Competitive Security Assessment

Mantle

Jul 17th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	Mantle
Platform & Language	Solidity and Go
Codebase	• https://github.com/mantlenetworkio/mantle/ • audit commit - 3e2b6dbcac7df08353d81d53ee41405211305432 • final commit - c4a8fd856aab7fd6eedb0c8bc2a8129d252de24b
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

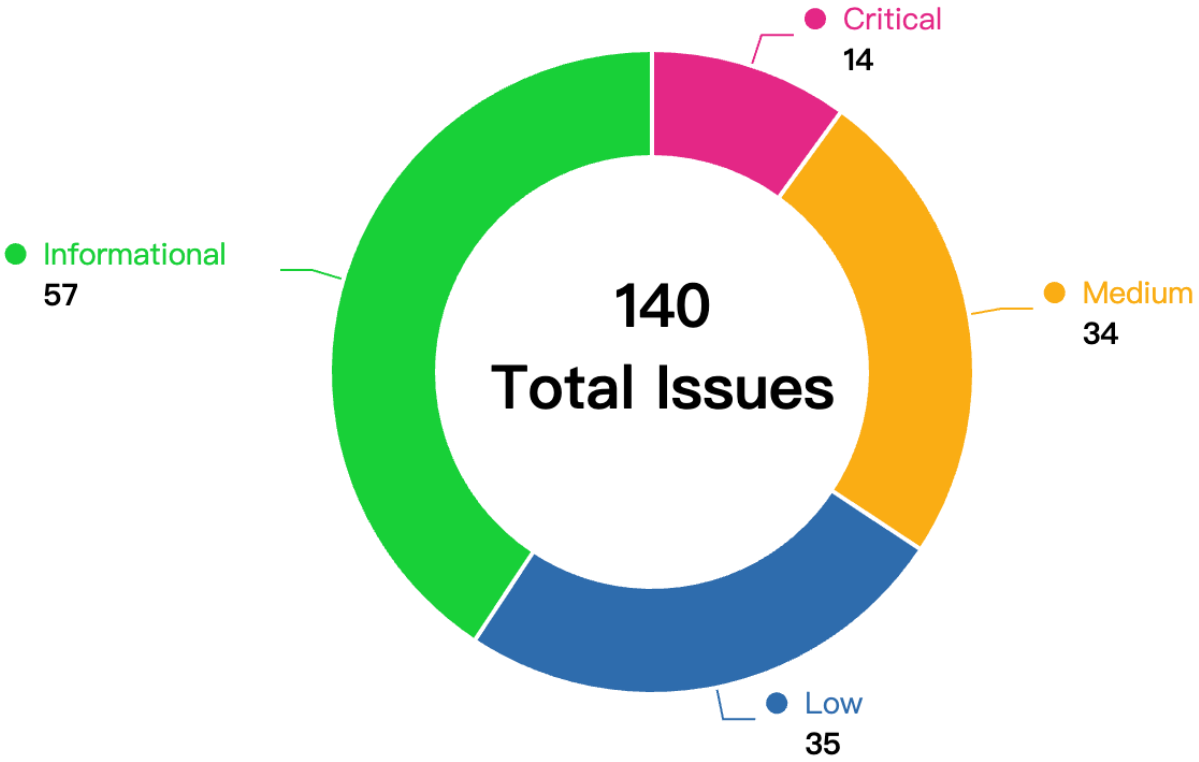
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	14	0	4	5	1	4
Medium	34	0	2	14	5	13
Low	35	0	4	19	3	9
Informational	57	0	22	28	1	6

Audit Scope

File	Commit Hash
batch-submitter directory	3e2b6dbcac7df08353d81d53ee41405211305432
bss-core directory	3e2b6dbcac7df08353d81d53ee41405211305432
gas-oracle directory	3e2b6dbcac7df08353d81d53ee41405211305432
l2geth directory	3e2b6dbcac7df08353d81d53ee41405211305432
mt-batcher directory	3e2b6dbcac7df08353d81d53ee41405211305432
mt-challenger directory	3e2b6dbcac7df08353d81d53ee41405211305432
packages directory	3e2b6dbcac7df08353d81d53ee41405211305432
tss directory	3e2b6dbcac7df08353d81d53ee41405211305432

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
MTC-1	Deadlock Risk due to Blocking on an Empty Channel	Logical	Critical	Declined	BradMoonUESTC
MTC-2	Insecure Seed Generation in `DerivePrivateKey` Function	Logical	Critical	Acknowledged	BradMoonUESTC
MTC-3	Logical error:Attempting to return data using a non-pointer type variable.	Logical	Critical	Declined	zircon

MTC-4	Logical error:ERC20 bridge arbitrage attack	Logical	Critical	Acknowledged	zircon, BradMoonUESTC
MTC-5	Potential Logic Error in Missed Batch Bit Array Handling	Logical	Critical	Declined	BradMoonUESTC
MTC-6	Proposers can roll back the chain to the past roll-backed state again due to TSS signature replay	Signature Forgery or Replay	Critical	Declined	iczc
MTC-7	Reentrancy risk:Stop using Solidity's transfer()	Reentrancy	Critical	Fixed	zircon
MTC-8	The TSS signature server does not have an authentication mechanism	Privilege Related	Critical	Fixed	zircon, iczc
MTC-9	Unintended Dos Attack due to Direct Transfer	DOS	Critical	Fixed	BradMoonUESTC
MTC-10	Burned sequencer fees can be spent normally due to the burner address can be set to any address	Privilege Related	Medium	Declined	iczc
MTC-11	Inadequate BIP39 seed phrase security	Logical	Medium	Fixed	BradMoonUESTC
MTC-12	Income of uptime slash type should be zero	Logical	Medium	Fixed	zircon
MTC-13	Incomplete Multi-Private Key Handling in `PrivateKeySignerFn` Function	Logical	Medium	Fixed	BradMoonUESTC
MTC-14	Incorrect Initialization of Validator's Signing Information	Logical	Medium	Declined	BradMoonUESTC
MTC-15	Lack of Input Validation in `setGroupPublicKey`	Logical	Medium	Mitigated	BradMoonUESTC, zircon
MTC-16	Missing 0 address check in `BVM_EigenDataLayrChain::updateSequencerAddress()`	Logical	Medium	Fixed	Hupixiong3
MTC-17	Not revert snapshot for a failed updateReward() call	Logical	Medium	Fixed	iczc

MTC-18	Penalized Token Distribution Recipient Exception	Logical	Medium	Fixed	BradMoonUESTC
MTC-19	Potential Manipulation of Reward Amount in `claimReward` function	Logical	Medium	Declined	BradMoonUESTC
MTC-20	Potential Overflow Risk and Hardcoded Label Values	Integer Overflow and Underflow	Medium	Mitigated	BradMoonUESTC
MTC-21	Punished Address Exception	Logical	Medium	Fixed	BradMoonUESTC
MTC-22	Replay Attack	Logical	Medium	Mitigated	BradMoonUESTC
MTC-23	Signature Replay:Potential Replay Attack in `submitReRollUpInfo` Function	Signature Forgery or Replay	Medium	Declined	BradMoonUESTC
MTC-24	Status Update Unclear	Logical	Medium	Declined	BradMoonUESTC
MTC-25	Uncaught errors returned by a function	Logical	Medium	Fixed	zircon
MTC-26	Unhandled Error in Node Update Routine	Logical	Medium	Declined	BradMoonUESTC
MTC-27	Use of weak random number generator	Weak Sources of Randomness	Medium	Declined	Hupixiong3
MTC-28	modifier onlyEOA() can be bypassed via depositETHTo() function	Logical	Medium	Declined	comcat
MTC-29	refund procedure from L2 to L1 can be manipulated to mint gas token	Logical	Medium	Declined	comcat
MTC-30	Address parsing does not support checksum addresses	Logical	Low	Acknowledged	BradMoonUESTC
MTC-31	Burned sequencer fees can be spent normally after the burn mode close	Logical	Low	Mitigated	iczc
MTC-32	Ignored Return Value	Logical	Low	Declined	BradMoonUESTC
MTC-33	Inconsistent Logic in Private Key Generation	Logical	Low	Acknowledged	BradMoonUESTC

MTC-34	Incorrect Handling of Absent Validator's Signing Information	Logical	Low	Declined	BradMoonUESTC
MTC-35	Incorrect Identification of Contract Accounts as EOAs in Address.isContract()	Logical	Low	Fixed	BradMoonUESTC
MTC-36	Ineffective checkBalance verification	Logical	Low	Fixed	Hupixiong3
MTC-37	Lack of address zero Input Validation and Error Handling	Code Style	Low	Fixed	BradMoonUESTC
MTC-38	Missing condition check	Logical	Low	Fixed	Hupixiong3
MTC-39	Missing limiting condition	Logical	Low	Fixed	Hupixiong3
MTC-40	Non-Asynchronous Execution of the HTTP Server	Logical	Low	Declined	BradMoonUESTC
MTC-41	Owner can take out all assets in TssRewardContract	Privilege Related	Low	Declined	iczc
MTC-42	The _threshold of the tss signature is allowed to be set to 0	Logical	Low	Mitigated	iczc
MTC-43	Unintended Usage of `NoSend` Option during Transaction Creation	Logical	Low	Declined	BradMoonUESTC
MTC-44	Use call for transferring funds instead of using transfer	Language Specific	Low	Fixed	Hupixiong3
MTC-45	loop logic is bypassed in `resetRollupBatchData()`	Logical	Low	Fixed	Hupixiong3
MTC-46	Redundant code:The getTssGroupUnJailMembers function does unnecessarily loop	Code Style	Informational	Acknowledged	iczc
MTC-47	Duplicate msg.sender check codes	Code Style	Informational	Fixed	iczc
MTC-48	Gas Optimization:Duplicate judge in `BVM_SequencerFeeVault::withdraw::minWithdrawalAmount`	Gas Optimization	Informational	Fixed	iczc
MTC-49	Gas Optimization:Move the l2Fee calculation outside the if statement	Gas Optimization	Informational	Fixed	iczc
MTC-50	Gas Optimization:Optimization of Loop Structures	Gas Optimization	Informational	Acknowledged	Hupixiong3

MTC-51	Gas Optimization:Simplify array operation in `TssGroupManager.sol::setTssGroupMember`	Gas Optimization	Informational	Declined	BradMoonUESTC
MTC-52	Gas Optimization:Simplify removeActiveTssMembers to avoid `out of gas`	Gas Optimization	Informational	Fixed	BradMoonUESTC
MTC-53	Gas Optimization:Unnecessary Boolean Comparison in `TssGroupManager::setGroupPublicKey`	Gas Optimization	Informational	Fixed	BradMoonUESTC
MTC-54	Gas Optimization:Using immutable marked as much as possible	Gas Optimization	Informational	Fixed	Hupixiong3
MTC-55	Incorrect comment in `BVM_SequencerFeeVault`	Code Style	Informational	Fixed	iczc
MTC-56	Logical error:The error message is incorrect	Logical	Informational	Fixed	Hupixiong3
MTC-57	Missing access control in `L1StandardBridge:: initialize`	Privilege Related	Informational	Mitigated	BradMoonUESTC
MTC-58	Missing event record	Code Style	Informational	Fixed	BradMoonUESTC
MTC-59	Missing parameter check in `L1StandardBridge::_initiateERC20Deposit`	Gas Optimization	Informational	Acknowledged	BradMoonUESTC
MTC-60	Potential information leakage through logs	Logical	Informational	Acknowledged	BradMoonUESTC
MTC-61	Predeployed wrapped token with incorrect metadata	Logical	Informational	Declined	iczc
MTC-62	Redundant code:Duplicate function in `BVM_EigenDataLayrChain`	Code Style	Informational	Acknowledged	Hupixiong3
MTC-63	Redundant code:Redundant check	Code Style	Informational	Fixed	iczc, Hupixiong3
MTC-64	Redundant code:Redundant initialization assignment	Code Style	Informational	Acknowledged	Hupixiong3
MTC-65	Redundant code:Unnecessary check	Gas Optimization	Informational	Fixed	Hupixiong3

MTC-66	Redundant code:Unnecessary duplicate functions	Logical	Informational	Fixed	Hupixiong3
MTC-67	Unused predeploy whitelist contract	Code Style	Informational	Acknowledged	iczc
MTC-68	Use `bool` type variable to control the switch in BVM_GasPriceOracle	Code Style	Informational	Acknowledged	iczc
MTL-1	A malicious user can permanently lock staker's money in `Rollup` contracts	Logical	Critical	Fixed	biakia
MTL-2	An invalid assertion can also be confirmed	Logical	Critical	Mitigated	biakia
MTL-3	Potential serialization errors in `batch-submitter` module	Logical	Critical	Acknowledged	biakia
MTL-4	There is a risk of funds being locked	Logical	Critical	Fixed	yekong
MTL-5	`BASEFEE` opcode will fail to execute	Language Specific	Critical	Acknowledged	Secure3
MTL-6	modifier `onlyEOA()` can be bypassed via `depositETHTo()` function	Logical	Medium	Declined	Secure3
MTL-7	BVM_EigenDataLayrChain.proveFraud uses a dangerous strict equality	Code Style	Medium	Declined	Secure3
MTL-8	Division by Zero and Potential Incorrect Behavior Due to Missing Input Validation	Logical	Medium	Fixed	yekong
MTL-9	Function does not accept a return value, but `return` is used	Logical	Medium	Acknowledged	yekong
MTL-10	Incorrect returned value in `GetStateBatch` function	Logical	Medium	Fixed	biakia
MTL-11	Loss of user assets in `_initiateETHDeposit` function	Logical	Medium	Acknowledged	yekong
MTL-12	Potential Gas Wasting and Transfer Failure Handling in the claimReward function	Logical	Medium	Declined	yekong

MTL-13	Potential Incorrect Behavior due to Default Value in Deposits Mapping	Logical	Medium	Fixed	yekong
MTL-14	Potential Reward Overpayments in `TssRewardContract`	Logical	Medium	Fixed	biakia
MTL-15	Potential Risk of Duplicate Entries in setTssGroupMember Function	Logical	Medium	Mitigated	yekong
MTL-16	Potential Risks with Misbehaving _l2Token Contracts in finalizeDeposit	Logical	Medium	Mitigated	yekong
MTL-17	Reentrancy in BVM_EigenDataLayrChain.confirmData	Reentrancy	Medium	Declined	Secure3
MTL-18	Unchecked return value may make the assertion fail to complete	Logical	Medium	Fixed	biakia
MTL-19	`TssRewardContract::totalAmount` isn't reduced in `claimReward` function and `updateReward` may fail	Logical	Medium	Fixed	biakia
MTL-20	Anyone can call `removeStake` in `Rollup` contract	Logical	Low	Fixed	biakia
MTL-21	BVM_EigenDataLayrChain.submitReRollUpInfo uses timestamp for comparisons	Code Style	Low	Declined	Secure3
MTL-22	Inconsistency between comments and implementation in function `completeChallenge`	Code Style	Low	Fixed	biakia
MTL-23	Incorrect Identification of Contract Accounts as EOAs in Address.isContract()	Logical	Low	Fixed	yekong
MTL-24	Incorrect buffer for gas limit	Logical	Low	Declined	biakia
MTL-25	Incorrect check for `threshold`	Logical	Low	Declined	biakia
MTL-26	Lack of address zero Input Validation	Code Style	Low	Fixed	Secure3, yekong
MTL-27	Lack of check for `db`	Code Style	Low	Fixed	biakia

MTL-28	Logic risk in `BVM_EigenDataLayrChain` contract `submitReRollUpInfo` function	Logical	Low	Declined	yekong
MTL-29	Maximum Limit Issue with _batchPublicKey in setTssGroupMember Function	Logical	Low	Mitigated	yekong
MTL-30	Parser.parse(bytes[],uint256,uint256). provenString is a local variable never initialized	Code Style	Low	Fixed	Secure3
MTL-31	Response Id not checked in function `unmarshalResponseBytes`	Logical	Low	Fixed	biakia
MTL-32	Returned value of function `SendMsg` not checked	Logical	Low	Fixed	biakia
MTL-33	Should avoid account nonce upper the limit `2^64-1`	Integer Overflow and Underflow	Low	Fixed	Secure3
MTL-34	`If` condition that can never be met in `advanceStake` function	Logical	Low	Fixed	biakia
MTL-35	`L1ChugSplashProxy::onlyWhenNotPaused` does not take effect.	Logical	Low	Acknowledged	Secure3
MTL-36	`L1StandardBridge` does not support fee-on-transfer tokens	Logical	Low	Acknowledged	biakia
MTL-37	`checkBalance` is not working as expected for `TssRewardContract` contract `updateReward` function	Logical	Low	Fixed	alansh
MTL-38	should not ignore error in NewStateTransition	Code Style	Low	Fixed	alansh
MTL-39	no receive ether function with fallback	Code Style	Informational	Declined	Secure3
MTL-40	Division Before Multiplication	Code Style	Informational	Acknowledged	biakia
MTL-41	Gas Optimization : Replace Hardcoded require with revert	Gas Optimization	Informational	Fixed	yekong
MTL-42	Gas Optimization : Streamline require Statements and Boolean Usage	Gas Optimization	Informational	Fixed	yekong

MTL-43	Gas Optimization: ChugSplashDictator.bitAddressSlotKey should be immutable	Gas Optimization	Informational	Fixed	Secure3
MTL-44	Gas Optimization: Inefficient Deletion of Elements in Large Arrays	Gas Optimization	Informational	Fixed	yekong
MTL-45	Gas Optimization: It is not necessary to use SafeMath in solidity version 0.8+	Gas Optimization	Informational	Acknowledged	biakia, yekong
MTL-46	Gas Optimization: Remove unused Hardhat imports	Gas Optimization	Informational	Acknowledged	biakia
MTL-47	Gas Optimization: `SlashType.nothing` is useless in `TssStakingSlashing` contract	Gas Optimization	Informational	Fixed	alansh
MTL-48	Gas Optimization: `_distributeTssReward` function	Gas Optimization	Informational	Fixed	biakia
MTL-49	Gas Optimization: `firstSupportedInterface` and `secondSupportedInterface` should be declared as constant in `L2StandardERC20` contract `supportsInterface` function	Gas Optimization	Informational	Acknowledged	alansh
MTL-50	Gas Optimization: `isEqual` function doesn't need to be public in `TssStakingSlashing`/`TssGroupManager` contract	Gas Optimization	Informational	Fixed	alansh
MTL-51	Gas Optimization: `publicKeyToAddress` is more fit for a library function	Gas Optimization	Informational	Fixed	alansh
MTL-52	Gas Optimization: could quit the loop earlier in `TssGroupManager` contract `removeMember` function	Gas Optimization	Informational	Fixed	alansh
MTL-53	Gas Optimization: in `TssStakingSlashing` contract `transformDeposit` function	Gas Optimization	Informational	Fixed	alansh

MTL-54	Gas Optimization: no need to call `_isValidBatchHeader` in `StateCommitmentChain` contract `deleteStateBatch` function	Gas Optimization	Informational	Acknowledged	alansh
MTL-55	Gas Optimization: no need to check all four variables in `TssStakingSlashing` contract `staking` function	Gas Optimization	Informational	Fixed	alansh
MTL-56	Gas Optimization: no need to inherit `Initializable` in `TssStakingSlashing` contract	Gas Optimization	Informational	Acknowledged	alansh
MTL-57	Gas Optimization: no need to save `pledgor` into storage in `TssStakingSlashing` contract `staking` function	Gas Optimization	Informational	Fixed	alansh
MTL-58	Gas Optimization: unnecessary variable `sentMessages` in `L2CrossDomainMessenger` contract `sendMessage` function	Gas Optimization	Informational	Declined	alansh
MTL-59	Gas Optimization:Cache array length out of the loop	Gas Optimization	Informational	Acknowledged	biakia
MTL-60	Gas Optimization:Unnecessary Gas Consumption for Zero ERC20 Deposits	Gas Optimization	Informational	Acknowledged	yekong
MTL-61	Gas optimization: `TssGroupManager::removeActiveTssMembers` function	Gas Optimization	Informational	Fixed	biakia
MTL-62	Incorrect error message in `accessors_chain.go`	Code Style	Informational	Acknowledged	biakia
MTL-63	Miss emitting event in important state-changing setter functions	Code Style	Informational	Acknowledged	yekong, biakia
MTL-64	Redundant function	Logical	Informational	Acknowledged	biakia
MTL-65	Unused variable: `BVM_GasPriceOracle::sccAddress`	Code Style	Informational	Acknowledged	biakia
MTL-66	Use `call` instead of `transfer` to transfer ETH	Language Specific	Informational	Fixed	yekong, biakia, Secure3

MTL-67	Use `calldata` instead of `memory`	Gas Optimization	Informational	Fixed	biakia
MTL-68	`dust`/`accu` calculation can be simplified in `TssRewardContract` contract `claimReward` function	Code Style	Informational	Fixed	alansh
MTL-69	missing calling `__ReentrancyGuard_init` in `TssGroupManager` contract `initialize` function	Code Style	Informational	Acknowledged	alansh
MTL-70	should check `_publicKey` exist before updating `tssActiveMemberInfo` in `TssGroupManager` contract `memberUnJail` function	Logical	Informational	Declined	alansh
MTL-71	should rename `RANDOM` to `PREVRANDAO` according to eip-4399	Language Specific	Informational	Acknowledged	Secure3
MTL-72	uses assembly in libraries	Code Style	Informational	Declined	Secure3

MTC-1:Deadlock Risk due to Blocking on an Empty Channel

Category	Severity	Client Response	Contributor
Logical	Critical	Declined	BradMoonUESTC

Code Reference

- code/mt-batcher/mt_batcher.go#L44

```
44:             <-(chan struct{})(nil)
```

Description

BradMoonUESTC : The code provided exhibits a potential issue in the function `Main`. Specifically, this line of code: `<-(chan struct{})(nil)` is blocking on an empty channel, which is never written to or closed elsewhere in the application. This operation will block indefinitely, leading to a potential deadlock situation. It causes the function to pause its execution and might result in the whole application being stuck in this state without any possibility to resume or terminate gracefully.

Recommendation

BradMoonUESTC : It is recommended to avoid blocking on an empty channel. If waiting for a specific condition to be met or for some goroutines to finish their execution is required, consider using a dedicated signaling mechanism, such as a channel which is actually written to or closed when the condition is met or the goroutines finish their execution. Another approach could be to use the `select` construct with a default case or timeout to avoid indefinite blocking. It's also important to review and understand the purpose of this blocking operation. If it's not necessary, consider removing it to prevent potential deadlocks.

Client Response

Declined.Please note, the Main function will not cause a block. It initiates goroutines such as RollupMainWorker, CheckConfirmedWorker, and RollUpFeeWorker. These goroutines are responsible for blocking the rollup transaction data, checking if the data requires re-rollup, and submitting DA fee data. The Main process is designed to block, anticipating other child goroutines to exit before it does, thereby avoiding any deadlock. It doesn't obstruct the execution of any logic.

MTC-2:Insecure Seed Generation in `DerivePrivateKey` Function

Category	Severity	Client Response	Contributor
Logical	Critical	Acknowledged	BradMoonUESTC

Code Reference

- code/mt-batcher/services/common/crypto.go#L59

```
59:     seed, err := bip39.NewSeedWithErrorChecking(mnemonic, "")
```

Description

BradMoonUESTC : The `DerivePrivateKey` function generates a private key from a BIP39 mnemonic and an HD key path. However, during the seed generation process, an empty string is used as the mnemonic password. This means that anyone possessing the mnemonic could obtain the identical seed and, consequently, the identical private key. If the mnemonic were to be compromised, anyone could gain access to the corresponding wallet, presenting a severe security risk.

```
seed, err := bip39.NewSeedWithErrorChecking(mnemonic, "")
```

Recommendation

BradMoonUESTC : Typically, a user-provided password should be used during seed generation. If the user doesn't provide a password, they should be prompted to input one, or be explicitly informed that the generated private key will be unprotected. Therefore, this function should be revised to ensure the security of the generated private key. This could involve adding an additional function parameter for a user-supplied password or improving the error handling and user notification process when no password is provided.

Client Response

Acknowledged.Keys were held in plain text in development and testing environment only. We have removed it and will save it offline ongoing. For Mantle mainnet and testnet, sensitive information is shared in multiple parts, encrypted by AWS KMS, and separately stored in AWS SecretsManager. We use CloudHSM for secp256k1 signing.

MTC-3:Logical error:Attempting to return data using a non-pointer type variable.

