that it is the `MultiHopComposerV1` contract which is being constructed. And update the documentation for the contract to reflect its current intended behavior.

Resolution

USDT0 Team: Partially Resolved.

I-06 | Zero Transfer Attempted

Category	Severity	Location	Status
Superfluous Code	Info	MultiHopComposerV1.sol: 117	Acknowledged

Description

Within the `retrieveFunds` function, if `retrieveNative` is set to true but the `amountToken` has already been retrieved, the function will still attempt to transfer `amountToken` even if it is a zero amount.

For USDT and USDT0 implementations this is not a large concern as they do not revert on zero amount transfers, however if the `MultiHopComposerV1` contract were to be used with a token that reverts on zero transfers this would prevent claiming of native funds.

Recommendation

Consider only executing the token transfer if the `amountToken` is larger than zero as an optimization.

Resolution

USDT0 Team: Acknowledged.

I-07 | Arbitrary Oft Address

Category	Severity	Location	Status
Best Practices	Info	MultiHopComposerV1.sol: 126	Acknowledged

Description

The `MultiHopComposerV1` contract accepts an arbitrary `oft` address as a part of the composed `_message` in the `lzCompose` and `retrySend` functions.

While no exploit has been identified with the call to an arbitrary `oft` address, out of an abundance of caution it may be best to limit the attack surface by requiring that the provided `oft` address is explicitly whitelisted.

It may be noteworthy that an untrusted oft contract can:

- Remove the approved amount of USDT from the `MultiHopComposerV1` contract
- Revert on purpose, causing a `_handleError` invocation
- Consume an unexpected amount of gas
- Return malformed `returndata`, causing a revert of the `lzCompose` function
- Re-enter into `MultiHopComposerV1` functions as well as other related OFT/LayerZero systems.

Recommendation

Consider introducing a `whitelistedOfts` mapping and validating the `oft` addresses used against this.

Resolution

USDT0 Team: Acknowledged.

I-08 | Malformed Composed Messages Trap USDT

Category	Severity	Location	Status
Trapped Funds	Info	MultiHopComposerV1.sol	Acknowledged

Description

In the lzCompose function in the event that the Oft.send function reverts a refund is stored for users with the _handleError function. However in the event that a revert occurs outside of the Oft.send function the user will not be able to retrieve their funds from the MultiHopComposerV1 contract.

Specifically, if the composed _message is malformed and does not contain the expected evmRefundAddress, oft, sendParam, and messageingFee types with no dirty upper bits then the lzCompose will chronically revert and the composed action can never be executed.

Recommendation

Be aware of this risk and warn users and integrators to verify the correctness of their composed message structure.

Resolution

USDT0 Team: Acknowledged.

I-09 | Arbitrary Oapp Deployed Through Factory

Category	Severity	Location	Status
Warning	Info	MultiHopComposerFactoryV1.sol	Partially Resolved

Description

Function `MultiHopComposerFactoryV1.createMultiHopComposer` takes an arbitrary oApp address. Therefore, users can populate the `composers` mapping with a composer that uses a malicious `Oapp` and lead to user interaction with an undesired contract and token within the `MultiHopComposer`.

There is also risk with a block reorg that a user may believe they are interacting with a composer with a safe `Oapp`, but within the context of the reorg a composer with a malicious `OApp` is deployed to that address first leading to users interacting with a malicious contract unexpectedly.

This is because the CREATE opcode relies solely on the sender address and nonce. Consider the following example:

(1) Alice deploys a composer with a safe Oapp A to address A.
(2) Bob deploys a composer with a malicious Oapp B to address B.
(3) A reorg occurs, such that Bob's tx is included first and the malicious composer is now deployed to address A, since the sender address and nonce are the same as when Alice created a composer initially.
(4) Users who thought they were interacting with Alice's safe composer at address A now interact with Bob's malicious composer.

Recommendation

Consider enforcing a whitelist for the `oApp` or document this risk.

Resolution

USDT0 Team: Partially resolved by using CREATE2 to mitigate re-org risk.

Guardian Team: Arbitrary addresses are still allowed to create arbitrary multihop contracts with arbitrary OApps, this is acknowledged by the team.

