

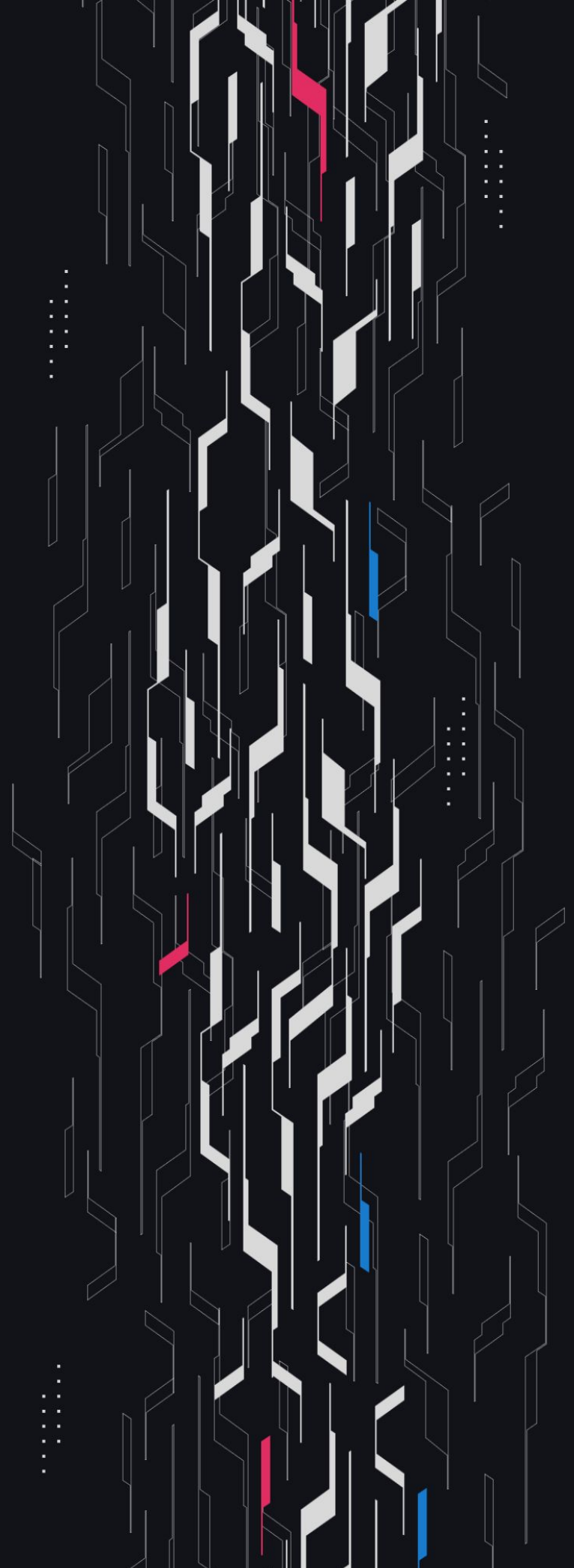
GA GUARDIAN

USDT0

OneSig EVM

Security Assessment

March 21st, 2025



Summary

Audit Firm Guardian

Prepared By Owen Thurm, Roberto Reigada, Daniel Gelfand

Client Firm USDT0

Final Report Date March 21, 2025

Audit Summary

USDT0 engaged Guardian to review the security of their OneSig multi-chain multisig EVM implementation. From the 10th of March to the 12th of March, a team of 3 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.