Category	Severity	Client Response	Contributor
Logical	Critical	Declined	zircon

Code Reference

- code/tss/node/tsslib/conversion/conversion.go#L56-L65

```
56:     for id, party := range parties {
57:             peerID, err := GetPeerIDFromPartyID(party)
58:             if err != nil {
59:                     return err
60:             }
61:             partyIDtoP2PID[id] = peerID
62:     }
63:     return nil
64:}
65:
```

Description

zircon : Look at the function following:

```
func SetupIDMaps(parties map[string]*tss.PartyID, partyIDtoP2PID map[string]peer.ID) error {
        for id, party := range parties {
                peerID, err := GetPeerIDFromPartyID(party)
                if err != nil {
                        return err
                }
                partyIDtoP2PID[id] = peerID
        }
        return nil
}
```

`partyIDtoP2PID` is a parameter of the `SetupIDMaps` function, but it is not a pointer type. When attempting to assign a value to it using `partyIDtoP2PID[id] = peerID`, the value will not be changed in the calling function since it is not a pointer.

The impact is that it can cause errors in the TSS P2P ID.

Recommendation

zircon : Consider below fix in the `SetupIDMaps()` function

```
func SetupIDMaps(parties map[string]*tss.PartyID, partyIDtoP2PID map[string]*peer.ID) error {
        for id, party := range parties {
                peerID, err := GetPeerIDFromPartyID(party)
                if err != nil {
                        return err
                }
                partyIDtoP2PID[id] = peerID
        }
        return nil
}
```

Client Response

Declined.Although 'partyIDtoP2PID' is not defined as a pointer type, the map data structure can still be assigned properly. Please note, however, this code has been removed in the latest version.

```
func SetupIDMaps(parties map[string]*tss.PartyID, partyIDtoP2PID map[string]peer.ID) error {
    for id, party := range parties {
    peerID, err := GetPeerIDFromPartyID(party)
   if err != nil {
          return err
    partyIDtoP2PID[id] = peerID
}
    return nil
}
```

MTC-4:Logical error:ERC20 bridge arbitrage attack

Category	Severity	Client Response	Contributor
Logical	Critical	Acknowledged	zircon, BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L2/messaging/L2StandardBridge.sol#L149-L196
- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L177-L230
- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L268-L284

```
149:     function finalizeDeposit(
150:         address _l1Token,
151:         address _l2Token,
152:         address _from,
153:         address _to,
154:         uint256 _amount,
155:         bytes calldata _data
156:     ) external virtual onlyFromCrossDomainAccount(l1TokenBridge) {
157:         // Check the target token is compliant and
158:         // verify the deposited token on L1 matches the L2 deposited token representation here
159:         if (
160:             // slither-disable-next-line reentrancy-events
161:             ERC165Checker.supportsInterface(_l2Token, 0x1d1d8b63) &&
162:             _l1Token == IL2StandardERC20(_l2Token).l1Token()
163:         ) {
164:             // When a deposit is finalized, we credit the account on L2 with the same amount of
165:             // tokens.
166:             // slither-disable-next-line reentrancy-events
167:             IL2StandardERC20(_l2Token).mint(_to, _amount);
168:             // slither-disable-next-line reentrancy-events
169:             emit DepositFinalized(_l1Token, _l2Token, _from, _to, _amount, _data);
170:         } else {
171:             // Either the L2 token which is being deposited-into disagrees about the correct add
ress
172:             // of its L1 token, or does not support the correct interface.
173:             // This should only happen if there is a  malicious L2 token, or if a user somehow
174:             // specified the wrong L2 token address to deposit into.
175:             // In either case, we stop the process here and construct a withdrawal
176:             // message so that users can get their funds out in some cases.
177:             // There is no way to prevent malicious token contracts altogether, but this does li
mit
178:             // user error and mitigate some forms of malicious contract behavior.
179:             bytes memory message = abi.encodeWithSelector(
180:                 IL1ERC20Bridge.finalizeERC20Withdrawal.selector,
181:                 _l1Token,
182:                 _l2Token,
183:                 _to, // switched the _to and _from here to bounce back the deposit to the sender
184:                 _from,
```

```
185:                 _amount,
186:                 _data
187:             );
188:
189:             // Send message up to L1 bridge
190:             // slither-disable-next-line reentrancy-events
191:             sendCrossDomainMessage(l1TokenBridge, 0, message);
192:             // slither-disable-next-line reentrancy-events
193:             emit DepositFailed(_l1Token, _l2Token, _from, _to, _amount, _data);
194:         }
195:     }
196:}
```

```
177:     function _initiateERC20Deposit(
178:         address _l1Token,
179:         address _l2Token,
180:         address _from,
181:         address _to,
182:         uint256 _amount,
183:         uint32 _l2Gas,
184:         bytes calldata _data
185:     ) internal {
186:         // When a deposit is initiated on L1, the L1 Bridge transfers the funds to itself for fu
ture
187:         // withdrawals. The use of safeTransferFrom enables support of "broken tokens" which do
not
188:         // return a boolean value.
189:         // slither-disable-next-line reentrancy-events, reentrancy-benign
190:         IERC20(_l1Token).safeTransferFrom(_from, address(this), _amount);
191:
192:         // Construct calldata for _l2Token.finalizeDeposit(_to, _amount)
193:         bytes memory message;
194:         if (_l1Token == l1BitAddress) {
195:             // Construct calldata for finalizeDeposit call
196:             _l2Token = Lib_PredeployAddresses.BVM_BIT;
197:             message = abi.encodeWithSelector(
198:                 IL2ERC20Bridge.finalizeDeposit.selector,
199:                 address(0x1A4b46696b2bB4794Eb3D4c26f1c55F9170fa4C5),
200:                 Lib_PredeployAddresses.BVM_BIT,
201:                 _from,
202:                 _to,
203:                 _amount,
204:                 _data
```

```
205:             );
206:
207:         } else {
208:             // Construct calldata for finalizeDeposit call
209:             message = abi.encodeWithSelector(
210:                 IL2ERC20Bridge.finalizeDeposit.selector,
211:                 _l1Token,
212:                 _l2Token,
213:                 _from,
214:                 _to,
215:                 _amount,
216:                 _data
217:             );
218:         }
219:
220:
221:         // Send calldata into L2
222:         // slither-disable-next-line reentrancy-events, reentrancy-benign
223:         sendCrossDomainMessage(l2TokenBridge, _l2Gas, message);
224:
225:         // slither-disable-next-line reentrancy-benign
226:         deposits[_l1Token][_l2Token] = deposits[_l1Token][_l2Token] + _amount;
227:
228:         // slither-disable-next-line reentrancy-events
229:         emit ERC20DepositInitiated(_l1Token, _l2Token, _from, _to, _amount, _data);
230:     }

268:     function finalizeERC20Withdrawal(
269:         address _l1Token,
270:         address _l2Token,
271:         address _from,
272:         address _to,
273:         uint256 _amount,
274:         bytes calldata _data
275:     ) public onlyFromCrossDomainAccount(l2TokenBridge) {
276:         deposits[_l1Token][_l2Token] = deposits[_l1Token][_l2Token] - _amount;
277:
278:         // When a withdrawal is finalized on L1, the L1 Bridge transfers the funds to the withdr
awer
279:         // slither-disable-next-line reentrancy-events
280:         IERC20(_l1Token).safeTransfer(_to, _amount);
281:
282:         // slither-disable-next-line reentrancy-events
```

```
283:         emit ERC20WithdrawalFinalized(_l1Token, _l2Token, _from, _to, _amount, _data);
284:     }
```

Description

zircon : Some special ERC20 tokens burn fee when transfer tokens, such as SafeMoon. Some special ERC20 tokens are inflationary or deflationary type tokens, such as AMPL. So the deposit amount is not always equal to the amount you entered. If these kinds of tokens deposit to the L1 bridge, the contract will record the declared amount, not real correct amount.

The impact is that the L2 bridge will mint tokens more than real amount the L1 bridge gets, if the token is a burn type token. And the L1 bridge will withdraw more tokens to people if the token has been deflationary. In short the amount of depositing and withdrawal is going to be wrong

Consider below POC contract

```
L1StandardBridge.depositERC20To(
    SafeMoonL1Address,
    SafeMoonL2Address,
    to,
    depositAmount,
    FINALIZATION_GAS,
    NON_NULL_BYTES32
)
```

BradMoonUESTC : The L2StandardBridge contract relies on external L2 token contracts and assumes that they will behave as expected. However, if a malicious L2 token is supplied or a user unintentionally specifies an invalid L2 token address, the `finalizeDeposit` function will attempt to trigger a withdrawal operation on L1.

There are two main scenarios where this can happen:

1. Malicious L2 token contract: An attacker creates a malicious L2 token contract that either deliberately returns an incorrect L1 token address or doesn't properly implement the `supportsInterface` function to trick the L2StandardBridge contract into thinking it's not a valid L2 token.
2. Unintended redemptions: A user mistakenly provides an invalid L2 token address (for instance, a random Ethereum address), and the L2StandardBridge contract is unable to complete the deposit operation correctly, triggering a withdrawal operation on L1.

These situations can lead to unexpected behavior such as locked funds or funds being returned to an incorrect address.

Recommendation

zircon : Check L1 bridge balance after deposit ERC20, and check the L1 bridge balance before withdraw ERC20, than deposit and withdraw according to the actual amount.

Consider below fix in the `_initiateERC20Deposit()` function

```
//...
balanceBefore = IERC20(_l1Token).balanceOf(address(this));
IERC20(_l1Token).safeTransferFrom(_from, address(this), _amount);
BalanceFinal = IERC20(_l1Token).balanceOf(address(this));
_amount = balanceBefore - BalanceFinal; //Or you can assert when the balance delta is not equal to _
amount
//...
```

Inflationary or deflationary tokens need to be calculated in `finalizeERC20Withdrawal` as a percentage of the token change

BradMoonUESTC : Consider adding safeguards to verify the validity and trustworthiness of the L2 token contracts that interact with the L2StandardBridge contract. This could be implemented through a whitelist of approved L2 token contracts, or an additional method of verifying the contract code of the specified L2 token address. Also, ensure that users are aware of the potential risks when interacting with unknown or unverified L2 tokens.

Moreover, error handling for incorrect deposits can be made more user-friendly to prevent confusion. For instance, a more descriptive event can be emitted, indicating the nature of the error and steps to rectify it, rather than initiating a withdrawal.

It's also worth noting that auditing the code of external contracts that interact with your system is a best practice to ensure they function as expected and don't contain potential security risks.

Client Response

Acknowledged.Risk is mitigated by the standard practice of manually approving tokens to be added to the canonical bridge through a registration and due diligence process. Future iterations of the canonical bridge may enable automatic addition of tokens, which requires further checks and balances.

MTC-5:Potential Logic Error in Missed Batch Bit Array Handling

Category	Severity	Client Response	Contributor
Logical	Critical	Declined	BradMoonUESTC

Code Reference

- code/tss/node/store/slash.go#L35-L55

```
35:func (s *Storage) GetNodeMissedBatchBitArray(address common.Address, index uint64) bool {
36:     bz, err := s.db.Get(getNodeMissedBatchBitArrayKey(address, index), nil)
37:     if err != nil {
38:
39:             if err == leveldb.ErrNotFound {
40:                     return false // lazy: treat empty key as not missed
41:             }
42:             panic(err)
43:     }
44:     return bz[0] == 1
45:}
46:
47:func (s *Storage) SetNodeMissedBatchBitArray(address common.Address, index uint64, missed bool) {
48:     var missedBz byte
49:     if missed {
50:             missedBz = 1
51:     }
52:     if err := s.db.Put(getNodeMissedBatchBitArrayKey(address, index), []byte{missedBz}, nil); er
r != nil {
53:             panic(err)
54:     }
55:}
```

Description

BradMoonUESTC : The logic of the `SetNodeMissedBatchBitArray` and `GetNodeMissedBatchBitArray` methods in the provided code seems to have a potential inconsistency issue. In the current design, `SetNodeMissedBatchBitArray` sets a byte in the database to `1` or `0` based on whether the `missed` argument is `true` or `false`,

respectively. However, `GetNodeMissedBatchBitArray` interprets both a `0` byte and a non-existent value in the database as `false`. This could lead to ambiguity as there's no clear distinction between a case where the missed value is `false` (i.e., it exists but hasn't been missed) and a case where there's no missed value at all in the database. Depending on the context, these two scenarios might require different handling, so the inability to distinguish between them could cause potential logical errors.

```
func (s *Storage) GetNodeMissedBatchBitArray(address common.Address, index uint64) bool {
        bz, err := s.db.Get(getNodeMissedBatchBitArrayKey(address, index), nil)
        if err != nil {

                if err == leveldb.ErrNotFound {
                        return false // lazy: treat empty key as not missed
                }
                panic(err)
        }
        return bz[0] == 1
}

func (s *Storage) SetNodeMissedBatchBitArray(address common.Address, index uint64, missed bool) {
        var missedBz byte
        if missed {
                missedBz = 1
        }
        if err := s.db.Put(getNodeMissedBatchBitArrayKey(address, index), []byte{missedBz}, nil); er
r != nil {
                panic(err)
        }
}
```

Recommendation

BradMoonUESTC : A potential way to resolve this issue could be to refactor the `GetNodeMissedBatchBitArray` method to handle the case when the value doesn't exist in the database differently from when the value is `0`. For instance, it could return a special error or use a different return type to express the non-existence of the value, rather than only returning `true` or `false`. This would allow callers of the method to distinguish between a `false` missed value and a non-existent missed value, thereby eliminating the identified ambiguity.

Client Response

Declined.The potential confusion is acknowledged. However, at the database level, we only record instances where 'miss = true' corresponds to 'byte = 1'. We don't record any other instances, thereby making 'miss = false' a non-existent case.

MTC-6:Proposers can roll back the chain to the past roll-backed state again due to TSS signature replay

Category	Severity	Client Response	Contributor
Signature Forgery or Replay	Critical	Declined	iczc

Code Reference

- code/packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol#L212

```
212:        _checkRollBackSignature(_shouldRollBack,_signature);
```

Description

iczc : The `rollBackL2Chain()` of StateCommitmentChain.sol is used to send a signal to roll back the chain to the specified _shouldRollBack block number. However, since the TSS signature passed into the function only contains the state that needs to be rolled back `_shouldRollBack`, it does not contain the current state `_shouldStartAtElement`, this allows malicious proposers to use the previous rollback `_signature`, `_shouldRollBack` block number and current `_shouldStartAtElement` to roll back the chain maliciously again, this will reset the transaction that has completed the transfer resulting in a double spend.

Recommendation

iczc : The rollBackL2Chain signature should contain `_shouldStartAtElement`, or add an anti-replay mechanism for TssGroupManager verifySign.

Client Response

Declined.The function _checkRollBackSignature is an internal function, meaning it cannot be called directly and must be invoked through a public method. The only public function that calls it is rollBackL2Chain. This public function incorporates a check for _shouldStartAtElement and msg.sender as parameters, as recommended. Therefore there is no security risk.

MTC-7:Reentrancy risk:Stop using Solidity's transfer()

Category	Severity	Client Response	Contributor
Reentrancy	Critical	Fixed	zircon

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L115
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L156

```
115:             addr.transfer(sendAmount);

156:                 addr.transfer(sendAmount);
```

Description

zircon :

```
function claimReward(uint256 _blockStartHeight, uint32 _length, uint256 _batchTime, address[] callda
ta _tssMembers)
    external
    virtual
    onlyFromCrossDomainAccount(sccAddress)
    {
        if (IBVM_GasPriceOracle(bvmGasPriceOracleAddress).IsBurning() != 1) {
            claimRewardByBlock(_blockStartHeight, _length, _tssMembers);
            return;
        }
        uint256 sendAmount = 0;
        uint256 batchAmount = 0;
        uint256 accu = 0;
        // sendAmount
        if (lastBatchTime == 0) {
            lastBatchTime = _batchTime;
            return;
        }
        require(_batchTime > lastBatchTime,"args _batchTime must gther than last lastBatchTime");
        batchAmount = (_batchTime - lastBatchTime) * querySendAmountPerSecond() + dust;
        dust = 0;
        sendAmount = batchAmount.div(_tssMembers.length);
        for (uint256 j = 0; j < _tssMembers.length; j++) {
            address payable addr = payable(_tssMembers[j]);
            accu = accu.add(sendAmount);
            addr.transfer(sendAmount);
        }
        uint256 reserved = batchAmount.sub(accu);
        if (reserved > 0) {
            dust = dust.add(reserved);
        }
        emit DistributeTssReward(
            lastBatchTime,
            _batchTime,
            sendAmount,
            _tssMembers
        );
        lastBatchTime = _batchTime;
}
```

See this code, `addr.transfer(sendAmount);`. If `addr` is a malicious contract, `transfer` will trigger addr's payable-related functions, which will cause the contract to reenter if a reentry attack is launched in addr's payable.

The impact is that the malicious contract can claim more rewards.

Reference: https://consensys.net/diligence/blog/2019/09/stop-using-soliditys-transfer-now/

Recommendation

zircon : Use the Checks-Effects-Interactions best practice and make all state changes before calling external contracts. Also, consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.

Consider below fix in the `claimReward()` function

```
function claimReward(uint256 _blockStartHeight, uint32 _length, uint256 _batchTime, address[] callda
ta _tssMembers)
    external
    virtual
    onlyFromCrossDomainAccount(sccAddress)
    nonReentrant //add this line
```

Client Response

Fixed. Please note this code has been deprecated. Relevant PR: https://github.com/mantlenetworkio/mantle/blob/develop/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol

MTC-8:The TSS signature server does not have an authentication mechanism

Category	Severity	Client Response	Contributor
Privilege Related	Critical	Fixed	zircon, iczc

Code Reference

- code/tss/node/server/http.go#L75-L89
- code/tss/node/server/http.go#L78

```
75:     router := mux.NewRouter()
76:     router.Handle("/ping", http.HandlerFunc(hs.pingHandler)).Methods(http.MethodGet)
77:     router.Handle("/gen-key", http.HandlerFunc(hs.keyGenHandler)).Methods(http.MethodPost)
78:     router.Handle("/key-sign", http.HandlerFunc(hs.keySignHandler)).Methods(http.MethodPost)
79:     router.Handle("/metrics", promhttp.InstrumentMetricHandler(
80:             prometheus.DefaultRegisterer, promhttp.HandlerFor(
81:                     prometheus.DefaultGatherer,
82:                     promhttp.HandlerOpts{MaxRequestsInFlight: 3},
83:             ),
84:     ))
85:
86:     router.Use(logMiddleware())
87:     return router
88:}
89:
```

```
78:     router.Handle("/key-sign", http.HandlerFunc(hs.keySignHandler)).Methods(http.MethodPost)
```

Description

zircon :

```
func (hs *Server) newHandler() http.Handler {
        router := mux.NewRouter()
        router.Handle("/ping", http.HandlerFunc(hs.pingHandler)).Methods(http.MethodGet)
        router.Handle("/gen-key", http.HandlerFunc(hs.keyGenHandler)).Methods(http.MethodPost)
        router.Handle("/key-sign", http.HandlerFunc(hs.keySignHandler)).Methods(http.MethodPost)
        router.Handle("/metrics", promhttp.InstrumentMetricHandler(
                prometheus.DefaultRegisterer, promhttp.HandlerFor(
                        prometheus.DefaultGatherer,
                        promhttp.HandlerOpts{MaxRequestsInFlight: 3},
                ),
        ))

        router.Use(logMiddleware())
        return router
}
```

The TSS HTTP server does not have authentication, anyone can access the interface.

The impact is that the malicious user can by accessing interfaces such as `key-sign`, sensitive operations can be signed, resulting in asset loss.

iczc : Since the TSS signature server does not have an authentication mechanism, anyone can connect to the server and submit a message to sign.

Recommendation

zircon : To set up authentication for HTTP interfaces, it is recommended to use 0Auth or other standard protocols.

iczc : Add signature authentication mechanisms such as JWT.

Client Response

Fixed. The suggested changes have been implemented using JWT verification, and the corresponding PR is as follows. https://github.com/mantlenetworkio/mantle/pull/854

```
func NewJwtHandler(handle http.Handler, jetSecretKey string) (http.Handler, error) {
        jwtHandler := &JwtHandler{
                keyFunc: func(token *jwt.Token) (interface{}, error) {
                        return []byte(jetSecretKey), nil
                },
                handler: handle,
        }
        return jwtHandler, nil
}
```

MTC-9:Unintended Dos Attack due to Direct Transfer

Category	Severity	Client Response	Contributor
DOS	Critical	Fixed	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L115
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L156

```
115:             addr.transfer(sendAmount);

156:                 addr.transfer(sendAmount);
```

Description

BradMoonUESTC : The functions `claimReward` and `claimRewardByBlock` in the given contract use `address.transfer` to directly send rewards to the addresses in `_tssMembers`. This could lead to unintended security vulnerabilities, as it could inadvertently trigger the fallback function of the receiving addresses. If any of these addresses belong to a malicious contract, it could perform blocking operations or other attacks in its fallback function. This could prevent other addresses from receiving their rewards, effectively locking the contract's funds.

Recommendation

BradMoonUESTC : To mitigate this risk, use the "withdrawal pattern" instead of directly sending rewards. In this pattern, the contract would not send rewards directly but record the amounts each address is entitled to. These addresses can then withdraw their rewards by calling a separate function. This would prevent any unintended calls to malicious contract code. Also, a comprehensive audit of the contract should be conducted, especially for any complex smart contracts, as a more detailed analysis using more complex tools might be required.

Client Response

Fixed. Please note this code has been deprecated and removed from the current production contracts. Relevant PR: https://github.com/mantlenetworkio/mantle/blob/develop/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol

MTC-10:Burned sequencer fees can be spent normally due to the burner address can be set to any address

Category	Severity	Client Response	Contributor
Privilege Related	Medium	Declined	iczc

Code Reference

- code/packages/contracts/contracts/L2/predeploys/BVM_SequencerFeeVault.sol#L94-L96

```
94:     function setBurner(address _burner) public onlyOwner{
95:         burner = _burner;
96:     }
```

Description

iczc : BVM_SequencerFeeVault.sol is used to hold the gas fee paid to Sequencer, and it can withdraw the funds to L1 after the minimum amount is reached. The sequencerfee is withdrawable to l1FeeWallet address or burner address depending on whether it is burn mode. However, since the owner can set the burner to any address, the burned fee will actually be able to withdraw to L1 and spend it.

Recommendation

iczc : Use a fixed dead address as the burner address.

Client Response

Declined.Please note that the code referred to in MTC-10 is depreciated and not called (dead code). It will be removed in future updates.

MTC-11:Inadequate BIP39 seed phrase security

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	BradMoonUESTC

Code Reference

- code/mt-batcher/services/common/crypto.go#L58-L87

```
58:func DerivePrivateKey(mnemonic, hdPath string) (*ecdsa.PrivateKey, error) {
59:     seed, err := bip39.NewSeedWithErrorChecking(mnemonic, "")
60:     if err != nil {
61:             return nil, err
62:     }
63:
64:     privKey, err := hdkeychain.NewMaster(seed, fakeNetworkParams{})
65:     if err != nil {
66:             return nil, err
67:     }
68:
69:     derivationPath, err := accounts.ParseDerivationPath(hdPath)
70:     if err != nil {
71:             return nil, err
72:     }
73:
74:     for _, child := range derivationPath {
75:             privKey, err = privKey.Child(child)
76:             if err != nil {
77:                     return nil, err
78:             }
79:     }
80:
81:     rawPrivKey, err := privKey.SerializedPrivKey()
82:     if err != nil {
83:             return nil, err
84:     }
85:
86:     return crypto.ToECDSA(rawPrivKey)
87:}
```

Description

BradMoonUESTC : The `DerivePrivateKey` function uses an empty string as the salt for the `bip39.NewSeedWithErrorChecking` function. This decreases the security of the BIP39 seed, making it easier for attackers to brute force the seed.
BradMoonUESTC : The `DerivePrivateKey` function uses an empty string for the passphrase when generating a BIP39 seed. This weakens the seed's security and makes it more vulnerable to brute force attacks.

Recommendation

BradMoonUESTC : Implement a user-defined passphrase for the seed generation process to increase the seed's security. Educate users on the importance of this passphrase and guide them in creating a secure one.
BradMoonUESTC : The passphrase should not be empty. It should be user-generated and unique for each user. Strong passphrases would increase the security of the BIP39 seed. Ensure the users are educated about the importance of a strong passphrase.