I-10 | SendParam Can Differ From Initial lzCompose

Category	Severity	Location	Status
Warning	Info	MultiHopComposerV1.sol	Acknowledged

Description

The SendParam initially used for the `lzCompose` can differ from the one passed by users during `retrySend`.

Recommendation

Be aware of this behavior.

Resolution

USDT0 Team: Acknowledged.

I-11 | Stargate Only Allows Same-Asset Transfers

Category	Severity	Location	Status
Documentation	Info	Global	Acknowledged

Description

Stargate currently does not support Bera/INK USDT0 to be transferred to USDT that is on other chains such as Polygon. Therefore, users should be aware that for most USDT0 transfers Arbitrum will be the intermediate step.

Recommendation

Make users aware that these pathways must be implemented on StarGate and cannot immediately be supported.

Resolution

USDT0 Team: Acknowledged.

I-12 | Refunds Are Not Supplied

Category	Severity	Location	Status
Documentation	Info	MultiHopComposerV1.sol	Acknowledged

Description

When invoking the `oft.send` function through `lzCompose` the refund address is the `MultiHopComposer`. In contrast, when sending through `retrySend` the refund address is the `msg.sender`.

This asymmetry makes it preferable for a user to attempt a retry to ensure unused native fee is returned to them rather than donated to the `MultiHopComposer` to be swept by the owner at a later time.

Recommendation

It may be unpreferred to allow the user to specify a refund address for the `oft.send` invocation given the address can make arbitrary actions, expend an unexpected amount of gas, or simply revert.

Instead, the USDT0 team may consider implementing `amountOwed` logic in the receive function to credit the `from` address of the current compose message, but only when an `lzCompose` invocation is in progress.

Resolution

USDT0 Team: Acknowledged.

I-13 | Arguments Can Be Calldata

Category	Severity	Location	Status
Optimization	Info	MultiHopComposerV1.sol: 126	Acknowledged

Description

The `messagingFee` function of the `retrySend` function is not modified and can therefore be declared as `calldata`.

Recommendation

Consider declaring the `messagingFee` parameter as `calldata` instead of `memory`.

Resolution

USDT0 Team: Acknowledged.

I-14 | Lacking setReservedGas Validation

Category	Severity	Location	Status
Validation	Info	MultiHopComposerV1.sol: 64	Acknowledged

Description

The `setReservedGas` function does not implement any validation which prevents the owner from assigning the `reservedGas` value to an errantly high amount.

If the `reservedGas` is assigned too high it will cause underflow reverts and can potentially trap USDT funds in the `MultiHopComposerV1` contract.

Recommendation

Consider adding maximum configurable value validation for the `setReservedGas` function.

Resolution

USDT0 Team: Acknowledged.

I-15 | retrySend DoS

Category	Severity	Location	Status
DoS	Info	MultiHopComposerV1.sol: 126	Resolved

Description

The `oft.send` function requires that the provided `msg.value` is exactly the same as the `nativeFee` specified by the `MessagingFee` parameter.

Since the value sent to the `oft.send` function is based upon `amountNative` stored in the `amountOwed` mapping, it is possible for a malicious actor to frontrun an attempt to `retrySend` and increase the `amountNative` entry to DoS the `retrySend` invocation.

The malicious actor can achieve this with a composed message waiting in the `EndpointV2` which uses the target victim address as the `evmRefundAddress` and sends to an invalid eid to force a send revert.

The malicious actor can then frontrun the `retrySend` invocation to execute their own compose message and increment the `amountNative` by even just 1 wei.

Recommendation

Be aware of this frontrunning DoS vector and document it for users and integrators of the `MultiHopComposerV1`.

Resolution

USDT0 Team: Resolved.

I-16 | Smart Contract Claimers Incompatibility

Category	Severity	Location	Status
Compatibility	Info	MultiHopComposerV1.sol: 139	Resolved

Description

In the retrySend function the refundReceiver is hardcoded as the msg.sender. As a result smart contract addresses which do not implement a receive function cannot accept the refund and therefore cause a revert when additional native fees are provided.

Users may provide an evmRefundAddress which can accept native funds to avoid this, however if the provided evmRefundAddress is not compatible then the user will be forced to redeem their funds with the retrieveFunds function to a recipient address.

Recommendation

Consider allowing the user to specify a refundRecipient address in the retrySend function.

Resolution

USDT0 Team: Resolved.

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 Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits 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