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

Table of Contents

Project Information

Project Overview 4

Audit Scope & Methodology 5

Smart Contract Risk Assessment

Findings & Resolutions 7

Addendum

Disclaimer 37

About Guardian Audits 38

Project Overview

Project Summary

Project Name	USDT0
Language	Solidity
Codebase	https://github.com/Everdawn-Labs/OneSig
Commit(s)	Initial commit: 7847d304d873fb6d5edf17cc148c779ab9e3cd31 Final commit: 849dc48287d043d41c4a871129c03b329806cbe1

Audit Summary

Delivery Date	March 21, 2025
Audit Methodology	Static Analysis, Manual Review, 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	6	0	0	5	0	1
● Info	21	0	0	20	0	1

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: <i>High</i>	Impact: <i>Medium</i>	Impact: <i>Low</i>
Likelihood: <i>High</i>	● Critical	● High	● Medium
Likelihood: <i>Medium</i>	● High	● Medium	● Low
Likelihood: <i>Low</i>	● 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	Seed Invalidations Can Be Frontrun	Frontrunning	● Low	Acknowledged
L-02	Unexpected Execution Order With Reentrancy	Reentrancy	● Low	Resolved
L-03	Lacking Gas Validations Allows Censoring	Validation	● Low	Acknowledged
L-04	Network Forks Enable Replay Attacks	Replay	● Low	Acknowledged
L-05	Stale Transaction Execution Allowed When Sequencer Down	Warning	● Low	Acknowledged
L-06	Multiple Signed Merkle Roots Allows Unexpected Ordering	Unexpected Behavior	● Low	Acknowledged
I-01	Unexpected OneSig Balance Usage	Validation	● Info	Acknowledged
I-02	Signature Length Magic Number	Best Practices	● Info	Resolved
I-03	Excess Signatures Are Not Allowed	Validation	● Info	Acknowledged
I-04	Unnecessary returnData	Optimization	● Info	Acknowledged
I-05	Exclusion Of Smart Contract Signers Due To ECDSA Verification	Best Practices	● Info	Acknowledged
I-06	Missing Event Emission In The Receive Function	Best Practices	● Info	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
I-07	Time Drift Across Networks May Be Unexpected	Warning	● Info	Acknowledged
I-08	Low-Level Calls Do Not Validate Contract Code	Validation	● Info	Acknowledged
I-09	Stuck Nonce Blocks Following Transactions	Logical Error	● Info	Acknowledged
I-10	Unexpected Reinstatement Of Transactions	Validation	● Info	Acknowledged
I-11	Useful Error Data	Errors	● Info	Acknowledged
I-12	Unused VERSION Constant	Best Practices	● Info	Acknowledged
I-13	OneSigs Must Have Version Compatibility	Warning	● Info	Acknowledged
I-14	Token Transfers May Silently Fail	Unexpected Behavior	● Info	Acknowledged
I-15	Lacking EIP5267 Support	Compatibility	● Info	Acknowledged
I-16	Floating Pragma	Best Practices	● Info	Acknowledged
I-17	Incomplete verifyNSignatures Documentation	Documentation	● Info	Acknowledged
I-18	Unwieldy Seed Behavior	Warning	● Info	Acknowledged
I-19	Arbitrary Transaction Ordering Risk	MEV	● Info	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
I-20	Documentation Issues	Documentation	<div><div></div> Info</div>	Acknowledged
I-21	Threshold And Signers Should Be Consistent	Warning	<div><div></div> Info</div>	Acknowledged

L-01 | Seed Invalidations Can Be Frontrun

Category	Severity	Location	Status
Frontrunning	● Low	OneSig.sol	Acknowledged

Description

There is no access control which prevents arbitrary callers from using the `executeTransaction` function. This is by design in the `OneSig` system, however this behavior introduces several risks.

Notably, when the multisig signers wish to invalidate a previous merkle proof with a seed change it is possible for an arbitrary user to frontrun the execution of this seed change or even the signature collection process for the merkle root of this seed change and execute the transactions which the signers wish to invalidate.

This can result in unexpected outcomes from the transactions which were desired to be skipped and furthermore the unexpected increment of the nonce for the `OneSig` contract.

Recommendation

Consider tracking a set of executor addresses which are allowed to invoke the `executeTransaction` function. The list of executors can be initialized to the set of signers and extended/decreased if the signers wish with an `onlyMultiSig` function.

Resolution

USDT0 Team: Acknowledged.