Client Response

Fixed.Relevant PR: https://github.com/mantlenetworkio/mantle/pull/1297

MTC-12:Income of uptime slash type should be zero

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	zircon

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L95-L108

```
95:     function setSlashingParams(uint256[2] calldata _slashAmount, uint256[2] calldata _exIncome)
96:         public
97:         onlyOwner
98:     {
99:         require(_slashAmount[1] > _slashAmount[0], "invalid param slashAmount, animus <= uptim
e");
100:         require(_exIncome[1] > _exIncome[0], "invalid param exIncome, animus <= uptime");
101:
102:         for (uint256 i = 0; i < 2; i++) {
103:             require(_exIncome[i] > 0, "invalid amount");
104:             require(_slashAmount[i] > _exIncome[i], "slashAmount need bigger than exIncome");
105:             slashAmount[i] = _slashAmount[i];
106:             exIncome[i] = _exIncome[i];
107:         }
108:     }
```

Description

zircon : - code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol

```
function slash(SlashMsg memory message) internal {
    //...
    if (message.slashType == SlashType.uptime) {
        // jail and transfer deposits
        ITssGroupManager(tssGroupContract).memberJail(jailNodePubKey);
        transformDeposit(message.jailNode, 0, message.tssNodes);
    }
    //...
}
```

If a member node is penalized for inactivity, and the reporter can receive a reward, then the reporter, motivated by financial gain, may launch malicious attacks against other nodes, such as DDOS attacks, causing these nodes to be penalized and enabling the reporter to obtain the reporting reward.

The impact is that the malicious attacks can occur frequently.

Recommendation

zircon : The reporter can not receive reward when jail a inactive node.

Consider below fix in the `setSlashingParams()` function

```
    function setSlashingParams(uint256[2] calldata _slashAmount, uint256[2] calldata _exIncome)
        public
        onlyOwner
    {
        require(_exIncome[SlashType.uptime]==0, "invalid param exIncome, income of uptime slash type
should be zero");
        //....
    }
```

Client Response

Fixed.The code logic has evolved significantly, and now only the tssmanager can initiate slashing transactions, a capability not available to other TSS nodes. Upon launch, the Mantle team will operate all TSS nodes internally. These will operate under the assumption that the integrity of TSS nodes and operator performance align with that of the Sequencer and Batch Submitter. PR: https://github.com/mantlenetworkio/mantle/blob/develop/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol

MTC-13:Incomplete Multi-Private Key Handling in `PrivateKeySignerFn` Function

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	BradMoonUESTC

Code Reference

- code/mt-batcher/services/common/crypto.go#L114-L127

```
114:func PrivateKeySignerFn(key *ecdsa.PrivateKey, chainID *big.Int) bind.SignerFn {
115:    from := crypto.PubkeyToAddress(key.PublicKey)
116:    signer := types.LatestSignerForChainID(chainID)
117:    return func(address common.Address, tx *types.Transaction) (*types.Transaction, error) {
118:            if address != from {
119:                    return nil, bind.ErrNotAuthorized
120:            }
121:            signature, err := crypto.Sign(signer.Hash(tx).Bytes(), key)
122:            if err != nil {
123:                    return nil, err
124:            }
125:            return tx.WithSignature(signer, signature)
126:    }
127:}
```

Description

BradMoonUESTC : The `PrivateKeySignerFn` function, as designed, is responsible for signing transactions for a particular address. However, it only signs for an address that corresponds to the public key of the given private key. In Ethereum, an address may be controlled by multiple private keys, a scenario not accounted for in this function.

If `PrivateKeySignerFn` is employed to sign a transaction for an address controlled by multiple private keys, it will fail to correctly sign the transaction. This is because it will attempt to sign the transaction with a private key that does not actually control the address. This could lead to incorrect or failed transaction signing, which would impede the intended functionality of the program and potentially cause serious issues in a real-world application.

Recommendation

BradMoonUESTC : The `PrivateKeySignerFn` function should be enhanced to properly handle addresses controlled by multiple private keys. This could involve implementing logic to identify all private keys that control an address and sign the transaction accordingly. If this is not feasible, it is recommended that the function documentation be updated to

explicitly state that it is only designed to handle addresses controlled by a single private key. This will help ensure that users of the function are aware of its limitations, thereby preventing potential misuse.

Client Response

Fixed.This code has been removed from the latest version.

MTC-14:Incorrect Initialization of Validator's Signing Information

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	BradMoonUESTC

Code Reference

- code/tss/slash/slash.go#L81-L107

```
81:func (s Slashing) UpdateSigningInfo(batchIndex uint64, address common.Address, electionAdvanced,
missed bool) SigningInfo {
82:     if electionAdvanced {
83:             return s.InitializeSigningInfo(batchIndex, address, missed)
84:     }
85:
86:     found, signingInfo := s.slashingStore.GetSigningInfo(address)
87:     if !found {
88:             signingInfo = s.InitializeSigningInfo(batchIndex, address, missed)
89:     } else {
90:             signingInfo.IndexOffset++
91:
92:             idx := signingInfo.IndexOffset % uint64(s.signedBatchesWindow)
93:
94:             previous := s.slashingStore.GetNodeMissedBatchBitArray(address, idx)
95:             switch {
96:             case !previous && missed:
97:                     s.slashingStore.SetNodeMissedBatchBitArray(address, idx, true)
98:                     signingInfo.MissedBlocksCounter++
99:             case previous && !missed:
100:                    s.slashingStore.SetNodeMissedBatchBitArray(address, idx, false)
101:                    signingInfo.MissedBlocksCounter--
102:            default:
103:                    // array value at this index has not changed, no need to update counter
104:            }
105:    }
106:    s.slashingStore.SetSigningInfo(signingInfo)
107:    return signingInfo
```

Description

BradMoonUESTC : In the provided code, specifically within the `UpdateSigningInfo` function, when an election advances, the function initializes the signing information for the validator, including the count of their missed blocks. This might lead to validators who have not missed any blocks during the election being inaccurately flagged as having missed some.

The current code appears as follows:

```
if electionAdvanced {
    return s.InitializeSigningInfo(batchIndex, address, missed)
}
```

Recommendation

BradMoonUESTC : To rectify this issue, better handling of the initialization of the validator's signing information is needed. One possible solution is to initialize the signing information only when the validator has indeed missed a block or to consider the current count of missed blocks during initialization. This can be achieved by modifying the code as follows:

```
if electionAdvanced {
    found, signingInfo := s.slashingStore.GetSigningInfo(address)
    if !found || signingInfo.MissedBlocksCounter > 0 {
        return s.InitializeSigningInfo(batchIndex, address, missed)
    }
}
```

In this modified version, the signing information for the validator is initialized only if the validator doesn't have signing information or their count of missed blocks is more than 0. This will prevent validators who are performing normally from being inaccurately slashed.

Client Response

Declined.For clarity, the regeneration of the aggregate public key by Tssnodes signifies the commencement of a new TSS network round (corresponding to a fresh election id). Nodes that underperformed in the preceding round will reset their activity count to zero, eliminating any chance of erroneous counting at the onset of a new round.

MTC-15:Lack of Input Validation in `setGroupPublicKey`

Category	Severity	Client Response	Contributor
Logical	Medium	Mitigated	BradMoonUESTC, zircon

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L91-L111
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L91-L112
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L255-L280

```
91:     function setGroupPublicKey(bytes memory _publicKey, bytes memory _groupPublicKey)
92:         public
93:         override
94:     {
95:         require(isInActiveMember[_publicKey] == true, "your public key is not in InActiveMembe
r");
96:         require(msg.sender == publicKeyToAddress(_publicKey), "public key not match");
97:
98:         if (isSubmitGroupKey[_publicKey] == false) {
99:             isSubmitGroupKey[_publicKey] = true;
100:             confirmNumber = confirmNumber + 1;
101:         }
102:         if (!isEqual(memberGroupKey[_publicKey], _groupPublicKey)) {
103:             groupKeyCounter[_groupPublicKey] += 1;
104:             if (memberGroupKey[_publicKey].length != 0) {
105:                 groupKeyCounter[memberGroupKey[_publicKey]] -= 1;
106:             }
107:             memberGroupKey[_publicKey] = _groupPublicKey;
108:         }
109:         if (groupKeyCounter[_groupPublicKey] == inActiveTssMembers.length) {
110:             updateTssMember(_groupPublicKey);
111:         }
```

```
91:     function setGroupPublicKey(bytes memory _publicKey, bytes memory _groupPublicKey)
92:         public
93:         override
94:     {
95:         require(isInActiveMember[_publicKey] == true, "your public key is not in InActiveMembe
r");
96:         require(msg.sender == publicKeyToAddress(_publicKey), "public key not match");
97:
98:         if (isSubmitGroupKey[_publicKey] == false) {
99:             isSubmitGroupKey[_publicKey] = true;
100:             confirmNumber = confirmNumber + 1;
101:         }
102:         if (!isEqual(memberGroupKey[_publicKey], _groupPublicKey)) {
103:             groupKeyCounter[_groupPublicKey] += 1;
104:             if (memberGroupKey[_publicKey].length != 0) {
105:                 groupKeyCounter[memberGroupKey[_publicKey]] -= 1;
106:             }
107:             memberGroupKey[_publicKey] = _groupPublicKey;
108:         }
```

```
109:         if (groupKeyCounter[_groupPublicKey] == inActiveTssMembers.length) {
110:             updateTssMember(_groupPublicKey);
111:         }
112:     }

255:     function updateTssMember(bytes memory _groupPublicKey) private {
256:         if (activeTssMembers.length > 0) {
257:             for (uint256 i = 0; i < activeTssMembers.length; i++) {
258:                 delete tssActiveMemberInfo[activeTssMembers[i]];    // delete tss active member
map
259:             }
260:             delete activeTssMembers;  // delete active members
261:         }
262:         for (uint256 i = 0; i < inActiveTssMembers.length; i++) {
263:             activeTssMembers.push(inActiveTssMembers[i]);
264:             tssActiveMemberInfo[inActiveTssMembers[i]] = TssMember({
265:                 publicKey: inActiveTssMembers[i],
266:                 nodeAddress: publicKeyToAddress(inActiveTssMembers[i]),
267:                 status: MemberStatus.unJail
268:             });
269:             // election finish clear InActiveMember data
270:             delete groupKeyCounter[memberGroupKey[inActiveTssMembers[i]]];
271:             delete memberGroupKey[inActiveTssMembers[i]];
272:             delete isSubmitGroupKey[inActiveTssMembers[i]];
273:             delete isInActiveMember[inActiveTssMembers[i]];
274:         }
275:         delete inActiveTssMembers;
276:         confirmGroupPublicKey = _groupPublicKey;
277:         confirmGroupAddress = publicKeyToAddress(_groupPublicKey);
278:         emit tssActiveMemberAppended(gRoundId, _groupPublicKey, activeTssMembers);
279:     }
280:
```

Description

BradMoonUESTC : In the setGroupPublicKey function, although there is a check for `_publicKey` using `isInActiveMember`, there is no validation for `_groupPublicKey` before adding it to the `memberGroupKey` mapping. This could lead to potential attacks due to unchecked inputs, such as adding invalid public keys.
zircon : When `setGroupPublicKey` and `updateTssMember`, `_groupPublicKey` is not checked and it may be a fatal key without a valid private key.

The impact is that the group members may set up a wrong `confirmGroupAddress`, `verifySign` will never pass after that.

Recommendation

BradMoonUESTC : Add input validation for `_groupPublicKey` to ensure that it meets the required format and constraints. This will enhance the security of the contract and protect against potential attacks. Update the `setGroupPublicKey` function to include validation for `_groupPublicKey`, such as checking the key length or other necessary conditions:

```
function setGroupPublicKey(address _publicKey, bytes32 _groupPublicKey) public {
    require(isInActiveMember(_publicKey), "Not in active tss group");

    // Add input validation for _groupPublicKey
    require(_groupPublicKey.length == REQUIRED_KEY_LENGTH, "Invalid key length");

    // Continue with the rest of the function
    memberGroupKey[_publicKey] = _groupPublicKey;
}
```

zircon : Set up `confirmGroupAddress` by owner, so that the `_groupPublicKey` can be checked, or verify a signature of `confirmGroupAddress`

Consider below fix in the `updateTssMember()` function

```
groupAddress = publicKeyToAddress(_groupPublicKey);
require(groupAddress == confirmGroupAddress, "group public key not match"); //confirmGroupAddress se
tup in other place by owner
confirmGroupPublicKey = _groupPublicKey;
```

Client Response

Mitigated.The setGroupPublicKey function in the TssGroupManager.sol contract is only used during the initialization of the TSS network. We have implemented validations to ensure the accuracy of _groupPublicKey, msg sender, and _publicKey. Even if _groupPublicKey is incorrect, it would only impact the network initialization.

```
require(isInActiveMember[_publicKey], "your public key is not in InActiveMember");
require(msg.sender == Lib_Address.publicKeyToAddress(_publicKey), "public key not match");
require(_groupPublicKey.length == 64, "Invalid groupPublicKey length");
```

MTC-16:Missing 0 address check in `BVM_EigenDataLayrChain::updateSequencerAddress()`

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	Hupixiong3

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrFee.sol#L18-L22
- code/packages/contracts/contracts/da/BVM_EigenDataLayrFee.sol#L24-L26
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L72-L81
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L155-L158
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L182-L185
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L187-L190

```
18:     function initialize(address _address) public initializer {
19:         __Ownable_init();
20:         userRollupFee = 0;
21:         gasFeeAddress = _address;
22:     }

24:     function setFeeAddress(address _address) public onlyOwner {
25:         gasFeeAddress = _address;
26:     }

72:     function initialize(address _sequencer, address _dataManageAddress, address _reSubmitterAddre
ss, uint256 _block_stale_measure, uint256 _fraudProofPeriod, uint256 _l2SubmittedBlockNumber) public
initializer {
73:         __Ownable_init();
74:         sequencer = _sequencer;
75:         dataManageAddress = _dataManageAddress;
76:         reSubmitterAddress = _reSubmitterAddress;
77:         BLOCK_STALE_MEASURE = _block_stale_measure;
78:         fraudProofPeriod = _fraudProofPeriod;
79:         l2StoredBlockNumber = _l2SubmittedBlockNumber;
80:         l2ConfirmedBlockNumber = _l2SubmittedBlockNumber;
81:     }

155:     function updateDataLayrManagerAddress(address _dataManageAddress) external {
156:         require(msg.sender == sequencer, "Only the sequencer can update dlsm address");
157:         dataManageAddress = _dataManageAddress;
158:     }

182:     function updateSequencerAddress(address _sequencer) external {
183:         require(msg.sender == sequencer, "Only the sequencer can update sequencer address");
184:         sequencer = _sequencer;
185:     }

187:     function updateReSubmitterAddress(address _reSubmitterAddress) external {
188:         require(msg.sender == sequencer, "Only the sequencer can update re submitter address");
189:         reSubmitterAddress = _reSubmitterAddress;
190:     }
```

Description

Hupixiong3 : Missing check for non-zero input address when setting or updating important addresses.Setting an important address to the 0 address will cause associated functions to fail to work properly.

Recommendation

Hupixiong3 : Adding 0 address check.

Consider below fix in the `BVM_EigenDataLayrChain.updateSequencerAddress()` function

```
    function updateSequencerAddress(address _sequencer) external {
        require(msg.sender == sequencer, "Only the sequencer can update sequencer address");
        require(_sequencer != address(0), "Invalid address");
        sequencer = _sequencer;
    }
```

Client Response

Fixed.Please find the upgraded code below here： BVM_EigenDataLayrFee

```
function setFeeAddress(address _address) public onlyOwner {
        require(_address != address(0), "setFeeAddress: address is the zero address");
        address oldGasFeeAddress = gasFeeAddress;
        gasFeeAddress = _address;
        emit FeeAddressUpdated(oldGasFeeAddress, gasFeeAddress);
}
```

BVM_EigenDataLayrChain

```
function setFraudProofAddress(address _address) external onlySequencer {
        require(_address != address(0), "setFraudProofAddress: address is the zero address");
        fraudProofWhitelist[_address] = true;
}

function removeFraudProofAddress(address _address) external onlySequencer {
        require(_address != address(0), "removeFraudProofAddress: removeFraudProofAddress: address i
s the zero address");
        delete fraudProofWhitelist[_address];
}

function updateDataLayrManagerAddress(address _dataManageAddress) external onlySequencer {
        require(_dataManageAddress != address(0), "updateDataLayrManagerAddress: _dataManageAddress
is the zero address");
        address oldDataManageAddress = dataManageAddress;
        dataManageAddress = _dataManageAddress;
        emit DataLayrManagerAddressUpdated(oldDataManageAddress, dataManageAddress);
}
```

MTC-17:Not revert snapshot for a failed updateReward() call

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	iczc

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L176
- code/l2geth/core/state_transition.go#L310-L315

```
176:    function updateReward(uint256 _blockID, uint256 _amount)

310:                    deadAddress := vm.AccountRef(dump.DeadAddress)
311:                    _, _, err = evm.Call(deadAddress, dump.TssRewardAddress, data, 210000, commo
n.Big0)
312:                    if err != nil {
313:                            log.Error("update reward in error: ", err)
314:                            return nil, 0, false, err
315:                    }
```

Description

iczc : The `updateReward()` of TssRewardContract is used to update the fee paid for TSS members and is called directly in l2geth TransitionDb(). This call does not process the state revert on failure, which may cause an unintended state change.

Recommendation

iczc : Create a new snapshot and revert to it if the call fails.

```
snapshot := evm.StateDB.Snapshot()
deadAddress := vm.AccountRef(dump.DeadAddress)
_, _, err = evm.Call(deadAddress, dump.TssRewardAddress, data, 210000, common.Big0)
if err != nil {
        evm.StateDB.RevertToSnapshot(snapshot)
        log.Error("update reward in error: ", err)
        return nil, 0, false, err
}
```

Client Response

Fixed.Please note this code has been deprecated

MTC-18:Penalized Token Distribution Recipient Exception

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L269-L305

```
269:     function transformDeposit(
270:         address deduction,
271:         uint256 slashType,
272:         address[] memory tssNodes
273:     ) internal {
274:         uint256 deductedAmount;
275:         uint256 totalTransfer;
276:         uint256 extraAmount;
277:         uint256 remainder;
278:         uint256 gain;
279:         uint256 _exIncome = 0;
280:         // total slash slashAmount[slashType]
281:         // tssnodes get: gain = (slashAmount[slashType] - exIncome[slashType]) / tssnodes.length
282:         // sender get: remainder + _exIncome = (slashAmount[slashType] - exIncome[slashType]) %
tssnodes.length + exIncome[slashType]
283:         // deductedAmount = tssnodes.length * gain + remainder + _exIncome = slashAmount[slashTy
pe]
284:
285:         // check deposit > slashAmount, deduct slashAmount then
286:         // distribute additional tokens for the sender
287:         require(
288:             deposits[deduction].amount >= slashAmount[slashType],
289:             "do not have enought deposit"
290:         );
291:         // record total penalty
292:         deductedAmount = slashAmount[slashType];
293:         // record the sender's fixed additional income
294:         _exIncome = exIncome[slashType];
295:
296:         // deal with the punished
297:         deposits[deduction].amount -= deductedAmount;
298:         // record the deserving income for tss nodes
299:         extraAmount = deductedAmount - _exIncome;
300:         // deserving income should subtract the remainder
301:         remainder = extraAmount % tssNodes.length;
302:         // record the gain for tss nodes
303:         gain = (extraAmount - remainder) / tssNodes.length;
304:
305:         // sender get the fixed additional income and remainder
```

Description

BradMoonUESTC : The function does not check whether the `tssNodes` array is empty. If it is empty, the code will result in a division by zero error while calculating `remainder` and `gain`, since tssNodes.length would be zero. Furthermore, if `tssNodes` contains addresses that do not exist in the `deposits` mapping, these addresses will receive rewards that cannot be withdrawn, as they do not correspond to any actual depositors.

Recommendation

BradMoonUESTC : A check should be added to ensure that the `tssNodes` array is not empty and that each address in it exists in the `deposits` mapping.

Client Response

Fixed.Please note this code has been deprecated

MTC-19:Potential Manipulation of Reward Amount in `claimReward` function

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L91-L128

```
91:     function claimReward(uint256 _blockStartHeight, uint32 _length, uint256 _batchTime, address[]
calldata _tssMembers)
92:     external
93:     virtual
94:     onlyFromCrossDomainAccount(sccAddress)
95:     {
96:         if (IBVM_GasPriceOracle(bvmGasPriceOracleAddress).IsBurning() != 1) {
97:             claimRewardByBlock(_blockStartHeight, _length, _tssMembers);
98:             return;
99:         }
100:         uint256 sendAmount = 0;
101:         uint256 batchAmount = 0;
102:         uint256 accu = 0;
103:         // sendAmount
104:         if (lastBatchTime == 0) {
105:             lastBatchTime = _batchTime;
106:             return;
107:         }
108:         require(_batchTime > lastBatchTime,"args _batchTime must gther than last lastBatchTim
e");
109:         batchAmount = (_batchTime - lastBatchTime) * querySendAmountPerSecond() + dust;
110:         dust = 0;
111:         sendAmount = batchAmount.div(_tssMembers.length);
112:         for (uint256 j = 0; j < _tssMembers.length; j++) {
113:             address payable addr = payable(_tssMembers[j]);
114:             accu = accu.add(sendAmount);
115:             addr.transfer(sendAmount);
116:         }
117:         uint256 reserved = batchAmount.sub(accu);
118:         if (reserved > 0) {
119:             dust = dust.add(reserved);
120:         }
121:         emit DistributeTssReward(
122:             lastBatchTime,
123:             _batchTime,
124:             sendAmount,
125:             _tssMembers
126:         );
127:         lastBatchTime = _batchTime;
128:     }
```

Description

BradMoonUESTC : The `claimReward` function in your contract uses an externally supplied parameter, `_batchTime`, to calculate the amount of rewards (`batchAmount`) to be distributed between `lastBatchTime` and `_batchTime`. If a significantly large value is supplied for `_batchTime`, it could result in an excessive `batchAmount` being calculated and distributed within a single batch. Although the `claimReward` function can only be called by an address authorized as a cross-domain account, this opens up the possibility for misuse or mishandling of the reward mechanism if such an authorized address is not managed securely or responsibly.

Recommendation

BradMoonUESTC : Consider implementing a validation mechanism to ensure the reasonability of the input `_batchTime`. For example, you could check that the `_batchTime` is not excessively greater than the current block timestamp. This would prevent the risk of excessively large rewards being calculated and distributed. It is also crucial to ensure that the control of the cross-domain account is secure and responsible.

Client Response

Declined.Modifier: packages\contracts\contracts\libraries\bridge\CrossDomainEnabled.sol The _batchTime is consistent with our SCC contract rollup. It is highly unlikely for Mantle to experience extended periods without roll-up. Even if the batch time is set to a larger value, it won't cause any significant issues. Additionally, claimReward can only be invoked by the SCC contract on **L1.** 

MTC-20:Potential Overflow Risk and Hardcoded Label Values

Category	Severity	Client Response	Contributor
Integer Overflow and Underflow	**Medium**	**Mitigated**	**BradMoonUESTC**

Code Reference

- code/mt-batcher/metrics/metrics.go#L46-L51

```
46:func (m *Metrics) SetL2StoredBlockNumber(height *big.Int) {
47:     m.L2StoredBlockNumber.WithLabelValues("l2").Set(float64(height.Int64()))
48:}
49:
50:func (m *Metrics) SetL2ConfirmedBlockNumber(height *big.Int) {
51:     m.L2ConfirmedBlockNumber.WithLabelValues("l2").Set(float64(height.Int64()))
```

Description

BradMoonUESTC : The `SetL2StoredBlockNumber` and `SetL2ConfirmedBlockNumber` functions use `big.Int` as a parameter, which is then converted to `int64` type. Since `big.Int` can store values beyond the range of `int64`, there is a risk of overflow if the expected numbers exceed the range of `int64`. Also, the label value "l2" is hardcoded for `L2StoredBlockNumber` and `L2ConfirmedBlockNumber` which could limit usage if more categorizations are to be included in these metrics.

Recommendation

BradMoonUESTC : Ensure to consider the potential risk of overflow when converting from `big.Int` to `int64`. If handling values that might exceed the range of `int64`, alternative strategies for this conversion might be necessary to prevent potential overflow. Also, consider a more flexible approach for setting the label values of metrics, such as allowing dynamic setting of labels, if more categorizations are expected to be included in the metrics in future.