L-02 | Unexpected Execution Order With Reentrancy

Category	Severity	Location	Status
Reentrancy	● Low	OneSig.sol: 165	Resolved

Description

The nonce for the OneSig contract is incremented directly before the execution of the set of external calls for the current transaction in the executeTransaction function. This successfully prevents replay of the current Transaction object through reentrancy in the executeTransaction function.

However if an arbitrary untrusted address were to gain control of the transaction execution during the execution of one of the external calls of the current Transaction, then that arbitrary untrusted address could re-enter and invoke the executeTransaction function again, now supplying the subsequent transaction in the same merkle root.

This would result in the transaction from the subsequent nonce, say nonce 2, being processed entirely in the middle of the external calls of the current nonce, say nonce 1.

This breaks the guarantee of atomicity for the external calls in a given Transaction object and ultimately could lead to unexpected behavior or even stolen funds/bricked systems.

Recommendation

Consider adding a nonReentrant modifier to the executeTransaction function.

Resolution

USDT0 Team: Resolved.

L-03 | Lacking Gas Validations Allows Censoring

Category	Severity	Location	Status
Validation	● Low	OneSig.sol: 182	Acknowledged

Description

The `executeTransaction` function may be invoked by any arbitrary address and does not include validation on the amount of gas that has been provided for each individual external call made as a part of the `Transaction` object.

Some external calls that are made from the `OneSig` contract may have different execution results depending on the amount of gas supplied.

For instance, if an invoked function includes a `try/catch` block around deeper logic, then the `catch` block can be trivially triggered by a malicious actor intentionally providing an insufficient amount of gas.

Additionally, there is no gas amount specified for each external call, therefore 63/64 of the transaction's remaining gas is automatically forwarded to each external call.

This allows prior external calls to consume gas in such a way that can impact the execution of later external calls in a `Transaction` object.

Recommendation

Consider including an optional amount of gas that is to be present and forwarded to each external call in a `Transaction` in the leaf data so that if signers must ensure the amount of gas provided to an external call then this can be validated and provided as such.

Resolution

USDT0 Team: Acknowledged.

L-04 | Network Forks Enable Replay Attacks

Category	Severity	Location	Status
Replay	● Low	Global	Acknowledged

Description

The OneSigId serves to differentiate deployments of OneSig Multisigs across chains, thus preventing replay.

However in the event that a network fork occurs, transactions can be replayed across instances of a OneSig contract on both resulting chains because the OneSig immutable value remains the same in both instances.

Recommendation

Consider including the block.chainId as an enforced portion of the OneSigId and updating the OneSigId if the block.chainid is determined to have changed from an originally cached value. Similarly to the logic used in the [domainSeparatorV4 function of the OpenZeppelin EIP712 contract](#).

Resolution

USDT0 Team: Acknowledged.

L-05 | Stale Transaction Execution Allowed When Sequencer Down

Category	Severity	Location	Status
Warning	● Low	Global	Acknowledged

Description

For L2 networks which use a sequencer it is possible for the `block.timestamp` used in a transaction to be out of date in cases where the sequencer is down for an extended period of time.

From the Arbitrum docs:

As mentioned, block timestamps are usually set based on the sequencer's clock. Because there's a possibility that the sequencer fails to post batches on the parent chain (i.e, Ethereum) for a period of time, it should have the ability to slightly adjust the timestamp of the block to account for those delays and prevent any potential reorganisations of the chain. To limit the degree to which the sequencer can adjust timestamps, some boundaries are set, currently to 24 hours earlier than the current time, and one hour in the future.

The behavior described here is that if the sequencer goes down for less than 24 hours, the sequencer can pick up any transactions from the Delayed Inbox and `block.timestamp` will be equal to the current time.

However, if the sequencer is down for more than 24 hours in extreme scenarios, then any `forceInclusion` transaction will be given a timestamp of the most recent L2 block, which is likely delayed significantly in this case, or the timestamp of the L1 from when the tx was placed in Delayed Inbox, whichever is greater. Consequently, a stale transaction could be executed past the expiry.

Recommendation

Be aware of this risk during an extended sequencer outage, and consider implementing a sequencer uptime check for L2 deployments which use a sequencer.

Resolution

USDT0 Team: Acknowledged.

L-06 | Multiple Signed Merkle Roots Allows Unexpected Ordering

Category	Severity	Location	Status
Unexpected Behavior	● Low	Global	Acknowledged

Description

In the event that multiple valid and signed merkle roots exist for the same OneSig safe that have overlapping nonces in their transactions, an arbitrary user can invoke the executeTransaction function mixing and matching transactions from each merkle root to create an unexpected execution outcome.

Recommendation

Ensure that users are made aware of this risk and thus are careful to not sign merkle roots with overlapping nonces for the same OneSig contract.

Resolution

USDT0 Team: Acknowledged.

I-01 | Unexpected OneSig Balance Usage

Category	Severity	Location	Status
Validation	● Info	OneSig.sol	Acknowledged

Description

The OneSig contract may be used to store native tokens on behalf of the signers for the contract, however because the executeTransaction function can be called by anyone this amount may be unexpectedly decreased in some scenarios.

If a merkle root which has been signed by the signers contains a function call with nonzero value, the signers may expect that the execution of executeTransaction would include that value as the msg.value of the transaction and thus not decrease the balance of the OneSig.

However an arbitrary user may invoke executeTransaction without providing any msg.value and thus decrease the OneSig native token balance by the transactions value. This may be unexpected for the signers and can deviate from exactly how they would like this transaction to be executed.

Recommendation

If this behavior is acceptable, then be sure to document it clearly for users of OneSig. Otherwise consider allowing the signers to specify as a part of the leaf data whether the native tokens should be supplied by the caller or the OneSig contract.

Alternatively, consider tracking a set of executor addresses which are allowed to invoke the executeTransaction function.

Resolution

USDT0 Team: Acknowledged.

I-02 | Signature Length Magic Number

Category	Severity	Location	Status
Best Practices	● Info	MultiSig.sol: 176, 183	Resolved

Description

In the MultiSig contract the value of 65 is referred to multiple times to indicate the length of the signatures provided. However rather than using a “magic number” to represent this length, a constant value can be declared in the contract.


Recommendation

Consider declaring a SIGNATURE_LENGTH constant to replace the bespoke instances of 65.

Resolution

USDT0 Team: Resolved.

I-03 | Excess Signatures Are Not Allowed

Category	Severity	Location	Status
Validation	 Info	OneSig.sol: 176	Acknowledged

Description

In the `verifyNSignatures` function the amount of signatures is validated to be exactly the `_threshold` provided. However in the documentation it is described that:

Modifying signers must require the same `signerThreshold` (or more) of signatures as executing a transaction.

The optionality for a `OneSig` contract to require more than the threshold to initiate a signer change does not currently exist in the EVM version and is specifically disallowed by the validation performed on the `_threshold` and `_signatures` values in the `verifyNSignatures` function.

Recommendation

Consider if this optionality of using a higher threshold for signer changes should be supported for the EVM version. If so then the validation and logic in `verifyNSignatures` should be refactored to allow greater than or equal to the threshold of signatures to be provided.

Resolution

USDT0 Team: Acknowledged.

I-04 | Unnecessary returnData

Category	Severity	Location	Status
Optimization	● Info	OneSig.sol: 182	Acknowledged

Description

The external call made in the `executeTransaction` function uses a `.call` invocation which automatically loads in the `returnData` into memory, regardless of if the `returnData` parameter is named in the returned tuple unpacking.

This is currently a waste of gas since the returned data is unused, and could potentially be a source of gas griefing for the executor in the case that the external call is made to an untrusted contract which may return a large amount of bytes unexpectedly.

Recommendation

Consider implementing a low level assembly call and specifying 0 as the length of return data to copy into memory.

Resolution

USDT0 Team: Acknowledged.