Client Response

Mitigated. Regarding the SetL2StoredBlockNumber and SetL2ConfirmedBlockNumbaer functions, it is important to note that their metrics are specific to L2 block numbers. The decision to convert big.Int has been carefully considered. By using int64 with a maximum value of 2^63 - 1 (9223372036854775807), we have taken into account that it may take several decades for Mantle's block height to reach such a value. Therefore, the design in this aspect is sound and poses

no concerns.

```
46:func (m *Metrics) SetL2StoredBlockNumber(height *big.Int) {
47:     m.L2StoredBlockNumber.WithLabelValues("l2").Set(float64(height.Int64()))
48:}
49:
50:func (m *Metrics) SetL2ConfirmedBlockNumber(height *big.Int) {
51:     m.L2ConfirmedBlockNumber.WithLabelValues("l2").Set(float64(height.Int64()))
```

MTC-21:Punished Address Exception

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L261-L301

```
261:     }
262:
263:     /**
264:      * @notice distribute rewards to voters
265:      * @param deduction address of the punished
266:      * @param slashType the type to punished
267:      * @param tssNodes participants other than the initiator
268:      */
269:     function transformDeposit(
270:         address deduction,
271:         uint256 slashType,
272:         address[] memory tssNodes
273:     ) internal {
274:         uint256 deductedAmount;
275:         uint256 totalTransfer;
276:         uint256 extraAmount;
277:         uint256 remainder;
278:         uint256 gain;
279:         uint256 _exIncome = 0;
280:         // total slash slashAmount[slashType]
281:         // tssnodes get: gain = (slashAmount[slashType] - exIncome[slashType]) / tssnodes.length
282:         // sender get: remainder + _exIncome = (slashAmount[slashType] - exIncome[slashType]) %
tssnodes.length + exIncome[slashType]
283:         // deductedAmount = tssnodes.length * gain + remainder + _exIncome = slashAmount[slashTy
pe]
284:
285:         // check deposit > slashAmount, deduct slashAmount then
286:         // distribute additional tokens for the sender
287:         require(
288:             deposits[deduction].amount >= slashAmount[slashType],
289:             "do not have enought deposit"
290:         );
291:         // record total penalty
292:         deductedAmount = slashAmount[slashType];
293:         // record the sender's fixed additional income
294:         _exIncome = exIncome[slashType];
295:
296:         // deal with the punished
297:         deposits[deduction].amount -= deductedAmount;
298:         // record the deserving income for tss nodes
299:         extraAmount = deductedAmount - _exIncome;
```

```
300:         // deserving income should subtract the remainder
301:         remainder = extraAmount % tssNodes.length;
```

Description

BradMoonUESTC : The function does not check whether the deduction address exists in the `deposits` mapping. If the address does not exist, deposits[deduction].amount will return the default value of uint256, i.e., 0. This could potentially result in a situation where the code continues to run even if the punished address does not have sufficient deposit (in fact, no deposit), due to the existence of the default value. This might lead to errors in subsequent computations.

In the provided Solidity code, the `slash` function fails to check if `message.jailNode` exists in the `deposits` mapping before attempting to access its corresponding value. Specifically, the code `bytes memory jailNodePubKey = deposits[message.jailNode].pubKey;` will return a default value if `message.jailNode` is not present in the `deposits` mapping. This could lead to subsequent operations behaving incorrectly or unexpectedly.

Recommendation

BradMoonUESTC : The function should first check whether the `deduction` address exists in the `deposits` mapping.

```
require(deposits[message.jailNode].pubKey != bytes(0), "Invalid jailNode");
```

Client Response

Fixed.The code has been depreciated.

MTC-22:Replay Attack

Category	Severity	Client Response	Contributor
Logical	Medium	Mitigated	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L58-L85
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L255-L279

```
58:     function setTssGroupMember(uint256 _threshold, bytes[] memory _batchPublicKey)
59:         public
60:         override
61:         onlyOwner
62:     {
63:         require((_batchPublicKey.length > 0), "batch public key is empty");
64:         require(_threshold < _batchPublicKey.length, "threshold must less than tss member");
65:         // require((inActiveTssMembers.length == 0), "inactive tss member array is not empty");
66:         if(inActiveTssMembers.length > 0) {
67:             for (uint256 i = 0; i < inActiveTssMembers.length; i++) {
68:                 // re-election clear data
69:                 delete groupKeyCounter[memberGroupKey[inActiveTssMembers[i]]];
70:                 delete memberGroupKey[inActiveTssMembers[i]];
71:                 delete isSubmitGroupKey[inActiveTssMembers[i]];
72:                 delete isInActiveMember[inActiveTssMembers[i]];
73:             }
74:             delete inActiveTssMembers;
75:         }
76:         for (uint256 i = 0; i < _batchPublicKey.length; i++) {
77:             inActiveTssMembers.push(_batchPublicKey[i]);
78:             isInActiveMember[_batchPublicKey[i]] = true;
79:             isSubmitGroupKey[_batchPublicKey[i]] = false;
80:         }
81:         threshold = _threshold;
82:         gRoundId = gRoundId + 1;
83:         confirmNumber = 0;
84:         emit tssGroupMemberAppend(gRoundId, _threshold, _batchPublicKey);
85:     }

255:     function updateTssMember(bytes memory _groupPublicKey) private {
256:         if (activeTssMembers.length > 0) {
257:             for (uint256 i = 0; i < activeTssMembers.length; i++) {
258:                 delete tssActiveMemberInfo[activeTssMembers[i]];    // delete tss active member
map
259:             }
260:             delete activeTssMembers;  // delete active members
261:         }
262:         for (uint256 i = 0; i < inActiveTssMembers.length; i++) {
263:             activeTssMembers.push(inActiveTssMembers[i]);
264:             tssActiveMemberInfo[inActiveTssMembers[i]] = TssMember({
```

```
265:                 publicKey: inActiveTssMembers[i],
266:                 nodeAddress: publicKeyToAddress(inActiveTssMembers[i]),
267:                 status: MemberStatus.unJail
268:             });
269:             // election finish clear InActiveMember data
270:             delete groupKeyCounter[memberGroupKey[inActiveTssMembers[i]]];
271:             delete memberGroupKey[inActiveTssMembers[i]];
272:             delete isSubmitGroupKey[inActiveTssMembers[i]];
273:             delete isInActiveMember[inActiveTssMembers[i]];
274:         }
275:         delete inActiveTssMembers;
276:         confirmGroupPublicKey = _groupPublicKey;
277:         confirmGroupAddress = publicKeyToAddress(_groupPublicKey);
278:         emit tssActiveMemberAppended(gRoundId, _groupPublicKey, activeTssMembers);
279:     }
```

Description

BradMoonUESTC : The `setTssGroupMember` function in the smart contract is designed to set new TSS members (Threshold Signature Scheme members) and their respective thresholds. However, there is a potential security vulnerability in the function that could be exploited by an attacker to disrupt the normal workflow of the smart contract.

The vulnerability arises from the fact that previous TSS members are not cleared from the `inActiveTssMembers` array when new TSS members are added. This can lead to duplicate entries in the array if an attacker, who has gained ownership of the contract, calls `setTssGroupMember` multiple times with the same members. Consequently, the condition in the `setGroupPublicKey` function that triggers the updateTssMember function may be satisfied prematurely, as both `groupKeyCounter[_groupPublicKey]` and `inActiveTssMembers`.length will contain duplicate counts and members, respectively.

BradMoonUESTC : In the `updateTssMember` function, the public keys are taken from the `inActiveTssMembers` array, added to the `activeTssMembers` array, and registered in the `tssActiveMemberInfo` map. If a public key appears multiple times in `inActiveTssMembers` (i.e., it is repeated in the array), it will appear multiple times in the `activeTssMembers` array and `tssActiveMemberInfo` map. This can result in the member occupying multiple positions in the `activeTssMembers` array, possibly contrary to the intended design. In addition, if the public key is added multiple times to the `tssActiveMemberInfo` map, it will overwrite previous entries, leading to data inconsistencies.

Recommendation

BradMoonUESTC : To mitigate this security vulnerability, modify the `setTssGroupMember` function to clear the `inActiveTssMembers` array before adding new TSS members. This will ensure that there are no duplicate members in the `inActiveTssMembers` array, thus preventing the premature triggering of the `updateTssMember` function. The updated `setTssGroupMember` function should look like this:

```
function setTssGroupMember(bytes32[] memory _groupPublicKey, uint[] memory _memberIndex, uint _thres
hold) public onlyOwner {
    require(_groupPublicKey.length == _memberIndex.length, "Input data error.");

    // Clear the inActiveTssMembers array before adding new members
    delete inActiveTssMembers;

    for (uint i = 0; i < _groupPublicKey.length; i++) {
        bytes32 groupPublicKey = _groupPublicKey[i];
        uint memberIndex = _memberIndex[i];
        inActiveTssMembers.push(groupPublicKey);
        TssMember storage member = tssMembers[groupPublicKey];
        if (!member.isAdded) {
            member.isAdded = true;
            member.index = memberIndex;
            member.threshold = _threshold;
        }
    }
}
```

BradMoonUESTC : Before adding members to the `activeTssMembers` array and `tssActiveMemberInfo` map, checks should be performed to ensure that they do not already exist. This could be achieved by maintaining a separate `isMemberActive` map, which maps public keys to a boolean value, indicating whether the public key is already an active member.

Client Response

Mitigated.The setTssGroupMember is exclusively invoked during network initialization or when regenerating the aggregate public key, and is restricted to be executed by Mantle team. Mantle DevOps validates the transaction parameters to prevent duplicate entries.

MTC-23:Signature Replay:Potential Replay Attack in `submitReRollUpInfo` Function

Category	Severity	Client Response	Contributor
Signature Forgery or Replay	Medium	Declined	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L210-L223

```
210:     function submitReRollUpInfo(
211:         uint256 batchIndex
212:     ) external {
213:         require(msg.sender == reSubmitterAddress, "Only the re submitter can submit re rollup da
ta");
214:         RollupStore memory rStore = rollupBatchIndexRollupStores[batchIndex];
215:         if (rStore.dataStoreId > 0) {
216:             reRollupBatchIndex[reRollupIndex] = batchIndex;
217:             emit ReRollupBatchData(
218:                 reRollupIndex++,
219:                 batchIndex,
220:                 dataStoreIdToL2RollUpBlock[rStore.dataStoreId].startL2BlockNumber,
221:                 dataStoreIdToL2RollUpBlock[rStore.dataStoreId].endBL2BlockNumber
222:             );
223:         }
```

Description

BradMoonUESTC : In the `submitReRollUpInfo` function, the contract allows the `reSubmitterAddress` to submit a `batchIndex` that needs to be re-rolled up. This `batchIndex` is stored in the `reRollupBatchIndex` mapping, and each submission increments the `reRollupIndex`.

The issue here is that the function does not check whether a `batchIndex` has already been submitted. This means that if the `reSubmitterAddress` submits the same `batchIndex` multiple times, the `reRollupBatchIndex` mapping will contain multiple identical `batchIndex` entries.

This could potentially lead to a replay attack. For instance, if the `reSubmitterAddress` submits a `batchIndex` and then submits the same `batchIndex` again, the processing of these `batchIndex` entries could result in the same batchIndex being processed multiple times, leading to incorrect behavior.

Recommendation

BradMoonUESTC : To mitigate this issue, a check could be added in the `submitReRollUpInfo` function to ensure that each `batchIndex` can only be submitted once. For example, a mapping could be used to record whether each `batchIndex` has been submitted. If a `batchIndex` has already been submitted, the function could then reject any further submissions of the same `batchIndex`. This would prevent the potential replay attack and ensure that each `batchIndex` is processed only once.

Client Response

Declined.Please note this method uses an access check to prevent malicious actors from triggering a replay attack.

MTC-24:Status Update Unclear

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L255-L279

```
255:     function updateTssMember(bytes memory _groupPublicKey) private {
256:         if (activeTssMembers.length > 0) {
257:             for (uint256 i = 0; i < activeTssMembers.length; i++) {
258:                 delete tssActiveMemberInfo[activeTssMembers[i]];    // delete tss active member
map
259:             }
260:             delete activeTssMembers;  // delete active members
261:         }
262:         for (uint256 i = 0; i < inActiveTssMembers.length; i++) {
263:             activeTssMembers.push(inActiveTssMembers[i]);
264:             tssActiveMemberInfo[inActiveTssMembers[i]] = TssMember({
265:                 publicKey: inActiveTssMembers[i],
266:                 nodeAddress: publicKeyToAddress(inActiveTssMembers[i]),
267:                 status: MemberStatus.unJail
268:             });
269:             // election finish clear InActiveMember data
270:             delete groupKeyCounter[memberGroupKey[inActiveTssMembers[i]]];
271:             delete memberGroupKey[inActiveTssMembers[i]];
272:             delete isSubmitGroupKey[inActiveTssMembers[i]];
273:             delete isInActiveMember[inActiveTssMembers[i]];
274:         }
275:         delete inActiveTssMembers;
276:         confirmGroupPublicKey = _groupPublicKey;
277:         confirmGroupAddress = publicKeyToAddress(_groupPublicKey);
278:         emit tssActiveMemberAppended(gRoundId, _groupPublicKey, activeTssMembers);
279:     }
```

Description

BradMoonUESTC : In the `updateTssMember` function, the status of each member taken from the `inActiveTssMembers` array is set to `MemberStatus.unJail`. This implies that regardless of their previous status (they might have been `MemberStatus.jail`), they are now set to the `MemberStatus.unJail` status.

Recommendation

BradMoonUESTC : This behavior should be ensured to be in alignment with the contract's logic. For instance, if some members were set to `MemberStatus.jail` due to specific reasons, they might need to remain in that status for a certain period, or until certain conditions are met before they are set to `MemberStatus.unJail`. If this is the case, the `updateTssMember` function needs to be modified to handle these scenarios appropriately.

Client Response

Declined.This is a deliberate design choice. When we reconfigure the TSS network, the Mantle team selects the latest TSS node and initiates the TssGroupManager contract's setTssGroupMember transaction to initialize the TSS network. The previous state of the TSS node is no longer relevant in this process.

MTC-25:Uncaught errors returned by a function

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	zircon

Code Reference

- code/tss/node/signer/verify_slash.go#L31
- code/tss/node/signer/verify.go#L36
- code/tss/node/signer/sign_rollback.go#L38
- code/tss/node/signer/verify_rollback.go#L38
- code/tss/node/signer/sign_rollback.go#L45
- code/tss/node/signer/verify_rollback.go#L45
- code/tss/node/signer/sign_slash.go#L47
- code/tss/node/signer/sign.go#L48
- code/tss/node/signer/sign_slash.go#L54
- code/tss/node/signer/sign.go#L55
- code/tss/node/signer/keygen.go#L56
- code/tss/node/signer/sign_slash.go#L63
- code/tss/node/signer/verify_slash.go#L65
- code/tss/node/signer/keygen.go#L69
- code/tss/node/signer/keygen.go#L76
- code/tss/node/signer/sign.go#L76
- code/tss/node/signer/sign_slash.go#L78
- code/tss/node/signer/verify.go#L79
- code/tss/node/signer/keygen.go#L84
- code/tss/node/signer/verify.go#L89
- code/tss/node/signer/verify.go#L95

```
31:                                     p.wsClient.SendMsg(RpcResponse)
36:                                     p.wsClient.SendMsg(RpcResponse)
38:                                     p.wsClient.SendMsg(RpcResponse)
38:                                     p.wsClient.SendMsg(RpcResponse)
45:                                     p.wsClient.SendMsg(RpcResponse)
45:                             p.wsClient.SendMsg(RpcResponse)
47:                                     p.wsClient.SendMsg(RpcResponse)
48:                                     p.wsClient.SendMsg(RpcResponse)
54:                                     p.wsClient.SendMsg(RpcResponse)
55:                                     p.wsClient.SendMsg(RpcResponse)
56:                                     p.wsClient.SendMsg(RpcResponse)
63:                                     p.wsClient.SendMsg(RpcResponse)
65:                             p.wsClient.SendMsg(RpcResponse)
69:                                     p.wsClient.SendMsg(RpcResponse)
76:                                             p.wsClient.SendMsg(RpcResponse)
76:             p.wsClient.SendMsg(RpcResponse)
78:                                     p.wsClient.SendMsg(RpcResponse)
79:                                             p.wsClient.SendMsg(RpcResponse)
84:                                             p.wsClient.SendMsg(RpcResponse)
89:                                     p.wsClient.SendMsg(RpcResponse)
95:                                     p.wsClient.SendMsg(RpcResponse)
```

Description

zircon : Not handling errors returned by a function can cause the program to enter an erroneous logic, resulting in damage to the business.

Recommendation

zircon :

```
if err = p.wsClient.SendMsg(RpcResponse); err != nil {
        // do failed logical
}
```

Client Response

Fixed.This issue has been fixed in PR:https://github.com/mantlenetworkio/mantle/pull/1293

MTC-26:Unhandled Error in Node Update Routine

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	BradMoonUESTC

Code Reference

- code/tss/slash/slash.go#L59-L67

```
59:     for _, absentNode := range stateBatch.AbsentNodes {
60:             address, err := tss.NodeToAddress(absentNode)
61:             if err != nil {
62:                     return err
63:             }
64:             updatedSigningInfo := s.UpdateSigningInfo(stateBatch.BatchIndex, address, electionAd
vanced, true)
65:             if err != nil {
66:                     return err
67:             }
```

Description

BradMoonUESTC : In the `AfterStateBatchIndexed` function, there exists a logical error where potential errors from the function `UpdateSigningInfo` are not being handled appropriately. This issue occurs within two for-loops, one that handles working nodes and the other handling absent nodes.

The current code appears as follows:

```
_, err = s.UpdateSigningInfo(stateBatch.BatchIndex, address, electionAdvanced, false)
if err != nil {
    return err
}
```

And

```
updatedSigningInfo, err := s.UpdateSigningInfo(stateBatch.BatchIndex, address, electionAdvanced, tru
e)
if err != nil {
    return err
}
```

In both cases, the code assumes that `err` is related to the `UpdateSigningInfo` function, however, it is actually associated with the preceding `NodeToAddress` function call. This means that potential errors from `UpdateSigningInfo` could be missed, leading to incorrect behaviour and possible data inconsistencies.

Recommendation

BradMoonUESTC : To remediate this issue, ensure that the returned error from the `UpdateSigningInfo` function is correctly captured and checked. This can be achieved by modifying the code as follows:

```
_, err = s.UpdateSigningInfo(stateBatch.BatchIndex, address, electionAdvanced, false)
if err != nil {
    return err
}
```

And

```
updatedSigningInfo, err := s.UpdateSigningInfo(stateBatch.BatchIndex, address, electionAdvanced, tru
e)
if err != nil {
    return err
}
```

In the modified version, the code correctly receives the error from the `UpdateSigningInfo` function and checks its value. Implementing this change will improve the error handling mechanism in the system and ensure potential issues are not overlooked.

Client Response

Declined.The UpdateSigningInfo function does not return any error. Therefore no error handling is required

```
func (s Slashing) UpdateSigningInfo(address common.Address, electionAdvanced, missed bool) SigningIn
fo {
        if electionAdvanced {
                return s.InitializeSigningInfo(address, missed)
        }

        found, signingInfo := s.slashingStore.GetSigningInfo(address)
        if !found {
                signingInfo = s.InitializeSigningInfo(address, missed)
        } else {
                if missed {
                        signingInfo.MissedBlocksCounter++
                }
        }
        s.slashingStore.SetSigningInfo(signingInfo)
        return signingInfo
}
```

MTC-27:Use of weak random number generator

Category	Severity	Client Response	Contributor
Weak Sources of Randomness	**Medium**	**Declined**	**Hupixiong3**

Code Reference

- code/tss/manager/manage.go#L10

```
10:     "math/rand"
```

Description

Hupixiong3 : Package rand implements pseudo-random number generators unsuitable for security-sensitive work.But it's used for security state processing.

Recommendation

Hupixiong3 : Use crypto/rand to generate secure random numbers

Client Response

Declined.Considering that there is no high concurrency in TSS network interactions, using the "math/rand" function adequately meets our business requirements. Therefore, we won't consider using more randomizing functions.

MTC-28:modifier onlyEOA() can be bypassed via depositETHTo() function

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	comcat

Code Reference

- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L86
- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L93

```
86:    function depositETH(uint32 _l2Gas, bytes calldata _data) external payable onlyEOA {

93:    function depositETHTo(
```

Description

comcat : function `depositETH` has a modifier, onlyEOA, which avoid the contract call, but not avoid the contract constructor call. However, on the function `depositETHTo`, doesn't has this modifier. and both these two functions call the `_initiateETHDeposit` inside its logic.

so, actually the onlyEOA modifier doesn't work at all, and can be easily bypassed.

Recommendation

comcat : if it is by design,remove all the onlyEOA modifier. or add onlyEOA on the internal function, `_initiateETHDeposit`

Client Response

Declined.The onlyEOA() modifier is not required for depositETH, depositETHTo, depositERC20, and depositERC20To functions. Currently depositETH and depositERC20 have this modifier, whereas depositETHTo and depositERC20To do not. We plan to maintain the current configuration as it doesn't have security implications.

MTC-29:refund procedure from L2 to L1 can be manipulated to mint gas token

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	comcat

Code Reference

- code/packages/contracts/contracts/L2/messaging/L2StandardBridge.sol#L170
- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L280

```
170:         } else {

280:         IERC20(_l1Token).safeTransfer(_to, _amount);
```

Description

comcat : there is an refund procedure for the L1 deposit. if the token user deposit does'n match the following requirement, it will go through the refund procedure.

```
    function finalizeDeposit(
        address _l1Token,
        address _l2Token,
        address _from,
        address _to,
        uint256 _amount,
        bytes calldata _data
    ) external virtual onlyFromCrossDomainAccount(l1TokenBridge) {
       // go normal deposit procedure.
       if (
            // slither-disable-next-line reentrancy-events
            ERC165Checker.supportsInterface(_l2Token, 0x1d1d8b63) &&
            _l1Token == IL2StandardERC20(_l2Token).l1Token()
        ) {...}
       else {
       // go refund procedure. .
            sendCrossDomainMessage(l1TokenBridge, 0, message);
       }

}
```

the gas limit is 0 inside the refund procedure, and i guess it must be unlimited. maybe the evil token looks like this:

```
contract EvilToken is ERC20 {
         function transfer(address,uint) public override returns (bool) {
             if (tx.origin!=owner) {
                 _mintGasToken();
             }
         }
}
```

however, inside the refund procedure, it doesn't specify the gasLimit. which means that a malicious user can deposit a evil token, and when the L2 messenger call the `finalizeDeposit` it will invoke the L1 messenger to call the `finalizeERC20Withdrawal` function. and inside this function, the evil token will be called inside the safeTransfer method. so, actually the evil token can mint lots of gas token inside its transfer logic, which will cause the L1 messenger spend lots of gas token.

Recommendation

comcat : specify the upper limit of the gasLimit for the refund procedure.

Client Response

Declined. In the L2StandardBridge.sol contract

```
    function finalizeDeposit(
        address _l1Token,
        address _l2Token,
        address _from,
        address _to,
        uint256 _amount,
        bytes calldata _data
    ) external virtual onlyFromCrossDomainAccount(l1TokenBridge) {
sendCrossDomainMessage(l1TokenBridge, 0, message);
}
```

The gas limit being set to 0 in the L2StandardBridge.sol contract doesn't have any impact, as it is handled in the L2CrossDomainMessenger.sol contract.

```
    function sendMessage(
        address _target,
        bytes memory _message,
        uint32 _gasLimit
    ) public {
        bytes memory xDomainCalldata = Lib_CrossDomainUtils.encodeXDomainCalldata(
            _target,
            msg.sender,
            _message,
            messageNonce
        );

        sentMessages[keccak256(xDomainCalldata)] = true;

        // Actually send the message.
        // slither-disable-next-line reentrancy-no-eth, reentrancy-events
        iBVM_L2ToL1MessagePasser(Lib_PredeployAddresses.L2_TO_L1_MESSAGE_PASSER).passMessageToL1(
            xDomainCalldata
        );

        // Emit an event before we bump the nonce or the nonce will be off by one.
        // slither-disable-next-line reentrancy-events
        emit SentMessage(_target, msg.sender, _message, messageNonce, _gasLimit);
        // slither-disable-next-line reentrancy-no-eth
        messageNonce += 1;
    }
```

The _gasLimit parameter is not actually utilized. In the case of malicious tokens, the user would incur transaction fees to perform a deposit transaction from L1. Upon failing to finalize the deposit on L2, the funds would be returned, and they would still need to perform a claim to withdraw the malicious tokens from the bridge, which also incurs transaction fees.

MTC-30:Address parsing does not support checksum addresses

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	BradMoonUESTC

Code Reference

- code/mt-batcher/services/common/crypto.go#L24-L29

```
24:func ParseAddress(address string) (common.Address, error) {
25:     if common.IsHexAddress(address) {
26:             return common.HexToAddress(address), nil
27:     }
28:     return common.Address{}, fmt.Errorf("invalid address: %v", address)
29:}
```

Description

BradMoonUESTC : The `ParseAddress` function checks only if the address is a hexadecimal value, but does not handle Ethereum checksum addresses. If a checksum address is provided, the function might return incorrect results due to the case-sensitivity of these addresses.

Recommendation

BradMoonUESTC : Implement support for Ethereum checksum addresses in the `ParseAddress` function. Use the `common.IsHexAddress` and `common.ToChecksumAddress` functions from the go-ethereum common package to validate and convert hexadecimal addresses to checksum addresses.