I-05 | Exclusion Of Smart Contract Signers Due To ECDSA Verification

Category	Severity	Location	Status
Best Practices	● Info	OneSig.sol: 184	Acknowledged

Description

The current implementation uses `ECDSA.recover` to verify signers, a method that only supports externally owned accounts (EOAs) signatures. As a result, any contract-based address cannot produce valid signatures under this scheme, effectively excluding multi signature wallets.

multi signature wallets and contract-based managers are incapable of directly signing in the same manner as EOAs, preventing them from providing their signatures to the `OneSig` contract.

This exclusion narrows the potential user base by disallowing participation from important categories such as DAO-managed treasuries or advanced contract-driven accounts, which are common in decentralized ecosystems.

Recommendation

Consider using the `SignatureChecker` library instead of `ECDSA` for signature verification. `OpenZeppelin's SignatureChecker.isValidSignatureNow` supports both EOAs and contract wallets.

This approach allows both types of accounts to produce valid signatures, enabling a broader range of automated and contract-based use cases without excluding any part of the user base.

Resolution

USDT0 Team: Acknowledged.

I-06 | Missing Event Emission In The Receive Function

Category	Severity	Location	Status
Best Practices	● Info	OneSig.sol: 260	Acknowledged

Description

The OneSig’s receive function currently allows native assets to be sent directly to the contract. While the receive function successfully collects and holds the native assets, there is no corresponding event to track deposits.

Recommendation

Consider emitting an event within the receive function that captures essential information, such as msg.sender, the msg.value and potentially the timestamp.

Resolution

USDT0 Team: Acknowledged.

I-07 | Time Drift Across Networks May Be Unexpected

Category	Severity	Location	Status
Warning	● Info	Global	Acknowledged

Description

The expiration time is associated with the signatures for the merkle root, hence the expiry applies to all transactions equally. However, transactions within the tree can be intended for different chains, and chains do not have a perfectly synchronized `block.timestamp` across them.

When if `(block.timestamp > _expiry) revert MerkleRootExpired();` is validated upon transaction execution, situations may arise where a transaction is meant to be expired on all chains, but is still executable on a subset of chains.

Recommendation

Be aware of this risk.

Resolution

USDT0 Team: Acknowledged.

I-08 | Low-Level Calls Do Not Validate Contract Code

Category	Severity	Location	Status
Validation	• Info	OneSig.sol: 182	Acknowledged

Description

The `executeTransaction` function uses low-level calls (e.g., `address.call{ value: ... }(data)`) without checking whether the target `to` address is a deployed smart contract that actually implements the function being called.

As a result, sending a call to an Externally Owned Account (EOA) or a non-existent contract would still succeed, even though the call effectively does nothing.

This can create a misleading indication of success, similar to historical issues (e.g. in `Solady's safeTransfer` library), where the transaction appears to execute successfully but no real action took place.

Recommendation

Consider extending the `Call` struct to include a `bool isSmartContract` field that signals whether the target should contain code. During execution, if `isSmartContract` is set to `true`, validate that `to.code.length > 0` (available in Solidity 0.8.18+).

If the address lacks code, revert the transaction. Conversely, if `isSmartContract` is `false`, allow the call to proceed.

Resolution

USDT0 Team: Acknowledged.

I-09 | Stuck Nonce Blocks Following Transactions

Category	Severity	Location	Status
Logical Error	● Info	Global	Acknowledged

Description

Currently function `executeTransaction` only increments the nonce if the transaction call was successful. This poses an issue for all transactions encoded with a future nonce, in anticipation of the prior execution going through successfully.

If a single transaction is unable to be executed, all future transactions (leaves) must be updated and the merkle root must be updated so those following transactions would be executed.

Recommendation

Consider incrementing the nonce even if the transaction call was unsuccessful. However, it is important to take caution with this approach since arbitrary users can call `executeTransaction`.

An arbitrary user may not send enough value, cause the transaction to fail purposefully, and then allow all the following transactions to move on in a potentially unintended state.

This may be remedied by having a set of trusted executors to call `executeTransaction` or adding validation for all relevant values such as `msg.value` or `gasleft()`.

Furthermore, it would be important to ensure `eth_estimateGas` can function appropriately if `executeTransaction` no longer reverts on failure.

Resolution

USDT0 Team: Acknowledged.

I-10 | Unexpected Reinstatement Of Transactions

Category	Severity	Location	Status
Validation	● Info	Global	Acknowledged

Description

When a seed is changed to invalidate a particular merkle root there is no logic to blacklist the seed from ever being reassigned.

As a result it is possible for signers of a multisig to accidentally reinstate a seed which was previously used and inadvertently reinstate a previously revoked merkle root.

Recommendation

Consider tracking a blacklist of past seeds so that they cannot be accidentally re-instated. Otherwise document this risk to users.

Resolution

USDT0 Team: Acknowledged.

I-11 | Useful Error Data

Category	Severity	Location	Status
Errors	<div><div></div> Info</div>	OneSig.sol: 187	Acknowledged

Description

The ExecutionFailed error contains the index of the call but neglects to include the nonce of the Transaction being executed. This data may be useful when debugging transaction failures or examining simulations.

Recommendation

Consider including the Transaction nonce in the ExecutionFailed data.

Resolution

USDT0 Team: Acknowledged.

I-12 | Unused VERSION Constant

Category	Severity	Location	Status
Best Practices	● Info	OneSig.sol: 19, 50	Acknowledged

Description

In the OneSig contract the VERSION constant value is declared and unused. The same version string is repeated in the declaration of the DOMAIN_SEPARATOR constant.

This poses a risk if the VERSION constant were to be updated without changing the DOMAIN_SEPARATOR to match.

Recommendation

Use the VERSION constant in the declaration of the DOMAIN_SEPARATOR value.

Resolution

USDT0 Team: Acknowledged.

I-13 | OneSigs Must Have Version Compatibility

Category	Severity	Location	Status
Warning	<div><div></div> Info</div>	OneSig.sol: 50	Acknowledged

Description

The DOMAIN_SEPARATOR includes the OneSig version as a field, therefore all OneSig instances that are expected to be used in unison across networks by the same signers must have the same VERSION applied.

Recommendation

Be aware of this requirement and document it for users.

Resolution

USDT0 Team: Acknowledged.

I-14 | Token Transfers May Silently Fail

Category	Severity	Location	Status
Unexpected Behavior	● Info	OneSig.sol: 182	Acknowledged

Description

The external call return data is not validated in the `executeTransaction` function which can result in silent failure of transactions which return false on failure instead of reverting.

For example, if the signers were to initiate a withdrawal of USDT which failed for some reason the `executeTransaction` function call would still succeed and emit the `TransactionExecuted` event while the USDT transfer actually failed.

Recommendation

Be aware of this behavior and document it for users.

Resolution

USDT0 Team: Acknowledged.

I-15 | Lacking EIP5267 Support

Category	Severity	Location	Status
Compatibility	● Info	Global	Acknowledged

Description

The OneSig contract does not expose functionality to read the signing domain. This functionality was standardized with [EIP5267](#) which introduces the eip712Domain function.

Recommendation

Consider implementing the EIP5267 defined interface for better discoverability for signers.

Resolution

USDT0 Team: Acknowledged.

I-16 | Floating Pragma

Category	Severity	Location	Status
Best Practices	● Info	Global	Acknowledged

Description

The OneSig and MultiSig contracts use a floating pragma version instead of a fixed pragma. This can introduce unexpected outcomes since the contracts can be compiled across multiple Solidity versions which may have slight known or unknown differences in behavior.

Recommendation

Remove this ambiguity by using a fixed pragma version.

Resolution

USDT0 Team: Acknowledged.

I-17 | Incomplete verifyNSignatures Documentation

Category	Severity	Location	Status
Documentation	● Info	MultiSig.sol: 173	Acknowledged

Description

The NatSpec for the verifyNSignatures function lists the ways in which the verifyNSignatures function reverts. However it does not include reverts which occur due to an invalid signature verification with the ECDSA library.