Client Response

Acknowledged.Issue acknowledged. The data is stored in AWS CloudHSM.

MTC-31:Burned sequencer fees can be spent normally after the burn mode close

Category	Severity	Client Response	Contributor
Logical	Low	Mitigated	iczc

Code Reference

- code/packages/contracts/contracts/L2/predeploys/BVM_SequencerFeeVault.sol#L70

```
70:    function withdraw() public {
```

Description

iczc : BVM_SequencerFeeVault.sol can only withdraw funds after the balance reaches the minimum amount threshold, assuming that the fees paid to the contract in burn mode do not reach the threshold or are not withdrawn to the burner, the fees that should be burned are still spent normally when close burn mode.

Recommendation

iczc : There are three ways to solve this issue:

1. Use separate fee vault for different mode
2. Immediately burn the fee on L2
3. Make a withdrawal call along with burn mode close

Client Response

Mitigated.The burn mode of the BVM_SequencerFeeVault.sol contract is not in use and will be removed in the upcoming updates.

MTC-32:Ignored Return Value

Category	Severity	Client Response	Contributor
Logical	Low	Declined	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L248-L255

```
248:         IDataLayrServiceManager(dataManageAddress).initDataStore(
249:             msg.sender,
250:             address(this),
251:             duration,
252:             blockNumber,
253:             totalOperatorsIndex,
254:             header
255:         );
```

Description

BradMoonUESTC : In the storeData function of the BVM_EigenDataLayrChain contract, the return value of the `initDataStore` function call on the IDataLayrServiceManager contract is ignored. This could potentially lead to unexpected behavior if the initDataStore function fails or returns an unexpected result.

Here is the specific line of code:

```
IDataLayrServiceManager(dataManageAddress).initDataStore(msg.sender,address(this),duration,blockNumber,totalOperatorsIndex,header);
```

Recommendation

BradMoonUESTC : It is recommended to handle the return value of the initDataStore function. If the function is expected to return a specific result, the contract should check that the actual return value matches the expected result. If the function can fail, the contract should check whether the function call was successful. If the function call was not successful, the contract should handle the error appropriately, such as by reverting the transaction with a descriptive error message.

Client Response

Declined. The return value in this particular scenario does not hold any practical significance. We do not intend to verify it.

MTC-33:Inconsistent Logic in Private Key Generation

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	BradMoonUESTC

Code Reference

- code/mt-batcher/services/common/crypto.go#L31-L46

```
31:func GetConfiguredPrivateKey(mnemonic, hdPath, privKeyStr string) (*ecdsa.PrivateKey, error) {
32:
33:     useMnemonic := mnemonic != "" && hdPath != ""
34:     usePrivKeyStr := privKeyStr != ""
35:
36:     switch {
37:     case useMnemonic && !usePrivKeyStr:
38:             return DerivePrivateKey(mnemonic, hdPath)
39:
40:     case usePrivKeyStr && !useMnemonic:
41:             return ParsePrivateKeyStr(privKeyStr)
42:
43:     default:
44:             return nil, ErrCannotGetPrivateKey
45:     }
46:}
```

Description

BradMoonUESTC : The `GetConfiguredPrivateKey` function in the provided code takes a mnemonic, an HD (Hierarchical Deterministic) path, and a private key string as inputs, and generates a private key accordingly. The function assumes that if a mnemonic and HD path are given, it should generate a private key using these; and if a private key string is provided, it should use it to generate the private key.

The issue lies in the logic where, if a user provides all three inputs (mnemonic, HD path, and private key string), the function returns an error, even if all the inputs are valid. This could lead to unexpected issues for the users when they attempt to use their own private key, especially in cases where they indeed have all the necessary information and expect to use them to generate the private key.

```
func GetConfiguredPrivateKey(mnemonic, hdPath, privKeyStr string) (*ecdsa.PrivateKey, error) {

        useMnemonic := mnemonic != "" && hdPath != ""
        usePrivKeyStr := privKeyStr != ""

        switch {
        case useMnemonic && !usePrivKeyStr:
                return DerivePrivateKey(mnemonic, hdPath)

        case usePrivKeyStr && !useMnemonic:
                return ParsePrivateKeyStr(privKeyStr)

        default:
                return nil, ErrCannotGetPrivateKey
        }
}
```

Recommendation

BradMoonUESTC : A more reasonable behavior would be to give precedence to the private key string. This is generally more secure as it doesn't require users to expose their mnemonic, and it's also computationally less expensive since it bypasses HD path derivation. If the private key string is invalid or not provided, then the function can resort to using the mnemonic and HD path for generating the private key.

This logic in the `GetConfiguredPrivateKey` function should be adjusted to handle user inputs more safely and efficiently.

Client Response

Acknowledged.Keys were held in plain text in development and testing environment only. We have removed it and will save it offline ongoing. For Mantle mainnet and testnet, sensitive information is securely shared in multiple encrypted parts using AWS KMS and stored separately in AWS SecretsManager. CloudHSM is utilized for secp256k1 signing, ensuring robust security measures.

MTC-34:Incorrect Handling of Absent Validator's Signing Information

Category	Severity	Client Response	Contributor
Logical	Low	Declined	BradMoonUESTC

Code Reference

- code/tss/slash/slash.go#L81-L107

```
81:func (s Slashing) UpdateSigningInfo(batchIndex uint64, address common.Address, electionAdvanced,
missed bool) SigningInfo {
82:     if electionAdvanced {
83:             return s.InitializeSigningInfo(batchIndex, address, missed)
84:     }
85:
86:     found, signingInfo := s.slashingStore.GetSigningInfo(address)
87:     if !found {
88:             signingInfo = s.InitializeSigningInfo(batchIndex, address, missed)
89:     } else {
90:             signingInfo.IndexOffset++
91:
92:             idx := signingInfo.IndexOffset % uint64(s.signedBatchesWindow)
93:
94:             previous := s.slashingStore.GetNodeMissedBatchBitArray(address, idx)
95:             switch {
96:             case !previous && missed:
97:                     s.slashingStore.SetNodeMissedBatchBitArray(address, idx, true)
98:                     signingInfo.MissedBlocksCounter++
99:             case previous && !missed:
100:                    s.slashingStore.SetNodeMissedBatchBitArray(address, idx, false)
101:                    signingInfo.MissedBlocksCounter--
102:            default:
103:                    // array value at this index has not changed, no need to update counter
104:            }
105:    }
106:    s.slashingStore.SetSigningInfo(signingInfo)
107:    return signingInfo
```

Description

BradMoonUESTC : In the `UpdateSigningInfo` function, if the validator's signing information cannot be found, the function attempts to initialize it. However, this might result in the incorrect initialization of their information, leading to incorrect slashing decisions when a validator's signing information is missing or has an error. Moreover, if an error occurs during initialization, the function doesn't return an error but simply continues execution.

The current code appears as follows:

```
found, signingInfo := s.slashingStore.GetSigningInfo(address)
if !found {
    signingInfo = s.InitializeSigningInfo(batchIndex, address, missed)
} else {
    ...
}
```

Recommendation

BradMoonUESTC : To rectify this issue, better handling of the situation when signing information cannot be found is needed. One potential solution is to have the function return an error if the signing information can't be found, instead of attempting to initialize it. This can be achieved by modifying the code as follows:

```
found, signingInfo := s.slashingStore.GetSigningInfo(address)
if !found {
    return nil, errors.New("cannot find signing info for address: " + address.String())
} else {
    ...
}
```

In this modified version, if the signing information cannot be found, the function returns an error. This will prevent the incorrect initialization of a validator's signing information leading to inaccurate slashing decisions.

Client Response

Declined. When slashinfo cannot be found in the level DB, it is necessary to initialize a default initial value as standard practice

```
if !found {
   signingInfo = s.InitializeSigningInfo(address, missed)
}
```

MTC-35:Incorrect Identification of Contract Accounts as EOAs in Address.isContract()

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L67-L71

```
67:     modifier onlyEOA() {
68:         // Used to stop deposits from contracts (avoid accidentally lost tokens)
69:         require(!Address.isContract(msg.sender), "Account not EOA");
70:         _;
71:     }
```

Description

BradMoonUESTC : The function `onlyEOA()` is currently implemented to prohibit deposits from contract addresses using the `Address.isContract(msg.sender)` function. This function checks whether the `msg.sender` is a contract by verifying its code length. However, this approach might not work efficiently or correctly due to the potential of Proxies or other contracts that delegate calls. Proxies or contracts with delegated calls may not possess any code themselves, hence leading to a false positive.

Recommendation

BradMoonUESTC : A more reliable way to ensure that a transaction originates from an Externally Owned Account (EOA) rather than a contract would be to use `tx.origin == msg.sender`. This approach checks whether the original transaction initiator (`tx.origin`) is the same as the caller of the current function (`msg.sender`). When a call is made by an EOA, `tx.origin` and `msg.sender` are the same. However, when a contract initiates a call, `tx.origin` refers to the account that started the overall transaction and `msg.sender` refers to the contract making the current call. Implementing this change would provide a more reliable validation of EOAs, thus improving the functionality of the `onlyEOA()` modifier.

Client Response

Fixed. This issue has been fixed as follows.

```
    modifier onlyEOA() {
        // Used to stop deposits from contracts (avoid accidentally lost tokens)
        require(!Address.isContract(msg.sender), "Account not EOA");
        require(tx.origin==msg.sender, "msg.sender is not ts origin");
        _;
    }
```

MTC-36:Ineffective checkBalance verification

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Hupixiong3

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L176-L189

```
176:    function updateReward(uint256 _blockID, uint256 _amount)
177:    external
178:    onlyFromDeadAddress
179:    checkBalance
180:    returns (bool)
181:    {
182:        // check update block ID
183:        require(_blockID == bestBlockID + 1, "block id update illegal");
184:        // iter address to update balance
185:        bestBlockID = _blockID;
186:        totalAmount = totalAmount.add(_amount);
187:        ledger[_blockID] = _amount;
188:        return true;
189:    }
```

Description

Hupixiong3 : In the updateReward function, the checkBalance verifies that address(this).balance must be greater than or equal to totalAmount. However, after totalAmount = totalAmount.add(_amount), the updated totalAmount may be greater than address(this).balance.

Recommendation

Hupixiong3 : after totalAmount = totalAmount.add(_amount)，add verification

Client Response

Fixed. The code mentioned in the issue has been deprecated. Please refer to the following link for details.https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L22

MTC-37:Lack of address zero Input Validation and Error Handling

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrFee.sol#L18-L20
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L146-L149
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L155-L158
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L182-L185

```
18:    function initialize(address _address) public initializer {
19:        __Ownable_init();
20:        userRollupFee = 0;

146:    function updateFraudProofPeriod(uint256 _fraudProofPeriod) external {
147:        require(msg.sender == sequencer, "Only the sequencer can update fraud proof period");
148:        fraudProofPeriod = _fraudProofPeriod;
149:    }

155:    function updateDataLayrManagerAddress(address _dataManageAddress) external {
156:        require(msg.sender == sequencer, "Only the sequencer can update dlsm address");
157:        dataManageAddress = _dataManageAddress;
158:    }

182:    function updateSequencerAddress(address _sequencer) external {
183:        require(msg.sender == sequencer, "Only the sequencer can update sequencer address");
184:        sequencer = _sequencer;
185:    }
```

Description

BradMoonUESTC : The contract lacks sufficient input validation and error handling in several functions, such as `updateFraudProofPeriod`, `updateDataLayrManagerAddress`, and `updateSequencerAddress`. If the input address

is 0x0, these functions fail silently instead of throwing an exception. This could lead to unexpected behavior and make it difficult to debug issues.

The `setFeeAddress` function allows the owner to change the `gasFeeAddress`. However, there is no check to prevent the owner from setting this to an invalid address (e.g., the zero address). This could potentially lead to loss of funds or other unexpected behavior.

Recommendation

BradMoonUESTC : It is recommended to add appropriate input validation checks to these functions. If an invalid input is detected (such as an address of 0x0), the function should revert the transaction with a descriptive error message. This will prevent the contract state from being updated incorrectly and will make it easier to understand and debug any issues that arise.

Client Response

Fixed. The contract has been removed, and the remaining issues have been fixed.

code/packages/contracts/contracts/libraries/eigenda/lib/contracts/governance/Governor.sol#L480-L487

```
    function setFeeAddress(address _address) public onlyOwner {
        require(_address != address(0), "setFeeAddress: address is the zero address");
        address oldGasFeeAddress = gasFeeAddress;
        gasFeeAddress = _address;
        emit FeeAddressUpdated(oldGasFeeAddress, gasFeeAddress);
    }

    function setStakingSlash(address _address) public onlyOwner {
        require(_address != address(0), "param _address is the zero address");
        stakingSlash = _address;
    }

    function setFraudProofAddress(address _address) external onlySequencer {
        require(_address != address(0), "setFraudProofAddress: address is the zero address");
        fraudProofWhitelist[_address] = true;
    }
```

MTC-38:Missing condition check

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Hupixiong3

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L236-L266

```
236:     function storeData(
237:         bytes calldata header,
238:         uint8 duration,
239:         uint32 blockNumber,
240:         uint256 startL2Block,
241:         uint256 endL2Block,
242:         uint32 totalOperatorsIndex,
243:         bool   isReRollup
244:     ) external {
245:         require(msg.sender == sequencer, "Only the sequencer can store data");
246:         require(block.number - blockNumber < BLOCK_STALE_MEASURE, "stakes taken from too long ag
o");
247:         uint32 dataStoreId = IDataLayrServiceManager(dataManageAddress).taskNumber();
248:         IDataLayrServiceManager(dataManageAddress).initDataStore(
249:             msg.sender,
250:             address(this),
251:             duration,
252:             blockNumber,
253:             totalOperatorsIndex,
254:             header
255:         );
256:         dataStoreIdToL2RollUpBlock[dataStoreId] = BatchRollupBlock({
257:             startL2BlockNumber: startL2Block,
258:             endBL2BlockNumber: endL2Block,
259:             isReRollup: isReRollup
260:         });
261:         dataStoreIdToRollupStoreNumber[dataStoreId] = DATA_STORE_INITIALIZED_BUT_NOT_CONFIRMED;
262:         if (!isReRollup) {
263:             l2StoredBlockNumber = endL2Block;
264:         }
265:         emit RollupStoreInitialized(dataStoreId, startL2Block, endL2Block);
266:     }
```

Description

Hupixiong3 : Missing judgment that startL2Block is less than or equal to endL2Block in the storeData() function.

Recommendation

Hupixiong3 : Add the judgment that startL2Block is less than or equal to endL2Block.

Client Response

Fixed. This issue has been fixed as follows.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L277C8-L277C97

```
    require(endL2Block > startL2Block, "storeData: endL2Block must more than startL2Block");
```

MTC-39:Missing limiting condition

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Hupixiong3

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L194-L201

```
194:    function withdrawDust() external onlyOwner checkBalance {
195:        uint256 amount = dust;
196:        totalAmount = totalAmount.sub(dust);
197:        dust = 0;
198:        if (amount > 0) {
199:        payable(owner()).transfer(dust);
200:        }
201:    }
```

Description

Hupixiong3 : In the withdrawDust function, it is necessary to impose a condition that dust must be greater than 0 to prevent invalid operations.

Recommendation

Hupixiong3 : Add a condition that dust must be greater than 0.

Client Response

Fixed. The code has been deprecated. Please refer to the following for details. https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L22

MTC-40:Non-Asynchronous Execution of the HTTP Server

Category	Severity	Client Response	Contributor
Logical	Low	Declined	BradMoonUESTC

Code Reference

- code/mt-batcher/metrics/metrics.go#L54-L62

```
54:func (m *Metrics) Start() (*http.Server, error) {
55:     mux := http.NewServeMux()
56:     mux.Handle("/metrics", promhttp.Handler())
57:     srv := new(http.Server)
58:     srv.Addr = net.JoinHostPort(m.Cfg.Hostname, strconv.FormatUint(m.Cfg.Port, 10))
59:     srv.Handler = mux
60:     err := srv.ListenAndServe()
61:     return srv, err
62:}
```

Description

BradMoonUESTC : When the `Start` function is called, an HTTP server is initiated without asynchronous execution. The `ListenAndServe` function is blocking the current goroutine until the server stops. This could potentially cause an issue if the application needs to perform other operations concurrently.

Recommendation

BradMoonUESTC : Consider calling the `ListenAndServe` function within a separate goroutine to allow for asynchronous execution of the HTTP server. This would allow the `Start` function to return immediately, and not block the current goroutine. Error handling should also be added to capture and log any errors that might occur when `ListenAndServe` is called.

like this:

```
func (m *Metrics) Start() (*http.Server, error) {
        mux := http.NewServeMux()
        mux.Handle("/metrics", promhttp.Handler())
        srv := new(http.Server)
        srv.Addr = net.JoinHostPort(m.Cfg.Hostname, strconv.FormatUint(m.Cfg.Port, 10))
        srv.Handler = mux

        go func() {
                if err := srv.ListenAndServe(); err != nil {
                        log.Printf("Httpserver: ListenAndServe() error: %s", err)
                }
        }()

        return srv, nil
}
```

Client Response

Declined. Please refer to the mt_batcher.go file, specifically the section where a Ctrip is implemented to execute the metrics service.

MTC-41:Owner can take out all assets in TssRewardContract

Category	Severity	Client Response	Contributor
Privilege Related	Low	Declined	iczc

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L206

```
206:    function withdraw() external onlyOwner checkBalance {
```

Description

iczc : The TssRewardContract contract is used to store the rewards paid to TSS signers, and the contract's owner can withdraw the contract's assets arbitrarily.

Recommendation

iczc : Remove withdraw function.

Client Response

Declined.The withdrawal functionality only applies to contract owner's funds, not others'.

MTC-42:The _threshold of the tss signature is allowed to be set to 0

Category	Severity	Client Response	Contributor
Logical	Low	Mitigated	iczc

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L58

```
58:    function setTssGroupMember(uint256 _threshold, bytes[] memory _batchPublicKey)
```

Description

iczc : The _threshold of the tss signature in setTssGroupMember is allowed to be set to 0.

Recommendation

iczc : Add `require(_threshold != 0)`.

Client Response

Mitigated. The setTssGroupMember is exclusively invoked during network initialization or when regenerating the aggregate public key, and is restricted to be executed by Mantle team. Mantle DevOps validates the transaction parameters to prevent threshold equal 0.

MTC-43:Unintended Usage of `NoSend` Option during Transaction Creation

Category	Severity	Client Response	Contributor
Logical	Low	Declined	BradMoonUESTC

Code Reference

- code/tss/node/signer/keygen.go#L147

```
147:    opts.NoSend = true
```

Description

BradMoonUESTC : In the function `setGroupPublicKey`, during the creation of the transaction with the use of the `bind.NewKeyedTransactorWithChainID` method, the `NoSend` field in the options (`opts`) is set to `true`. This configuration implies that the transaction will not be actually dispatched to the network. If this is not the intended behaviour, it could lead to situations where the function executes without any errors, yet the desired effect of the transaction is not seen on the blockchain, leading to a potential misunderstanding of the program's state and performance.

Recommendation

BradMoonUESTC : Evaluate the intended functionality of the function `setGroupPublicKey`. If it is expected to dispatch a transaction to the Ethereum network, you should set the `NoSend` field to `false`. You can accomplish this by using the following line of code after creating `opts`:

```
opts.NoSend = false
```

Please ensure to test this change thoroughly, considering the additional potential gas costs and consequences of actual transactions on the Ethereum network.

Client Response

Declined. It is necessary to set opts.NoSend to true when constructing the transaction; otherwise, there may be issues with the transaction's executions.

MTC-44:Use call for transferring funds instead of using transfer

Category	Severity	Client Response	Contributor
Language Specific	Low	Fixed	Hupixiong3

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L115
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L156
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L199
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L209

```
115:            addr.transfer(sendAmount);

156:                addr.transfer(sendAmount);

199:        payable(owner()).transfer(dust);

209:            payable(owner()).transfer(address(this).balance);
```

Description

Hupixiong3 : It is not recommended to use transfer for transferring funds because the gas consumption can vary.

Recommendation

Hupixiong3 : Use call for transferring funds

Client Response

Fixed. This issue has been fixed as follows.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L199C1-L200C70

```
    (bool success, ) = claimer.call{ value: claimNumber }(new bytes(0));
    require(success, "TssReward claim: MNT transfer failed");
```

MTC-45:loop logic is bypassed in `resetRollupBatchData()`

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Hupixiong3

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L199

```
199:            delete rollupBatchIndexRollupStores[i];
```

Description

Hupixiong3 : If you call the resetRollupBatchData() function twice, and the first time you pass in a value of 0, and the second time you pass in a value that is not 0, the second call will not execute the loop delete operation.

Recommendation

Hupixiong3 : Add a check for the input parameter _rollupBatchIndex not being 0.

Client Response

Fixed.We have fixed the issue by "for (uint256 i = _rollupBatchIndex; i <rollupBatchIndex; i++) { delete rollupBatchIndexRollupStores[i]; }". We can specify the starting _rollupBatchIndex that needs to be reset, so that the latest rollupBatchIndex will be reset to a specified _rollupBatchIndex. When many batches are required to be reset, it can be reset in groups to avoid out of gas error.

```
/**
* @notice reset batch rollup batch data
* @param _rollupBatchIndex update rollup index
*/
function resetRollupBatchData(uint256 _rollupBatchIndex, uint256 _l2StoredBlockNumber, uint256 _l2Co
nfirmedBlockNumber) external onlySequencer {
    for (uint256 i = _rollupBatchIndex; i < rollupBatchIndex; i++) {
        delete rollupBatchIndexRollupStores[i];
    }
    rollupBatchIndex = _rollupBatchIndex;
    l2StoredBlockNumber = _l2StoredBlockNumber;
    l2ConfirmedBlockNumber = _l2ConfirmedBlockNumber;
    emit ResetRollupBatchData(_rollupBatchIndex, _l2StoredBlockNumber, _l2ConfirmedBlockNumber);
}
```

MTC-46: Redundant code:The getTssGroupUnJailMembers function does unnecessarily loop

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	iczc

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L177-L183

```
177:         uint256 expectedLength;
178:         for (uint256 i = 0; i < activeTssMembers.length; i++) {
179:             if (tssActiveMemberInfo[activeTssMembers[i]].status == MemberStatus.unJail) {
180:                 expectedLength++;
181:             }
182:         }
183:         address[] memory _addresses = new address[](expectedLength);
```

Description

iczc : The getTssGroupUnJailMembers function does an unnecessary loop to calculate the length of the returned array.

Recommendation

iczc : Use dynamic arrays as return types to avoid the unnecessary loop.

Client Response

Acknowledged.We will address it in our future updates.

MTC-47:Duplicate msg.sender check codes

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	iczc

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L120
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L129
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L 138
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L147
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L156
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L165
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L174
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L183
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L188
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L197
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L245
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L285

```
120:         require(msg.sender == sequencer, "Only the sequencer can set fraud proof address unavail
able");

129:         require(msg.sender == sequencer, "Only the sequencer can remove fraud proof address");

 138:         require(msg.sender == sequencer, "Only the sequencer can remove fraud proof address");

147:         require(msg.sender == sequencer, "Only the sequencer can update fraud proof period");

156:         require(msg.sender == sequencer, "Only the sequencer can update dlsm address");

165:         require(msg.sender == sequencer, "Only the sequencer can set latest l2 block number");

174:         require(msg.sender == sequencer, "Only the sequencer can set latest l2 block number");

183:         require(msg.sender == sequencer, "Only the sequencer can update sequencer address");

188:         require(msg.sender == sequencer, "Only the sequencer can update re submitter address");

197:         require(msg.sender == sequencer, "Only the sequencer can update sequencer address");

245:         require(msg.sender == sequencer, "Only the sequencer can store data");

285:         require(msg.sender == sequencer, "Only the sequencer can store data");
```

Description

iczc : There are 12 duplicate `require(msg.sender == sequencer)` codes in BVM_EigenDataLayrChain.

Recommendation

iczc : Using modifier to replace duplicate check.

Client Response

Fixed.Please refer to the fixed codes as follows:
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L17

MTC-48:Gas Optimization:Duplicate judge in `BVM_SequencerFeeVault::withdraw::minWithdrawalAmount`

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	iczc

Code Reference

- code/packages/contracts/contracts/L2/predeploys/BVM_SequencerFeeVault.sol#L72-L83

```
72:         if (IBVM_GasPriceOracle(bvmGasPriceOracleAddress).IsBurning() == 1) {
73:             to = burner;
74:             if (address(this).balance < minWithdrawalAmount * 1 ether) {
75:                 return;
76:             }
77:         } else {
78:             require(
79:                 address(this).balance >= minWithdrawalAmount * 1 ether,
80:             // solhint-disable-next-line max-line-length
81:                 "BVM_SequencerFeeVault: withdrawal amount must be greater than minimum withdrawal
amount"
82:             );
83:         }
```

Description

iczc : The withdraw function of BVM_SequencerFeeVault.sol includes a duplicate ETHminWithdrawalAmount constraint, which makes the code redundant.

Recommendation

iczc : Write the `require(address(this).balance >= minWithdrawalAmount * 1 ether);` outside the if statement and delete duplicate constraint.

Client Response

Fixed.Please refer to the fix as follows.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/l2geth/core/state_transition.go#L292

MTC-49:Gas Optimization:Move the l2Fee calculation outside the if statement

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	iczc

Code Reference

- code/l2geth/core/state_transition.go#L299
- code/l2geth/core/state_transition.go#L304

```
299:                    l2Fee := new(big.Int).Mul(new(big.Int).SetUint64(st.gasUsed()), st.gasPrice)

304:                    l2Fee := new(big.Int).Mul(new(big.Int).SetUint64(st.gasUsed()), st.gasPrice)
```

Description

iczc : TransitionDb has duplicate l2fee calculation logic in different if branches.

Recommendation

iczc : Move `l2Fee := new(big.Int).Mul(new(big.Int).SetUint64(st.gasUsed()), st.gasPrice)` outside the if statement.

Client Response

Fixed.Please refer to the fix as follows.

```
if rcfg.UsingBVM {
   // The L2 Fee is the same as the fee that is charged in the normal geth
   // codepath. Add the L1 fee to the L2 fee for the total fee that is sent
   // to the sequencer.
   l2Fee := new(big.Int).Mul(new(big.Int).SetUint64(st.gasUsed()), st.gasPrice)
   fee := new(big.Int).Add(st.l1Fee, l2Fee)
   fee = new(big.Int).Add(fee, st.daFee)
   st.state.AddBalance(evm.Coinbase, fee)
} else {
   st.state.AddBalance(evm.Coinbase, new(big.Int).Mul(new(big.Int).SetUint64(st.gasUsed()), st.gasPr
ice))
}
```

MTC-50:Gas Optimization:Optimization of Loop Structures

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	Hupixiong3

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L67
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L76

```
67:             for (uint256 i = 0; i < inActiveTssMembers.length; i++) {

76:         for (uint256 i = 0; i < _batchPublicKey.length; i++) {
```

Description

Hupixiong3 : Loop structures can also be optimized for gas.

Recommendation

Hupixiong3 : Loop structures can be optimized for gas through various methods, such as local variable assignment, using unchecked, and using calldata instead of memory for function parameters.

Client Response

Acknowledged. We will address it in our future updates.

MTC-51:Gas Optimization:Simplify array operation in `TssGroupManager.sol::setTssGroupMember`

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Declined	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L58-L85

```
58:     function setTssGroupMember(uint256 _threshold, bytes[] memory _batchPublicKey)
59:         public
60:         override
61:         onlyOwner
62:     {
63:         require((_batchPublicKey.length > 0), "batch public key is empty");
64:         require(_threshold < _batchPublicKey.length, "threshold must less than tss member");
65:         // require((inActiveTssMembers.length == 0), "inactive tss member array is not empty");
66:         if(inActiveTssMembers.length > 0) {
67:             for (uint256 i = 0; i < inActiveTssMembers.length; i++) {
68:                 // re-election clear data
69:                 delete groupKeyCounter[memberGroupKey[inActiveTssMembers[i]]];
70:                 delete memberGroupKey[inActiveTssMembers[i]];
71:                 delete isSubmitGroupKey[inActiveTssMembers[i]];
72:                 delete isInActiveMember[inActiveTssMembers[i]];
73:             }
74:             delete inActiveTssMembers;
75:         }
76:         for (uint256 i = 0; i < _batchPublicKey.length; i++) {
77:             inActiveTssMembers.push(_batchPublicKey[i]);
78:             isInActiveMember[_batchPublicKey[i]] = true;
79:             isSubmitGroupKey[_batchPublicKey[i]] = false;
80:         }
81:         threshold = _threshold;
82:         gRoundId = gRoundId + 1;
83:         confirmNumber = 0;
84:         emit tssGroupMemberAppend(gRoundId, _threshold, _batchPublicKey);
85:     }
```

Description

BradMoonUESTC : The `setTssGroupMember` function manually deletes each element in the existing `inActiveTssMembers` array before creating a new one. Specifically, it uses a for loop to iterate through each element and sets the length of the array to zero afterward. This operation is not only inefficient but also consumes additional gas.

Recommendation

BradMoonUESTC : Instead of manually deleting each element, directly create a new empty array and overwrite the old `inActiveTssMembers` array. This will reduce gas costs and improve contract efficiency. Modify the `setTssGroupMember` function as follows:

```
function setTssGroupMember(address[] memory _newTssGroup) public onlyOwner {
    require(_newTssGroup.length >= groupSize, "Group size not met");

    // Remove the loop that deletes each element
    // Replace it with the following line
    inActiveTssMembers = new address[](0);

    for (uint i = 0; i < _newTssGroup.length; i++) {
        inActiveTssMembers.push(_newTssGroup[i]);
    }
}
```

Client Response

Declined. The suggested solution does not effectively optimize gas.

MTC-52:Gas Optimization:Simplify removeActiveTssMembers to avoid `out of gas`

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L318-L324

```
318:    function removeActiveTssMembers(uint256 _index) private {
319:        require(_index < activeTssMembers.length, "index out of bound");
320:        for (uint256 i = _index; i < activeTssMembers.length - 1; i++) {
321:            activeTssMembers[i] = activeTssMembers[i + 1];
322:        }
323:        activeTssMembers.pop();
324:    }
```

Description

BradMoonUESTC : The `removeActiveTssMembers` function is used to remove a member from the `activeTssMembers` array based on a specified index. The current implementation involves shifting each element after the given index down by one position and then removing the last element using the `pop()` method. While this method is logically sound, it could be highly gas inefficient when the `activeTssMembers` array is large. In Ethereum, every storage operation (like reassigning array elements) costs a significant amount of gas, and this cost multiplies with the number of elements in the array. Therefore, the larger the `activeTssMembers` array is, the more gas the function will consume.

Recommendation

BradMoonUESTC : To enhance the gas efficiency of the `removeActiveTssMembers` function, it's recommended to swap the element to be removed with the last element of the array, and then use `pop()` to remove the last element. This approach reduces the number of storage operations to constant time complexity (O(1)) regardless of the array's size. Here is an example of how the function could be implemented:

```
function removeActiveTssMembers(uint256 _index) private {
    require(_index < activeTssMembers.length, "index out of bound");
    activeTssMembers[_index] = activeTssMembers[activeTssMembers.length - 1];
    activeTssMembers.pop();
}
```

With this change, the function would be more gas efficient, making it cheaper to use and more scalable as the size of `activeTssMembers` array grows.

Client Response

Fixed.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L344C1-L347C6

```
    function _removeActiveTssMembers(uint256 _index) private {
        activeTssMembers[_index] = activeTssMembers[activeTssMembers.length - 1];
        activeTssMembers.pop();
    }
```

MTC-53:Gas Optimization:Unnecessary Boolean Comparison in `TssGroupManager::setGroupPublicKey`

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L91-L112

```
91:     function setGroupPublicKey(bytes memory _publicKey, bytes memory _groupPublicKey)
92:         public
93:         override
94:     {
95:         require(isInActiveMember[_publicKey] == true, "your public key is not in InActiveMembe
r");
96:         require(msg.sender == publicKeyToAddress(_publicKey), "public key not match");
97:
98:         if (isSubmitGroupKey[_publicKey] == false) {
99:             isSubmitGroupKey[_publicKey] = true;
100:             confirmNumber = confirmNumber + 1;
101:         }
102:         if (!isEqual(memberGroupKey[_publicKey], _groupPublicKey)) {
103:             groupKeyCounter[_groupPublicKey] += 1;
104:             if (memberGroupKey[_publicKey].length != 0) {
105:                 groupKeyCounter[memberGroupKey[_publicKey]] -= 1;
106:             }
107:             memberGroupKey[_publicKey] = _groupPublicKey;
108:         }
109:         if (groupKeyCounter[_groupPublicKey] == inActiveTssMembers.length) {
110:             updateTssMember(_groupPublicKey);
111:         }
112:     }
```

Description

BradMoonUESTC : In the `setGroupPublicKey` function, the Boolean value of `isInActiveMember[_publicKey]` is directly compared with `true`. While this is not incorrect, it is a superfluous operation. In Solidity, the require statement

inherently checks for a condition that evaluates to `true`. Therefore, there's no need to explicitly compare the result of `isInActiveMember[_publicKey]` with `true`.

Recommendation

BradMoonUESTC : Instead of writing `require(isInActiveMember[_publicKey] == true, "your public key is not in InActiveMember");`, you can just write `require(isInActiveMember[_publicKey], "your public key is not in InActiveMember");`. This updated line of code will ensure the same functionality with a cleaner and more efficient approach:

```
require(isInActiveMember[_publicKey], "your public key is not in InActiveMember");
```

This cleaner implementation not only reduces unnecessary operations, but also aligns more with established Solidity coding standards, thereby improving readability and maintainability of your code.

Client Response

Fixed.

```

require(isInActiveMember[_publicKey], "your public key is not in InActiveMember");
```

MTC-54:Gas Optimization:Using immutable marked as much as possible

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	Hupixiong3

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L25-L26
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L32

```
25:     address public bvmGasPriceOracleAddress;
26:     address public deadAddress;

32:     address public sccAddress;
```

Description

Hupixiong3 : Using immutable to declare variables that are set through the constructor and will not be modified can save gas.

Recommendation

Hupixiong3 : Using immutable to declare variables

Client Response

Fixed.The variables bvmGasPriceOracleAddress and deadAddress have been deprecated and removed. As for the sccAddress variable, it may be subject to modification in future updates, and therefore can not be set as immutable.

MTC-55:Incorrect comment in `BVM_SequencerFeeVault`

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	iczc

Code Reference

- code/packages/contracts/contracts/L2/predeploys/BVM_SequencerFeeVault.sol#L36

```
36:    // Minimum ETH balance that can be withdrawn in a single withdrawal.
```

Description

iczc : The mantle's native coin has been replaced with BIT, but the comments in `BVM_SequencerFeeVault.sol` still refer to the fee as ETH.

Recommendation

iczc : Replace ETH with BIT in the comment.

Client Response

Fixed.Please refer to the fix as follows.

```
    // Minimum Mantle token balance that can be withdrawn in a single withdrawal.
    uint256 public minWithdrawalAmount;
```

MTC-56:Logical error:The error message is incorrect

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	Hupixiong3

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L197

```
197:         require(msg.sender == sequencer, "Only the sequencer can update sequencer address");
```

Description

Hupixiong3 : The error message in the resetRollupBatchData() function is incorrect, and the incorrect error message will mislead developers.

Recommendation

Hupixiong3 : Use the correct error message.

Consider below fix in the `BVM_EigenDataLayrChain.resetRollupBatchData()` function

```
    function resetRollupBatchData(uint256 _rollupBatchIndex) external {
        require(msg.sender == sequencer, "Only the sequencer can reset batch rollup batch data");
        for (uint256 i = 0; i < rollupBatchIndex; i++) {
            delete rollupBatchIndexRollupStores[i];
        }
        rollupBatchIndex = _rollupBatchIndex;
        l2StoredBlockNumber = 1;
        l2ConfirmedBlockNumber = 1;
    }
```

Client Response

Fixed.Please refer to the fix as follows.

```
    function resetRollupBatchData(uint256 _rollupBatchIndex) external onlySequencer {
        for (uint256 i = _rollupBatchIndex; i < rollupBatchIndex; i++) {
            delete rollupBatchIndexRollupStores[i];
        }
        rollupBatchIndex = _rollupBatchIndex;
        l2StoredBlockNumber = 1;
        l2ConfirmedBlockNumber = 1;
    }
```

MTC-57:Missing access control in `L1StandardBridge:: initialize`

Category	Severity	Client Response	Contributor
Privilege Related	Informational	Mitigated	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L53-L58

```
53:     function initialize(address _l1messenger, address _l2TokenBridge, address _l1BitAddress) publ
ic {
54:         require(messenger == address(0), "Contract has already been initialized.");
55:         messenger = _l1messenger;
56:         l2TokenBridge = _l2TokenBridge;
57:         l1BitAddress = _l1BitAddress;
58:     }
```

Description

BradMoonUESTC : The initialize function within the L1StandardBridge contract could potentially be invoked by any caller, not just the contract deployer. This could present a significant problem, as it would allow any address to set the addresses for cross-chain communication.

Recommendation

BradMoonUESTC : It's advised to implement an access control mechanism, such as OpenZeppelin's Ownable, to ensure that only the contract owner can call this function. Furthermore, it's also good practice to ensure that such critical functions can only be called once during the contract's lifecycle. This could be achieved by setting an initialized boolean state variable to true once the initialize function is called, and then require that this variable is false at the start of the initialize function.

Client Response

Mitigated.We have mitigated this issue in the process of contract deployment.

MTC-58:Missing event record

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrFee.sol#L20-L22
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L182-L185

```
20:         userRollupFee = 0;
21:         gasFeeAddress = _address;
22:     }

182:     function updateSequencerAddress(address _sequencer) external {
183:         require(msg.sender == sequencer, "Only the sequencer can update sequencer address");
184:         sequencer = _sequencer;
185:     }
```

Description

BradMoonUESTC : The contract does not emit events for all significant operations. For example, in the `updateSequencerAddress` and `updateReSubmitterAddress` functions, important state variables are modified, but no events are emitted. This lack of event logging reduces the transparency of the contract's operations and makes it more difficult to monitor and verify the contract's behavior.

The setFeeAddress function in the BVM_EigenDataLayrFee contract allows the contract owner to change the gasFeeAddress. This is a significant state change as it alters the recipient address for gas fees. However, the function does not emit an event to log this change. This lack of event emission can reduce the transparency of the contract's operations and make it more difficult for off-chain services or users to track changes in the contract's state.

Recommendation

BradMoonUESTC : It is recommended to emit events for all significant operations in the contract. This includes any operation that modifies the state of the contract. By emitting events, it becomes easier for external observers to monitor the contract's behavior and verify that it is functioning correctly.

Client Response

Fixed.This issue has been fixed as follows.

```
    event RollupFeeHistory(uint256 l2Block, uint256 userRollupFee);
    event FeeAddressUpdated(address oldFeeAddress, address newFeeAddress);
    function setRollupFee(uint256 _l2Block, uint256 _userRollupFee) public onlyGasFee {
        userRollupFee = _userRollupFee;
        emit RollupFeeHistory(_l2Block, _userRollupFee);
    }

    function setFeeAddress(address _address) public onlyOwner {
        require(_address != address(0), "setFeeAddress: address is the zero address");
        address oldGasFeeAddress = gasFeeAddress;
        gasFeeAddress = _address;
        emit FeeAddressUpdated(oldGasFeeAddress, gasFeeAddress);
    }
```

MTC-59:Missing parameter check in `L1StandardBridge::_initiateERC20Deposit`

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	BradMoonUESTC

Code Reference

- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L177-L185

```
177:    function _initiateERC20Deposit(
178:        address _l1Token,
179:        address _l2Token,
180:        address _from,
181:        address _to,
182:        uint256 _amount,
183:        uint32 _l2Gas,
184:        bytes calldata _data
185:    ) internal {
```

Description

BradMoonUESTC : The `_initiateERC20Deposit` function does not check whether the `_amount` parameter is greater than zero. This lack of validation may allow users to execute this function with an `_amount` of zero, which, while not harmful, is probably not intended functionality and would consume unnecessary gas.

Recommendation

BradMoonUESTC : Implement a requirement check to ensure the `_amount` parameter is greater than zero. A simple `require(_amount > 0, "Deposit amount must be greater than 0");` at the beginning of the function could prevent this potential misuse and avoid unnecessary gas consumption.

Client Response

Acknowledged.We will address it in our future updates.

MTC-60:Potential information leakage through logs

Category	Severity	Client Response	Contributor
Logical	Informational	Acknowledged	BradMoonUESTC

Code Reference

- code/mt-batcher/services/common/crypto.go#L94-L112

```
94:func ParseWalletPrivKeyAndContractAddr(name string, mnemonic string, hdPath string, privKeyStr st
ring, contractAddrStr string) (*ecdsa.PrivateKey, common.Address, error) {
95:
96:     privKey, err := GetConfiguredPrivateKey(mnemonic, hdPath, privKeyStr)
97:     if err != nil {
98:             return nil, common.Address{}, err
99:     }
100:
101:    contractAddress, err := ParseAddress(contractAddrStr)
102:    if err != nil {
103:            return nil, common.Address{}, err
104:    }
105:
106:    walletAddress := crypto.PubkeyToAddress(privKey.PublicKey)
107:
108:    log.Info(name+" wallet params parsed successfully", "wallet_address",
109:            walletAddress, "contract_address", contractAddress)
110:
111:    return privKey, contractAddress, nil
112:}
```

Description

BradMoonUESTC : The `ParseWalletPrivKeyAndContractAddr` function logs wallet addresses and contract addresses. Even though these data do not contain private keys, they could provide attackers with insight into the system's behavior or be associated with specific users.

Recommendation

BradMoonUESTC : Consider the risks before logging sensitive or potentially identifiable information. If such logging is necessary for debugging or other purposes, ensure the logs are properly secured and access is restricted. Consider using log redaction techniques to remove or obfuscate sensitive data.

Client Response

Acknowledged.It is very rare that both mnemonic and private key are provided. If it happens, the caller might make a critical mistake. The derived private key from mnemonic might be different from the provided private key. In cases where a default private key is returned without an error, there is a risk of mistakenly setting the private key, as the default one may not be the intended one. It's more prudent to return an error to remind the caller to be mindful of his/her private key.

MTC-61:Predeployed wrapped token with incorrect metadata

Category	Severity	Client Response	Contributor
Logical	Informational	Declined	iczc

Code Reference

- code/packages/contracts/contracts/L2/predeploys/WETH9.sol#L19-L20

```
19:     string public name     = "Wrapped Ether";
20:     string public symbol   = "WETH";
```

Description

iczc : Since mantle has replaced the native coin from ETH to BIT, however, the official predeployed wrapped token's symbol is still WETH even if this is not the real erc20 version ETH. This leads to malicious users can add this incorrect WETH to the dex pool and spoof users to buy it at the wrong expensive price.

Recommendation

iczc : Fix the wrapped token metadata to match the native coin.

Client Response

Declined.This contract is not associated with $BIT/$MNT or any native token.

MTC-62:Redundant code:Duplicate function in `BVM_EigenDataLayrChain`

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	Hupixiong3

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L128-L140

```
128:    function unavailableFraudProofAddress(address _address) external {
129:        require(msg.sender == sequencer, "Only the sequencer can remove fraud proof address");
130:        fraudProofWhitelist[_address] = false;
131:    }
132:
133:    /**
134:    * @notice remove fraud proof address
135:    * @param _address for fraud proof
136:    */
137:    function removeFraudProofAddress(address _address) external {
138:        require(msg.sender == sequencer, "Only the sequencer can remove fraud proof address");
139:        delete fraudProofWhitelist[_address];
140:    }
```

Description

Hupixiong3 : The functions unavailableFraudProofAddress() and removeFraudProofAddress() perform the same functionality. Using only removeFraudProofAddress() can keep the code concise, and removeFraudProofAddress() will return gas.

Recommendation

Hupixiong3 : Using only removeFraudProofAddress()

Client Response

Acknowledged.We will address it in our future updates.

MTC-63:Redundant code:Redundant check

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	iczc, Hupixiong3

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L124-L128
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L125-L128
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L244-L248
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L245-L248
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L321-L325

```
124:        // slashing params check
125:        for (uint256 i = 0; i < 2; i++) {
126:            require(slashAmount[i] > 0, "have not set the slash amount");
127:            require(exIncome[i] > 0, "have not set the extra income amount");
128:        }

125:        for (uint256 i = 0; i < 2; i++) {
126:            require(slashAmount[i] > 0, "have not set the slash amount");
127:            require(exIncome[i] > 0, "have not set the extra income amount");
128:        }

244:        // slashing params check
245:        for (uint256 i = 0; i < 2; i++) {
246:            require(slashAmount[i] > 0, "have not set the slash amount");
247:            require(exIncome[i] > 0, "have not set the extra income amount");
248:        }

245:        for (uint256 i = 0; i < 2; i++) {
246:            require(slashAmount[i] > 0, "have not set the slash amount");
247:            require(exIncome[i] > 0, "have not set the extra income amount");
248:        }

321:        // slashing params check
322:        for (uint256 i = 0; i < 2; i++) {
323:            require(slashAmount[i] > 0, "have not set the slash amount");
324:            require(exIncome[i] > 0, "have not set the extra income amount");
325:        }
```

Description

iczc : There are three duplicate slashing params check codes in TssStakingSlashing.
Hupixiong3 : In the staking and slash functions, relying on the condition slashAmount[i] and exIncome[i] being greater than 0 to determine if the slash amount and extra income amount are set is unnecessary. It is sufficient to use a boolean variable to record whether the setSlashingParams function has been called, as this function already includes the corresponding checks.

Recommendation

iczc : Using modifier to replace duplicate check.
Hupixiong3 : Translation: Use a boolean variable to record whether the setSlashingParams function has been called, remove redundant value checks, and add a check to verify if the boolean variable is true.

Client Response

Fixed. Please refer to the fix as follows.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L75

```
  isSetParam
  require(isSetParam,"have not set the slash amount");
```

MTC-64:Redundant code:Redundant initialization assignment

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	Hupixiong3

Code Reference

- code/packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol#L60
- code/packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol#L87

```
60:    address internal xDomainMsgSender = Lib_DefaultValues.DEFAULT_XDOMAIN_SENDER;

87:        xDomainMsgSender = Lib_DefaultValues.DEFAULT_XDOMAIN_SENDER;
```

Description

Hupixiong3 : There are redundant initialization assignments in the xDomainMsgSender variable.

Recommendation

Hupixiong3 : Keep the initialization only once.

Client Response

Acknowledged. We will address it in our future updates.

MTC-65:Redundant code:Unnecessary check

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	Hupixiong3

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L319

```
319:         require(_index < activeTssMembers.length, "index out of bound");
```

Description

Hupixiong3 : The comparison check between _index and activeTssMembers.length in the removeActiveTssMembers function is unnecessary. The values of _index and activeTssMembers.length are passed from the upper loop structure, and the comparison condition will always be true.

Recommendation

Hupixiong3 : Remove the check

Client Response

Fixed.The redundant codes have been removed. Please refer to the fix as follows. https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L14

MTC-66:Redundant code:Unnecessary duplicate functions

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	Hupixiong3

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L373-L379

```
373:    function isEqual(bytes memory byteListA, bytes memory byteListB) public pure returns (bool)
{
374:        if (byteListA.length != byteListB.length) return false;
375:        for (uint256 i = 0; i < byteListA.length; i++) {
376:            if (byteListA[i] != byteListB[i]) return false;
377:        }
378:        return true;
379:    }
```

Description

Hupixiong3 : The existence of the isEqual function in both the TssGroupManager and TssStakingSlashing contracts is unnecessary. You can refer to the publicKeyToAddress function in the TssGroupManager contract when calling it from the TssStakingSlashing contract.

Recommendation

Hupixiong3 : To remove the isEqual function from the TssStakingSlashing contract and instead call the isEqual function from the TssGroupManager contract.

Client Response

Fixed.We have removed the isEqual function in the TssStakingSlashing.sol contract. The latest version of TssGroupManager.sol has addressed the existing issues.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L336

```
    function _isEqual(bytes memory byteListA, bytes memory byteListB) private pure returns (bool) {
        if (byteListA.length != byteListB.length) return false;
        for (uint256 i = 0; i < byteListA.length; i++) {
            if (byteListA[i] != byteListB[i]) return false;
        }
        return true;
    }
```

MTC-67:Unused predeploy whitelist contract

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	iczc

Code Reference

- code/packages/contracts/contracts/L2/predeploys/BVM_DeployerWhitelist.sol#L11

```
11:contract BVM_DeployerWhitelist {
```

Description

iczc : The mantle's l2geth has removed the whitelist feature for deployment contracts, but the predeploy contract used to configure the whitelist address is still retained.

Recommendation

iczc : Remove the `BVM_DeployerWhitelist.sol`.

Client Response

Acknowledged. The deprecated contract will be removed in the future.