Namely, the verifyNSignatures function can also revert when the provided s value for the signature is from the top half of the range or when the recovered signer address is the zero address.

Recommendation

Consider including these revert scenarios in the function documentation.

Resolution

USDT0 Team: Acknowledged.

I-18 | Unwieldy Seed Behavior

Category	Severity	Location	Status
Warning	● Info	Global	Acknowledged

Description

The documentation in the `PROTOCOL.md` file indicates that seeds are used to prevent signature replay, however this cannot be the case.

Seeds must be identical across `OneSig` instances which are to be used in unison, otherwise a new signature for the merkle tree would have to be generated for each chain.

This results in an unwieldy use-case when the signers want to invalidate the merkle on one chain but not on all chains. The signer must first update the seed on one chain and then execute the desired transactions on all other chains.

Finally the signer must update the seed across all of the companion deployments of `OneSig` so they are all in unison again.

Recommendation

Signers can instead execute new merkle on a target chain to advance the nonce past the desired transaction to skip. This method allows for granularity on the chain and transactions which the signers desire to skip.

Without requiring that all seeds are updated across all companion `OneSig` deployments. The seed should then only be used to invalidate a merkle root across all companion `OneSig` deployments at once, and the seed should always be updated in unison and never vary across deployments.

Resolution

USDT0 Team: Acknowledged.

I-19 | Arbitrary Transaction Ordering Risk

Category	Severity	Location	Status
MEV	● Info	OneSig.sol	Acknowledged

Description

The `executeTransaction` function may be called by any arbitrary address once the merkle signatures are available. An arbitrary actor may take advantage of this by controlling the ordering in which actions take place with respect to the `MultiSig` transaction.

For example, if the multisig aims to execute a swap with a slippage of 5%, an arbitrary user can guarantee that they are able to sandwich that swap by executing it in a multical with their own frontrun and backrun swaps.

In another scenario, the signers of a `OneSig` multisig could have signed off on an important protocol altering transaction which should only be applied when the protocol is paused. The paused state could be controlled by a separate `PAUSER_ROLE` address which is not the `OneSig` multisig.

In this case an arbitrary address can execute the `OneSig` action before the protocol is paused and potentially cause damage this way.

Recommendation

Be aware of the risk involved in allowing arbitrary executors for signed transactions. If this risk should be mitigated then consider implementing a whitelisted set of executors. Otherwise be sure to clearly document this risk to users of `OneSig`.

Resolution

USDT0 Team: Acknowledged.

I-20 | Documentation Issues

Category	Severity	Location	Status
Documentation	● Info	PROTOCOL.md	Acknowledged

Description

In the PROTOCOL.md file there are several misleading comments or grammatical errors:

Firstly, it is mentioned “Note: Although the domain-separator parameters are static across different deployments of OneSig, the risk of replay/unintentional signing attacks is mitigated due to the use of unique seeds, unique signers, and incrementing nonces.”

But not mentioned here is that the contract address is included in the leaf encoding, which removes the ability for replay across OneSig deployments on the same chain.

The PROTOCOL.md file still mentions the ChainID which has been replaced with the OneSigId. The SignerWithAddress interfaces is misspelled as SingerWithAddress. unsigned is misspelled as unisigned on line 86.

The description for the contract entry in the leaf encoding section reads, a 32 byte identifier representing a deployment which is a copy of the description for the oneSigId entry.

Recommendation

Consider correcting these documentation issues.

Resolution

USDT0 Team: Acknowledged.

I-21 | Threshold And Signers Should Be Consistent

Category	Severity	Location	Status
Warning	<div><div></div> Info</div>	Global	Acknowledged

Description

At the contract level it is possible for companion deployments of OneSigs to have a different threshold and varying signer sets, however this may present complications from a UI/UX perspective.

Recommendation

For maximum simplicity and to avoid potential confusion during the transaction execution process, consider requiring on the frontend that all companion OneSig deployments have the same signer threshold and set.

Resolution

USDT0 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 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>