MTC-68:Use `bool` type variable to control the switch in BVM_GasPriceOracle

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	iczc

Code Reference

- code/packages/contracts/contracts/L2/predeploys/BVM_GasPriceOracle.sol#L35-L37
- code/packages/contracts/contracts/L2/predeploys/BVM_GasPriceOracle.sol#L56-L62

```
35:     uint256    isBurning;
36:     // Switch controls whether charge L2 GasFee
37:     uint256    public charge;

56:     modifier checkValue(uint256 value) {
57:         require(
58:             value == 0 || value == 1,
59:             "invalid value,must be 0 or 1"
60:         );
61:         _;
62:     }
```

Description

iczc : The switch control variables `isBurning` and `charge` in BVM_GasPriceOracle currently use uint 1 and 0, which requires additional parameter checks and brings more possibilities of setting the wrong switch status.

Recommendation

iczc : Use bool type instead of uint.

Client Response

Acknowledged. This is a contract optimization issue that we will address in our future updates.

MTL-1:A malicious user can permanently lock staker's money in `Rollup` contracts

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	biakia

Code Reference

- code/packages/contracts/contracts/L1/fraud-proof/Rollup.sol#L197-L208

```
197:    function removeStake(address stakerAddress) external override {
198:        requireStaked(stakerAddress);
199:        // Require that staker is staked on a confirmed assertion.
200:        Staker storage staker = stakers[stakerAddress];
201:        if (staker.assertionID > lastConfirmedAssertionID) {
202:            revert("StakedOnUnconfirmedAssertion");
203:        }
204:        deleteStaker(stakerAddress);
205:        // Note: we don't need to modify assertion state because you can only unstake from a con
firmed assertion.
206:        (bool success,) = stakerAddress.call{value: staker.amountStaked}("");
207:        if (!success) revert("TransferFailed");
208:    }
```

Description

biakia : In contract `Rollup`, anyone can call `removeStake` to remove a staker, as long as the staker's current assertion is confirmed:

```
function removeStake(address stakerAddress) external override {
        requireStaked(stakerAddress);
        // Require that staker is staked on a confirmed assertion.
        Staker storage staker = stakers[stakerAddress];
        if (staker.assertionID > lastConfirmedAssertionID) {
            revert("StakedOnUnconfirmedAssertion");
        }
        deleteStaker(stakerAddress);
        // Note: we don't need to modify assertion state because you can only unstake from a confirm
ed assertion.
        (bool success,) = stakerAddress.call{value: staker.amountStaked}("");
        if (!success) revert("TransferFailed");
    }
```

It first gets the data from the `stakers` map, and it is important to note that the variable here is defined as `storage`:

```
Staker storage staker = stakers[stakerAddress];
```

and then delete it:

```
deleteStaker(stakerAddress);

function deleteStaker(address stakerAddress) private {
        numStakers--;
        delete stakers[stakerAddress];
 }
```

Finally, it will use `staker.amountStaked` as an argument to pay back the money, but since `staker` is a `storage` variable and has been deleted, `staker.amountStaked` will be 0, and eventually, no money is paid back to the user:

```
(bool success,) = stakerAddress.call{value: staker.amountStaked}("");
 if (!success) revert("TransferFailed");
```

After calling the `removeStake` function, the staker's money will be permanently locked in the current contract. A malicious user can exploit this vulnerability to lock all the staker's money in the current contract.

Recommendation

biakia : Consider caching the `amountStaked` before deleting it:

```
function removeStake(address stakerAddress) external override {
        requireStaked(stakerAddress);
        // Require that staker is staked on a confirmed assertion.
        Staker storage staker = stakers[stakerAddress];
        if (staker.assertionID > lastConfirmedAssertionID) {
            revert("StakedOnUnconfirmedAssertion");
        }
        uint256 amountToSent = staker.amountStaked;
        deleteStaker(stakerAddress);
        // Note: we don't need to modify assertion state because you can only unstake from a confirm
ed assertion.
        (bool success,) = stakerAddress.call{value: amountToSent}("");
        if (!success) revert("TransferFailed");
    }
```

Client Response

Fixed.The issue has been fixed in PR https://github.com/mantlenetworkio/mantle/pull/858. Only the staker has the authority to access removeStake. Furthermore, if removeStake is executed, the staker's token will be promptly refunded, eliminating the possibility of anyone locking the staker's funds.

```
196 219       /// @inheritdoc IRollup
197       -   function removeStake(address stakerAddress) external override {
      220 +   function removeStake(address stakerAddress) onlyOwner external override {
198 221           requireStaked(stakerAddress);
199 222           // Require that staker is staked on a confirmed assertion.
200 223           Staker storage staker = stakers[stakerAddress];
201 224           if (staker.assertionID > lastConfirmedAssertionID) {
202 225               revert("StakedOnUnconfirmedAssertion");
203 226           }
204 227           deleteStaker(stakerAddress);
205       -       // Note: we don't need to modify assertion state because you can only unstake from a confirmed assertion.
206       -       (bool success,) = stakerAddress.call{value: staker.amountStaked}("");
207       -       if (!success) revert("TransferFailed");
      228 +       // send erc20 token to user
      229 +       require(
      230 +           IERC20(stakeToken).transfer(stakerAddress, staker.amountStaked),
      231 +           "transfer erc20 token failed"
      232 +       );
208 233       }
```

MTL-2:An invalid assertion can also be confirmed

Category	Severity	Client Response	Contributor
Logical	Critical	Mitigated	biakia

Code Reference

- code/packages/contracts/contracts/L1/fraud-proof/Rollup.sol#L333-L366

```
333:     function confirmFirstUnresolvedAssertion() external override {
334:         if (lastResolvedAssertionID >= lastCreatedAssertionID) {
335:             revert("NoUnresolvedAssertion");
336:         }
337:
338:         // (1) there is at least one staker, and
339:         if (numStakers <= 0) revert("NoStaker");
340:
341:         uint256 lastUnresolvedID = lastResolvedAssertionID + 1;
342:
343:         // (2) challenge period has passed
344:         if (block.timestamp < assertions.getDeadline(lastUnresolvedID)) {
345:             revert("ChallengePeriodPending");
346:         }
347:
348:         // (3) predecessor has been confirmed
349:         if (assertions.getParentID(lastUnresolvedID) != lastConfirmedAssertionID) {
350:             revert("InvalidParent");
351:         }
352:
353:         // Remove old zombies
354:         // removeOldZombies();
355:
356:         // (4) all stakers are staked on the block.
357:         // if (assertions.getNumStakers(lastUnresolvedID) != numStakers) {
358:         //     revert("NotAllStaked");
359:         // }
360:
361:         // Confirm assertion.
362:         // assertions.deleteAssertion(lastConfirmedAssertionID);
363:         lastResolvedAssertionID++;
364:         lastConfirmedAssertionID = lastResolvedAssertionID;
365:         emit AssertionConfirmed(lastResolvedAssertionID);
366:     }
```

Description

biakia : In contract `Rollup`, assertions can be confirmed by calling `confirmFirstUnresolvedAssertion`:

```
/// @inheritdoc IRollup
    function confirmFirstUnresolvedAssertion() external override {
        if (lastResolvedAssertionID >= lastCreatedAssertionID) {
            revert("NoUnresolvedAssertion");
        }

        // (1) there is at least one staker, and
        if (numStakers <= 0) revert("NoStaker");

        uint256 lastUnresolvedID = lastResolvedAssertionID + 1;

        // (2) challenge period has passed
        if (block.timestamp < assertions.getDeadline(lastUnresolvedID)) {
            revert("ChallengePeriodPending");
        }

        // (3) predecessor has been confirmed
        if (assertions.getParentID(lastUnresolvedID) != lastConfirmedAssertionID) {
            revert("InvalidParent");
        }

        // Remove old zombies
        // removeOldZombies();

        // (4) all stakers are staked on the block.
        // if (assertions.getNumStakers(lastUnresolvedID) != numStakers) {
        //     revert("NotAllStaked");
        // }

        // Confirm assertion.
        // assertions.deleteAssertion(lastConfirmedAssertionID);
        lastResolvedAssertionID++;
        lastConfirmedAssertionID = lastResolvedAssertionID;
        emit AssertionConfirmed(lastResolvedAssertionID);
    }
```

The check on whether all stakers are staked on the same block are commented:

```
        // (4) all stakers are staked on the block.
        // if (assertions.getNumStakers(lastUnresolvedID) != numStakers) {
        //     revert("NotAllStaked");
        // }
```

This will allow the invalid assertion be confirmed. Consider the following case:

```
// [1] <- [3]           | [1] is last confirmed assertion, [3] is an valid assertion
//  ^---- [2]           | [2] is firstUnresolved assertion which is invalid
```

We say that `[3]` is an assertion that all honest stakers are staked on and `[2]` is a malicious staker's assertion. The malicious staker starts an challenge with `[3]` and failed. At this time, all honest stakers are still staked on `[3]`, only the malicious staker is staked on `[2]`. The malicious staker can call `confirmFirstUnresolvedAssertion` to confirm `[2]` because the challenge period of `[2]` has passed, the predecessor of `[2]` is `[1]` which has been confirmed, and the check on whether all stakers are staked on the block has been commented. At last, the invalid assertion `[2]` is confirmed.

Recommendation

biakia : Consider adding a check on whether all stakers are staked on the block:

```
        // (4) all stakers are staked on the block.
        if (assertions.getNumStakers(lastUnresolvedID)  != countStakedZombies(lastUnresolvedID) + nu
mStakers) {
            revert NotAllStaked();
        }
```

Client Response

Mitigated. No slashing mechanism is in place to penalize offline validators that could potentially disrupt the confirmation process.

MTL-3:Potential serialization errors in `batch-submitter` module

Category	Severity	Client Response	Contributor
Logical	Critical	Acknowledged	biakia

Code Reference

- code/batch-submitter/drivers/sequencer/encoding.go#L341-L422

```
341:func (p *AppendSequencerBatchParams) Read(r io.Reader) error {
342:    if err := readUint64(r, &p.ShouldStartAtElement, 5); err != nil {
343:            return err
344:    }
345:    if err := readUint64(r, &p.TotalElementsToAppend, 3); err != nil {
346:            return err
347:    }
348:
349:    // Read number of contexts and deserialize each one.
350:    var numContexts uint64
351:    if err := readUint64(r, &numContexts, 3); err != nil {
352:            return err
353:    }
354:
355:    // Assume that it is a legacy batch at first, this will be overwrritten if
356:    // we detect a marker context.
357:    var batchType = BatchTypeLegacy
358:    // Ensure that contexts is never nil
359:    p.Contexts = make([]BatchContext, 0)
360:    for i := uint64(0); i < numContexts; i++ {
361:            var batchContext BatchContext
362:            if err := batchContext.Read(r); err != nil {
363:                    return err
364:            }
365:
366:            if i == 0 && batchContext.IsMarkerContext() {
367:                    batchType = batchContext.MarkerBatchType()
368:                    continue
369:            }
370:
371:            p.Contexts = append(p.Contexts, batchContext)
372:    }
373:
374:    // Define a closure to clean up the reader used by the specified encoding.
375:    var closeReader func() error
376:    switch batchType {
377:
378:    // The legacy serialization does not require clsing, so we instatiate a
379:    // dummy closure.
380:    case BatchTypeLegacy:
381:            closeReader = func() error { return nil }
382:
```

```
383:    // The zlib serialization requires decompression before reading the
384:    // plaintext bytes, and also requires proper cleanup.
385:    case BatchTypeZlib:
386:            zr, err := zlib.NewReader(r)
387:            if err != nil {
388:                    return err
389:            }
390:            closeReader = zr.Close
391:
392:            r = bufio.NewReader(zr)
393:    }
394:
395:    // Deserialize any transactions. Since the number of txs is omitted
396:    // from the encoding, loop until the stream is consumed.
397:    for {
398:            var txLen uint64
399:            err := readUint64(r, &txLen, TxLenSize)
400:            // Getting an EOF when reading the txLen expected for a cleanly
401:            // encoded object. Silence the error and return success if
402:            // the batch is well formed.
403:            if err == io.EOF {
404:                    if len(p.Contexts) == 0 && len(p.Txs) != 0 {
405:                            return ErrMalformedBatch
406:                    }
407:                    if len(p.Txs) == 0 && len(p.Contexts) != 0 {
408:                            return ErrMalformedBatch
409:                    }
410:                    return closeReader()
411:            } else if err != nil {
412:                    return err
413:            }
414:
415:            tx := new(l2types.Transaction)
416:            if err := tx.DecodeRLP(l2rlp.NewStream(r, txLen)); err != nil {
417:                    return err
418:            }
419:
420:            p.Txs = append(p.Txs, NewCachedTx(tx))
421:    }
422:}
```

Description

biakia : In file `encoding.go`, the `AppendSequencerBatchParams` holds the raw data required to submit a batch of L2 txs to L1 CTC. When encoding the `AppendSequencerBatchParams`, the function `Serialize` will be called:

```
func (p *AppendSequencerBatchParams) Serialize(
        batchType BatchType,
        l2BlockNumber *big.Int,
        upgradeBlock *big.Int,
) ([]byte, error) {
        var buf bytes.Buffer
        if l2BlockNumber.Cmp(upgradeBlock) > 0 {
                if err := p.WriteNoTxn(&buf, batchType); err != nil {
                        return nil, err
                }
        } else {
                if err := p.Write(&buf, batchType); err != nil {
                        return nil, err
                }
        }
        return buf.Bytes(), nil
}
```

If `l2BlockNumber > upgradeBlock`, it will call `WriteNoTxn` function:

```
func (p *AppendSequencerBatchParams) WriteNoTxn(
        w *bytes.Buffer,
        batchType BatchType,
) error {
        _ = writeUint64(w, p.ShouldStartAtElement, 5)
        _ = writeUint64(w, p.TotalElementsToAppend, 3)

        // There must be contexts if there are transactions
        if len(p.Contexts) == 0 && len(p.Txs) != 0 {
                return ErrMalformedBatch
        }

        // There must be transactions if there are contexts
        if len(p.Txs) == 0 && len(p.Contexts) != 0 {
                return ErrMalformedBatch
        }

        // copy the contexts as to not malleate the struct
        // when it is a typed batch
        contexts := make([]BatchContext, 0, len(p.Contexts)+1)
        // Add the marker context, if any, for non-legacy encodings.
        markerContext := batchType.MarkerContext()
        if markerContext != nil {
                contexts = append(contexts, *markerContext)
        }
        contexts = append(contexts, p.Contexts...)

        // Write number of contexts followed by each fixed-size BatchContext.
        _ = writeUint64(w, uint64(len(contexts)), 3)
        for _, context := range contexts {
                context.Write(w)
        }
        return nil
}
```

It only writes number of contexts to the bytes stream and ignores transactions. When decoding `AppendSequencerBatchParams`, the function `Read` will be called:

```
func (p *AppendSequencerBatchParams) Read(r io.Reader) error {
        if err := readUint64(r, &p.ShouldStartAtElement, 5); err != nil {
                return err
        }
        if err := readUint64(r, &p.TotalElementsToAppend, 3); err != nil {
                return err
        }

        // Read number of contexts and deserialize each one.
        var numContexts uint64
        if err := readUint64(r, &numContexts, 3); err != nil {
                return err
        }

        // Assume that it is a legacy batch at first, this will be overwrritten if
        // we detect a marker context.
        var batchType = BatchTypeLegacy
        // Ensure that contexts is never nil
        p.Contexts = make([]BatchContext, 0)
        for i := uint64(0); i < numContexts; i++ {
                var batchContext BatchContext
                if err := batchContext.Read(r); err != nil {
                        return err
                }

                if i == 0 && batchContext.IsMarkerContext() {
                        batchType = batchContext.MarkerBatchType()
                        continue
                }

                p.Contexts = append(p.Contexts, batchContext)
        }

        // Define a closure to clean up the reader used by the specified encoding.
        var closeReader func() error
        switch batchType {

        // The legacy serialization does not require clsing, so we instatiate a
        // dummy closure.
        case BatchTypeLegacy:
                closeReader = func() error { return nil }

        // The zlib serialization requires decompression before reading the
```

```
// plaintext bytes, and also requires proper cleanup.
        case BatchTypeZlib:
                zr, err := zlib.NewReader(r)
                if err != nil {
                        return err
                }
                closeReader = zr.Close

                r = bufio.NewReader(zr)
        }

        // Deserialize any transactions. Since the number of txs is omitted
        // from the encoding, loop until the stream is consumed.
        for {
                var txLen uint64
                err := readUint64(r, &txLen, TxLenSize)
                // Getting an EOF when reading the txLen expected for a cleanly
                // encoded object. Silence the error and return success if
                // the batch is well formed.
                if err == io.EOF {
                        if len(p.Contexts) == 0 && len(p.Txs) != 0 {
                                return ErrMalformedBatch
                        }
                        if len(p.Txs) == 0 && len(p.Contexts) != 0 {
                                return ErrMalformedBatch
                        }
                        return closeReader()
                } else if err != nil {
                        return err
                }

                tx := new(l2types.Transaction)
                if err := tx.DecodeRLP(l2rlp.NewStream(r, txLen)); err != nil {
                        return err
                }

                p.Txs = append(p.Txs, NewCachedTx(tx))
        }
}
```

It will first decode `ShouldStartAtElement` and `TotalElementsToAppend`, and then decode the contexts. At last, it will try to decode txs:

```
for {
                var txLen uint64
                err := readUint64(r, &txLen, TxLenSize)
                // Getting an EOF when reading the txLen expected for a cleanly
                // encoded object. Silence the error and return success if
                // the batch is well formed.
                if err == io.EOF {
                        if len(p.Contexts) == 0 && len(p.Txs) != 0 {
                                return ErrMalformedBatch
                        }
                        if len(p.Txs) == 0 && len(p.Contexts) != 0 {
                                return ErrMalformedBatch
                        }
                        return closeReader()
                } else if err != nil {
                        return err
                }

                tx := new(l2types.Transaction)
                if err := tx.DecodeRLP(l2rlp.NewStream(r, txLen)); err != nil {
                        return err
                }

                p.Txs = append(p.Txs, NewCachedTx(tx))
        }
```

Since the function `WriteNoTxn` didn't write txs to the bytes stream, we will read a `EOF` error and go into the if clause:

```
if err == io.EOF {
        if len(p.Contexts) == 0 && len(p.Txs) != 0 {
                return ErrMalformedBatch
        }
        if len(p.Txs) == 0 && len(p.Contexts) != 0 {
                return ErrMalformedBatch
        }
        return closeReader()
}
```

Since there are contexts with no txs, the `len(p.Contexts)` will be greater than 0 but the `len(p.Txs)` will be equal to 0, which meets the second if condition `if len(p.Txs) == 0 && len(p.Contexts) != 0`. Finally, it will return `ErrMalformedBatch`. The `Read` function call will fail due to the `ErrMalformedBatch` error.

Recommendation

biakia : Consider redesigning the logic of the function `Read`.

Client Response

Acknowledged. CTC contract handles transaction counts, eliminating the need for data compression and encoding within the batch-submitter. The data undergoes encoding and compression through mt-batcher before being submitted to MantleDA, not CTC. We intend to remove this code segment in future updates.

MTL-4:There is a risk of funds being locked

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	yekong

Code Reference

- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/governance/Governor.sol#L304-L318

```
304:     function execute(uint256 proposalId) external payable {
305:         require(
306:             state(proposalId) == ProposalState.Queued,
307:             "RepositoryGovernance::execute: proposal can only be executed if it is queued"
308:         );
309:         Proposal storage proposal = proposals[proposalId];
310:         proposal.executed = true;
311:         for (uint256 i = 0; i < proposal.targets.length; ++i) {
312:             // slither-disable-next-line unused-return
313:             timelock.executeTransaction{value: proposal.values[i]}(
314:                 proposal.targets[i], proposal.values[i], proposal.signatures[i], proposal.callda
tas[i], proposal.eta
315:             );
316:         }
317:         emit ProposalExecuted(proposalId);
318:     }
```

Description

yekong : In the `execute` function of the `Governor` contract, the amount of ETH transferred is passed as proposal.values[i] to the timelock.executeTransaction function. However, the received ETH by the `execute` function (i.e., msg.value) is not handled at all within the function. If proposal.values[i] is inconsistent with msg.value, the transaction may fail or excess ether is left in the contract. In addition, in this contract, it seems that there is no function to extract ether

Recommendation

yekong : It is recommended to have a clear logic to handle msg.value within your payable functions. And add a function to extract ether in the contract to prevent ether from being locked

Client Response

Fixed. Please note this code has been deprecated.

MTL-5: `BASEFEE` opcode will fail to execute

Category	Severity	Client Response	Contributor
Language Specific	Critical	Acknowledged	Secure3

Code Reference

- code/l2geth/core/vm/runtime/env.go#L25-L41
- code/l2geth/core/vm/runtime/runtime.go#L34-L49

```
25:func NewEnv(cfg *Config) *vm.EVM {
26:     context := vm.Context{
27:             CanTransfer: core.CanTransfer,
28:             Transfer:    core.Transfer,
29:             GetHash:     func(uint64) common.Hash { return common.Hash{} },
30:
31:             Origin:      cfg.Origin,
32:             Coinbase:    cfg.Coinbase,
33:             BlockNumber: cfg.BlockNumber,
34:             Time:        cfg.Time,
35:             Difficulty:  cfg.Difficulty,
36:             GasLimit:    cfg.GasLimit,
37:             GasPrice:    cfg.GasPrice,
38:     }
39:
40:     return vm.NewEVM(context, cfg.State, cfg.ChainConfig, cfg.EVMConfig)
41:}

34:type Config struct {
35:     ChainConfig *params.ChainConfig
36:     Difficulty  *big.Int
37:     Origin      common.Address
38:     Coinbase    common.Address
39:     BlockNumber *big.Int
40:     Time        *big.Int
41:     GasLimit    uint64
42:     GasPrice    *big.Int
43:     Value       *big.Int
44:     Debug       bool
45:     EVMConfig   vm.Config
46:
47:     State     *state.StateDB
48:     GetHashFn func(n uint64) common.Hash
49:}
```

Description

Secure3 : In `code/l2geth/core/vm/runtime/env.go` it define a function `NewEnv` that will be called when try to execute some evm opcodes.However it is missing to define `basefee` , so when a contract has a `basefee` opcode, it will fail to execute Consider below POC contract

```
pragma solidity ^0.8.0;

contract TEST {


    event Log(string , uint256);

    function test() public returns(uint256){
       uint256 x;
        assembly{
            x := number()
        }
        emit Log("number()", x);
        return x;

   }

   function test2() public returns(uint256){
        uint256 x;
        assembly{
            x := basefee()
        }
        emit Log("basefee()", x);
        return x;
   }

}
```

the `test2()` failed to execute in mantle testnet ,here is the tx hash `0x3a7c892270d780cac71d9233dcff7e061feacf6732151c0548d60bb0d4f9458f`

```
geth attach https://rpc.testnet.mantle.xyz/

> eth.getTransactionReceipt("0x3a7c892270d780cac71d9233dcff7e061feacf6732151c0548d60bb0d4f9458f")
{
  blockHash: "0x8f6a0390d925c9b76db91f62f84c5fd6cc04f4b1acce6adc9a744c47c38a03bb",
  blockNumber: 6573415,
  contractAddress: null,
  cumulativeGasUsed: 3000000,
  from: "0x21a826e25938cd46417833ce6de02c584567f6c0",
  gasUsed: 3000000,
  l1Fee: "0x19f6a92a1d8d5100",
  l1FeeScalar: "1.5",
  l1GasPrice: "0x102de6f14f9a0",
  l1GasUsed: "0x111e",
  logs: [],
  logsBloom: "0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",
  status: "0x0",
  to: "0x61be44175ee82a6681b2ebb2c78e089a0551c4a1",
  transactionHash: "0x3a7c892270d780cac71d9233dcff7e061feacf6732151c0548d60bb0d4f9458f",
```

Recommendation

Secure3 : define `basefee` in `NewEnv` function in `code/l2geth/core/vm/runtime/env.go` and add definition in `Config` struct in `code/l2geth/core/vm/runtime/runtime.go` as well

Consider below fix:

```
func NewEnv(cfg *Config) *vm.EVM {
        txContext := vm.TxContext{
                Origin:   cfg.Origin,
                GasPrice: cfg.GasPrice,
        }
        blockContext := vm.BlockContext{
                CanTransfer: core.CanTransfer,
                Transfer:    core.Transfer,
                GetHash:     cfg.GetHashFn,
                Coinbase:    cfg.Coinbase,
                BlockNumber: cfg.BlockNumber,
                Time:        cfg.Time,
                Difficulty:  cfg.Difficulty,
                GasLimit:    cfg.GasLimit,
                BaseFee:     cfg.BaseFee,
        }

        return vm.NewEVM(blockContext, txContext, cfg.State, cfg.ChainConfig, cfg.EVMConfig)
}

type Config struct {
        ChainConfig *params.ChainConfig
        Difficulty  *big.Int
        Origin      common.Address
        Coinbase    common.Address
        BlockNumber *big.Int
        Time        *big.Int
        GasLimit    uint64
        GasPrice    *big.Int
        Value       *big.Int
        Debug       bool
        EVMConfig   vm.Config
        BaseFee     *big.Int

        State     *state.StateDB
        GetHashFn func(n uint64) common.Hash
}
```

Client Response

Acknowledged. Currently we don't support EIP-1559, which means the BaseFee opcode is not available. We have plans to support EIP-1559 in the future.

MTL-6: modifier `onlyEOA()` can be bypassed via `depositETHTo()` function

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	Secure3

Code Reference

- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L93-L99

```
93:     function depositETHTo(
94:         address _to,
95:         uint32 _l2Gas,
96:         bytes calldata _data
97:     ) external payable {
98:         _initiateETHDeposit(msg.sender, _to, _l2Gas, _data);
99:     }
```

Description

Secure3 : In `code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol` `depositETH` function has a modifier which only allow `msg.sender` is an EOA, however `depositETHTo()` function lacks the check, both of them will eventually trace back to this function `_initiateETHDeposit()` which lacks the check as well. So if someone intend to bypass this modifier, they can just call `depositETHTo()` function, which is no different from `depositETH()` function.

Consider below POC contract

```
contract hack {
    function hack(address _to, uint32 _l2Gas, bytes calldata _data) external {
        L1StandardBridge(_addr).depositETHTo(address(_to), _l2Gas, _data);
    }
}
```

BTW, the problem also exists in `depositERC20To()` function.

Recommendation

Secure3 : Add a check at `depositETHTo()` function.

Consider below fix in the `sample.test()` function

```
function depositETHTo(
        address _to,
        uint32 _l2Gas,
        bytes calldata _data
    ) external payable onlyEOA{
        _initiateETHDeposit(msg.sender, _to, _l2Gas, _data);
    }
```

Client Response

Declined. The onlyEOA() modifier is not required for depositETH, depositETHTo, depositERC20, and depositERC20To functions. Currently depositETH and depositERC20 have this modifier, whereas depositETHTo and depositERC20To do not. We plan to maintain the current configuration as it doesn't have security implications.

MTL-7:BVM_EigenDataLayrChain.proveFraud uses a dangerous strict equality

Category	Severity	Client Response	Contributor
Code Style	Medium	Declined	Secure3

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L340
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L349

```
340:        require(rollupStore.status == RollupStoreStatus.COMMITTED && rollupStore.confirmAt > blo
ck.timestamp, "RollupStore must be committed and unconfirmed");

349:        require(searchData.metadata.globalDataStoreId == rollupStore.dataStoreId, "seachData's d
atastore id is not consistent with given rollup store");
```

Description

Secure3 : Use of strict equalities that can be easily manipulated by an attacker.

```
 require(rollupStore.status == RollupStoreStatus.COMMITTED && rollupStore.confirmAt > block.timestam
p, "RollupStore must be committed and unconfirmed");
require(searchData.metadata.globalDataStoreId == rollupStore.dataStoreId, "seachData's datastore id
is not consistent with given rollup store");
```

Recommendation

Secure3 : Don't use strict equality to determine if an account has enough Ether or tokens.

Consider below fix in the `sample.test()` function

```
val = newVal;
transfer call...
```

Client Response

Declined. We have tested but didn't identify any vulnerabilities or manipulation methods.

MTL-8:Division by Zero and Potential Incorrect Behavior Due to Missing Input Validation

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	yekong

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L301-L303

```
301:         remainder = extraAmount % tssNodes.length;
302:         // record the gain for tss nodes
303:         gain = (extraAmount - remainder) / tssNodes.length;
```

Description

yekong : The function transformDeposit is vulnerable to a division by zero error if the tssNodes array is empty. This would cause a runtime error in Solidity, causing the transaction to revert. This can happen in the following lines of code:

```
remainder = extraAmount % tssNodes.length;
gain = (extraAmount - remainder) / tssNodes.length;
```

In addition, this function may behave incorrectly when tssNodes[i] does not exist in the deposit mapping. If tssNodes[i] does not exist in the deposit mapping, adding gain to the default deposit struct can lead to incorrect behavior.

Recommendation

yekong : You should add an input validation to check if the tssNodes array is empty and add checks to ensure that tssNodes[i] exists in the deposit mapping. An example of such checks would be:

```
require(tssNodes.length > 0, "tssNodes array cannot be empty");
for (uint256 i = 0; i < tssNodes.length; i++) {
    require(deposits[tssNodes[i]] != address(0), "tssNode must exist in deposits");
}
```

Client Response

Fixed.Please note this code has been deprecated. Please review https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L277

```
//removed
remainder = extraAmount % tssNodes.length;
// record the gain for tss nodes
gain = (extraAmount - remainder) / tssNodes.length;
```

MTL-9:Function does not accept a return value, but `return` is used

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	yekong

Code Reference

- code/packages/contracts/contracts/L1/rollup/ChainStorageContainer.sol#L71

```
71:    function setGlobalMetadata(bytes27 _globalMetadata) public onlyOwner {
```

Description

yekong : The setGlobalMetadata function does not receive a return value, and the buffer.setExtraData function does not have a return value There is a critical misuse of the 'return' statement within the 'setGlobalMetadata' function in the smart contract. Despite being declared as a void return type, the function incorrectly attempts to return the output of 'buffer.setExtraData(_globalMetadata)'. This erroneous usage of 'return' in a non-returning function can lead to catastrophic implications and erratic behavior of the smart contract.

If this function is invoked by external parties, it could potentially lead to unpredictable outcomes. The contract may not perform as intended or expected, leading to loss of trust in the contract's functionality.

```
function setGlobalMetadata(bytes27 _globalMetadata) public onlyOwner {
        return buffer.setExtraData(_globalMetadata);
    }
```

Recommendation

yekong : The severity of this finding requires immediate attention. The return statement should be removed if 'setGlobalMetadata' is indeed intended not to return anything. The function should be written as follows:

```
function setGlobalMetadata(bytes27 _globalMetadata) public onlyOwner {
    buffer.setExtraData(_globalMetadata);
}
```

However, if the function 'setGlobalMetadata' needs to return the result of 'buffer.setExtraData(_globalMetadata)', then the return type should be specified in the function declaration appropriately, like so:

```
function setGlobalMetadata(bytes27 _globalMetadata) public onlyOwner returns (type) {
    return buffer.setExtraData(_globalMetadata);
}
```

In this case, 'type' should be replaced with the appropriate data type that 'buffer.setExtraData' returns.

Ensure to conduct thorough testing after making these changes to confirm that the function works as expected and does not introduce any new issues.

Client Response

Acknowledged. This is native code from Optimism and we haven't made any modifications to it. We will consider fixing it in the future as it has no security implications.

MTL-10:Incorrect returned value in `GetStateBatch` function

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	biakia

Code Reference

- code/tss/manager/store/state_batch.go#L16-L26

```
16:func (s *Storage) GetStateBatch(root [32]byte) (bool, index.StateBatchInfo) {
17:     bz, err := s.db.Get(getStateBatchKey(root), nil)
18:     if err != nil {
19:             return handleError2(index.StateBatchInfo{}, err)
20:     }
21:     var sbi index.StateBatchInfo
22:     if err = json.Unmarshal(bz, &sbi); err != nil {
23:             return true, index.StateBatchInfo{}
24:     }
25:     return true, sbi
26:}
```

Description

biakia : In module `observe.go`, the function `indexBatch` will call `GetStateBatch` to get the state batch with root:

```
func indexBatch(store StateBatchStore, stateBatchRoot [32]byte, batchIndex uint64) (retry bool, foun
d bool) {
        found, stateBatch := store.GetStateBatch(stateBatchRoot)
        if !found {
                log.Error("can not find the state batch with root, skip this batch", "root", hexuti
l.Encode(stateBatchRoot[:]))
                return false, found
        }
        stateBatch.BatchIndex = batchIndex
        if err := store.SetStateBatch(stateBatch); err != nil { // update stateBatch with index
                log.Error("failed to SetStateBatch with index", err)
                time.Sleep(2 * time.Second)
                return true, found
        }

        if err := store.IndexStateBatch(batchIndex, stateBatchRoot); err != nil {
                log.Error("failed to IndexStateBatch", err)
                time.Sleep(2 * time.Second)
                return true, found
        }
        return false, found
}
```

In the function `GetStateBatch`, it returns true and a default `StateBatchInfo` when parsing `StateBatchInfo` fails:

```
         var sbi index.StateBatchInfo
        if err = json.Unmarshal(bz, &sbi); err != nil {
                return true, index.StateBatchInfo{}
        }
```

This means that the variable `found` in `indexBatch` function will be true, and the following `if` clause will be ignored:

```
if !found {
                log.Error("can not find the state batch with root, skip this batch", "root", hexuti
l.Encode(stateBatchRoot[:]))
                return false, found
}
```

Finally, the default state batch will be updated in `indexBatch` function:

```
stateBatch.BatchIndex = batchIndex
if err := store.SetStateBatch(stateBatch); err != nil { // update stateBatch with index
                log.Error("failed to SetStateBatch with index", err)
                time.Sleep(2 * time.Second)
                return true, found
}
```

This is unreasonable to set a default state batch into the storage because a default state batch has neither `BatchRoot` nor `ElectionId`.

Recommendation

biakia : Consider returning false when parsing `StateBatchInfo` fails:

```
func (s *Storage) GetStateBatch(root [32]byte) (bool, index.StateBatchInfo) {
        bz, err := s.db.Get(getStateBatchKey(root), nil)
        if err != nil {
                return handleError2(index.StateBatchInfo{}, err)
        }
        var sbi index.StateBatchInfo
        if err = json.Unmarshal(bz, &sbi); err != nil {
                return false, index.StateBatchInfo{}
        }
        return true, sbi
}
```

Client Response

Fixed.Deprecated code has been removed. Please review https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/tss/manager/store/state_batch.go#L16C1-L26C2

```
func (s *Storage) GetStateBatch(root [32]byte) (bool, index.StateBatchInfo) {
        bz, err := s.db.Get(getStateBatchKey(root), nil)
        if err != nil {
                return handleError2(index.StateBatchInfo{}, err)
        }
        var sbi index.StateBatchInfo
        if err = json.Unmarshal(bz, &sbi); err != nil {
                return false, index.StateBatchInfo{}
        }
        return true, sbi
}
```

MTL-11:Loss of user assets in `_initiateETHDeposit` function

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	yekong

Code Reference

- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L130
- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L223

```
130:         sendCrossDomainMessage(l2TokenBridge, _l2Gas, message);

223:         sendCrossDomainMessage(l2TokenBridge, _l2Gas, message);
```

Description

yekong : If _l2Gas is not enough, IL2ERC20Bridge.finalizeDeposit may fail, resulting in loss of user assets

Recommendation

yekong : Due to the potential risk of loss of user assets, I recommend emphasizing this risk in the documentation, or prompting this risk in the user interface

Client Response

Acknowledged. Our official bridge and SDK automatically handle this situation as indicated in our tech documentation.

MTL-12:Potential Gas Wasting and Transfer Failure Handling in the claimReward function

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	yekong

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L91-L128

```
91:     function claimReward(uint256 _blockStartHeight, uint32 _length, uint256 _batchTime, address[]
calldata _tssMembers)
92:     external
93:     virtual
94:     onlyFromCrossDomainAccount(sccAddress)
95:     {
96:         if (IBVM_GasPriceOracle(bvmGasPriceOracleAddress).IsBurning() != 1) {
97:             claimRewardByBlock(_blockStartHeight, _length, _tssMembers);
98:             return;
99:         }
100:         uint256 sendAmount = 0;
101:         uint256 batchAmount = 0;
102:         uint256 accu = 0;
103:         // sendAmount
104:         if (lastBatchTime == 0) {
105:             lastBatchTime = _batchTime;
106:             return;
107:         }
108:         require(_batchTime > lastBatchTime,"args _batchTime must gther than last lastBatchTim
e");
109:         batchAmount = (_batchTime - lastBatchTime) * querySendAmountPerSecond() + dust;
110:         dust = 0;
111:         sendAmount = batchAmount.div(_tssMembers.length);
112:         for (uint256 j = 0; j < _tssMembers.length; j++) {
113:             address payable addr = payable(_tssMembers[j]);
114:             accu = accu.add(sendAmount);
115:             addr.transfer(sendAmount);
116:         }
117:         uint256 reserved = batchAmount.sub(accu);
118:         if (reserved > 0) {
119:             dust = dust.add(reserved);
120:         }
121:         emit DistributeTssReward(
122:             lastBatchTime,
123:             _batchTime,
124:             sendAmount,
125:             _tssMembers
126:         );
127:         lastBatchTime = _batchTime;
128:     }
```

Description

yekong : The claimReward function allows to distribute rewards among the members of _tssMembers. However, there are two issues to be noted:

Inefficient gas usage: In case of an empty _tssMembers array, the function does not prevent the execution of further operations which could waste gas unnecessarily.

Absence of failure handling: The transfer function used to send funds can fail due to a number of reasons. If it fails, the transaction will revert and this could lead to undesired effects especially when called in a loop. The contract does not handle this potential failure.

Recommendation

yekong : To prevent unnecessary gas usage, it is recommended to add a check at the beginning of the function to ensure that the _tssMembers array is not empty.

Consider replacing the transfer function with the call{value:...}("") pattern and add a check to ensure the transfer was successful. This would give you more control over the transaction, including error handling.

For example:

```
(bool success, ) = addr.call{value: sendAmount}("");
require(success, "Transfer failed");
```

Client Response

Declined. Please refer to the following for details about "inefficient gas usage".

https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol#L361C9-L361C119

```
address[] memory tssMembers = ITssGroupManager(resolve("Proxy__TSS_GroupManager")).getTssGroupUnJailMembers();
```

The ITssGroupManager smart contract will always have value after the network is live, making it impossible to be empty.

Regarding the recommendation to replace the transfer function with the call{value:...}("") pattern and add a check for successful transfer, we have already resolved the issue.

https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L199C1-L200C70

```
    (bool success, ) = claimer.call{ value: claimNumber }(new bytes(0));
    require(success, "TssReward claim: MNT transfer failed");
```

MTL-13:Potential Incorrect Behavior due to Default Value in Deposits Mapping

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	yekong

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L249

```
249:        bytes memory jailNodePubKey = deposits[message.jailNode].pubKey;
```

Description

yekong : In the slash function, the code bytes memory jailNodePubKey = deposits[message.jailNode].pubKey attempts to fetch a public key from the deposits mapping. If the message.jailNode does not exist in the deposits mapping, this will return a default value (an empty bytes object). This can lead to incorrect or unexpected behavior in subsequent operations, specifically when the memberJail or removeMember functions of the ITssGroupManager contract are invoked with an empty public key.

Recommendation

yekong : Before performing operations on deposits[message.jailNode].pubKey, you should check whether a deposit exists for message.jailNode. You could introduce a function depositExists(address _node) which returns true if a deposit exists for _node and false otherwise. In the slash function, add a require statement to ensure that a deposit exists:

```
require(depositExists(message.jailNode), "No deposit exists for the provided node");
```

Client Response

Fixed.Deprecated code has been removed. Please review https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L24

```
//removed
 bytes memory jailNodePubKey = deposits[message.jailNode].pubKey;
```

MTL-14:Potential Reward Overpayments in `TssRewardContract`

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	biakia

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L91-L128

```
91:     function claimReward(uint256 _blockStartHeight, uint32 _length, uint256 _batchTime, address[]
calldata _tssMembers)
92:     external
93:     virtual
94:     onlyFromCrossDomainAccount(sccAddress)
95:     {
96:         if (IBVM_GasPriceOracle(bvmGasPriceOracleAddress).IsBurning() != 1) {
97:             claimRewardByBlock(_blockStartHeight, _length, _tssMembers);
98:             return;
99:         }
100:         uint256 sendAmount = 0;
101:         uint256 batchAmount = 0;
102:         uint256 accu = 0;
103:         // sendAmount
104:         if (lastBatchTime == 0) {
105:             lastBatchTime = _batchTime;
106:             return;
107:         }
108:         require(_batchTime > lastBatchTime,"args _batchTime must gther than last lastBatchTim
e");
109:         batchAmount = (_batchTime - lastBatchTime) * querySendAmountPerSecond() + dust;
110:         dust = 0;
111:         sendAmount = batchAmount.div(_tssMembers.length);
112:         for (uint256 j = 0; j < _tssMembers.length; j++) {
113:             address payable addr = payable(_tssMembers[j]);
114:             accu = accu.add(sendAmount);
115:             addr.transfer(sendAmount);
116:         }
117:         uint256 reserved = batchAmount.sub(accu);
118:         if (reserved > 0) {
119:             dust = dust.add(reserved);
120:         }
121:         emit DistributeTssReward(
122:             lastBatchTime,
123:             _batchTime,
124:             sendAmount,
125:             _tssMembers
126:         );
127:         lastBatchTime = _batchTime;
128:     }
```

Description

biakia : In contract `TssRewardContract`, the `claimReward` function is used to distribute rewards to tss members. Two reward distribution mechanisms are implemented here, one is block-based and the other is time-based.
When it comes to the block-based mechanism, it will call `claimRewardByBlock` and return directly:

```
if (IBVM_GasPriceOracle(bvmGasPriceOracleAddress).IsBurning() != 1) {
            claimRewardByBlock(_blockStartHeight, _length, _tssMembers);
            return;
}
```

When it comes to the time-based mechanism, it will calculated the rewards based on the input param `_batchTime` and a local variable `lastBatchTime`:

```
      if (lastBatchTime == 0) {
            lastBatchTime = _batchTime;
            return;
        }
        require(_batchTime > lastBatchTime,"args _batchTime must gther than last lastBatchTime");
        batchAmount = (_batchTime - lastBatchTime) * querySendAmountPerSecond() + dust;
```

At last, it will update the `lastBatchTime`:

```
...
...
         emit DistributeTssReward(
            lastBatchTime,
            _batchTime,
            sendAmount,
            _tssMembers
        );
        lastBatchTime = _batchTime;
```

The issue here is that when it comes to the block-based mechanism, the `lastBatchTime` isn't updated. Consider the following case:

1. in blocktime 10000~12000, the mechanism is time-based, the final `lastBatchTime` will be 12000
2. in blocktime 12000~14000, the mechanism has been changed to block-based, the `lastBatchTime` is still 12000
3. in blocktime 140001, the mechanism has been changed to time-based, at this point, the calculation for `batchAmount` will be `(14001-12000)*querySendAmountPerSecond() + dust` and these rewards wii be distributed to the tss members. However in fact rewards between the blocktime 12,000 ~ 14,000 have been distributed already. So there will be reward overpayments.

Note: I'm not sure if the implementation is by design, but the issue still deserves attention.

Recommendation

biakia : Consider updating `lastBatchTime` when the mechanism is block-based:

```
if (IBVM_GasPriceOracle(bvmGasPriceOracleAddress).IsBurning() != 1) {
            if(_batchTime > lastBatchTime){
                    lastBatchTime = _batchTime;
            }
            claimRewardByBlock(_blockStartHeight, _length, _tssMembers);
            return;
}
```

Client Response

Fixed.We have made significant changes to this logic by removing the conditional statements and the claimRewardByBlock function. Please refer to the following for more details. https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L99

```
if (IBVM_GasPriceOracle(bvmGasPriceOracleAddress).IsBurning() != 1) {
            claimRewardByBlock(_blockStartHeight, _length, _tssMembers);
            return;
}
```

MTL-15:Potential Risk of Duplicate Entries in setTssGroupMember Function

Category	Severity	Client Response	Contributor
Logical	Medium	Mitigated	yekong

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L58-L85

```
58:     function setTssGroupMember(uint256 _threshold, bytes[] memory _batchPublicKey)
59:         public
60:         override
61:         onlyOwner
62:     {
63:         require((_batchPublicKey.length > 0), "batch public key is empty");
64:         require(_threshold < _batchPublicKey.length, "threshold must less than tss member");
65:         // require((inActiveTssMembers.length == 0), "inactive tss member array is not empty");
66:         if(inActiveTssMembers.length > 0) {
67:             for (uint256 i = 0; i < inActiveTssMembers.length; i++) {
68:                 // re-election clear data
69:                 delete groupKeyCounter[memberGroupKey[inActiveTssMembers[i]]];
70:                 delete memberGroupKey[inActiveTssMembers[i]];
71:                 delete isSubmitGroupKey[inActiveTssMembers[i]];
72:                 delete isInActiveMember[inActiveTssMembers[i]];
73:             }
74:             delete inActiveTssMembers;
75:         }
76:         for (uint256 i = 0; i < _batchPublicKey.length; i++) {
77:             inActiveTssMembers.push(_batchPublicKey[i]);
78:             isInActiveMember[_batchPublicKey[i]] = true;
79:             isSubmitGroupKey[_batchPublicKey[i]] = false;
80:         }
81:         threshold = _threshold;
82:         gRoundId = gRoundId + 1;
83:         confirmNumber = 0;
84:         emit tssGroupMemberAppend(gRoundId, _threshold, _batchPublicKey);
85:     }
```

Description

yekong : The setTssGroupMember function accepts an array of public keys _batchPublicKey and sets them as new TSS group members. It is currently not checking if _batchPublicKey contains any duplicate entries. If there are duplicate entries in the array, the function will push the same key multiple times into inActiveTssMembers and set the isInActiveMember and isSubmitGroupKey mappings for the same key multiple times. This could lead to undesired consequences, including inconsistent states and possibly erroneous behavior of the contract.

Recommendation

yekong : Before processing the _batchPublicKey array, it would be prudent to check that all its elements are unique. This could be achieved by using a temporary mapping to check the existence of each key in the array. If a duplicate key is found, the function should revert.

Below is an example of how this check can be implemented:

```
mapping(bytes => bool) seen;
for (uint256 i = 0; i < _batchPublicKey.length; i++) {
    require(!seen[_batchPublicKey[i]], "Duplicate keys not allowed");
    seen[_batchPublicKey[i]] = true;
}
```

This code should be placed before the processing of _batchPublicKey in the setTssGroupMember function. This will ensure that the array does not contain any duplicates and thus prevent the possible issues described above.

Client Response

Mitigated.Considering that setTssGroupMember is exclusively invoked during network initialization or when regenerating the aggregate public key, and restricted to execution by the Mantle team, Mantle DevOps validates the transaction parameters to prevent duplicate entries. Therefore, we have decided to postpone fixing this issue for now.

MTL-16:Potential Risks with Misbehaving _l2Token Contracts in finalizeDeposit

Category	Severity	Client Response	Contributor
Logical	Medium	Mitigated	yekong

Code Reference

- code/packages/contracts/contracts/L2/messaging/L2StandardBridge.sol#L149-L195

```
149:     function finalizeDeposit(
150:         address _l1Token,
151:         address _l2Token,
152:         address _from,
153:         address _to,
154:         uint256 _amount,
155:         bytes calldata _data
156:     ) external virtual onlyFromCrossDomainAccount(l1TokenBridge) {
157:         // Check the target token is compliant and
158:         // verify the deposited token on L1 matches the L2 deposited token representation here
159:         if (
160:             // slither-disable-next-line reentrancy-events
161:             ERC165Checker.supportsInterface(_l2Token, 0x1d1d8b63) &&
162:             _l1Token == IL2StandardERC20(_l2Token).l1Token()
163:         ) {
164:             // When a deposit is finalized, we credit the account on L2 with the same amount of
165:             // tokens.
166:             // slither-disable-next-line reentrancy-events
167:             IL2StandardERC20(_l2Token).mint(_to, _amount);
168:             // slither-disable-next-line reentrancy-events
169:             emit DepositFinalized(_l1Token, _l2Token, _from, _to, _amount, _data);
170:         } else {
171:             // Either the L2 token which is being deposited-into disagrees about the correct add
ress
172:             // of its L1 token, or does not support the correct interface.
173:             // This should only happen if there is a  malicious L2 token, or if a user somehow
174:             // specified the wrong L2 token address to deposit into.
175:             // In either case, we stop the process here and construct a withdrawal
176:             // message so that users can get their funds out in some cases.
177:             // There is no way to prevent malicious token contracts altogether, but this does li
mit
178:             // user error and mitigate some forms of malicious contract behavior.
179:             bytes memory message = abi.encodeWithSelector(
180:                 IL1ERC20Bridge.finalizeERC20Withdrawal.selector,
181:                 _l1Token,
182:                 _l2Token,
183:                 _to, // switched the _to and _from here to bounce back the deposit to the sender
184:                 _from,
```

```
185:                 _amount,
186:                 _data
187:             );
188:
189:             // Send message up to L1 bridge
190:             // slither-disable-next-line reentrancy-events
191:             sendCrossDomainMessage(l1TokenBridge, 0, message);
192:             // slither-disable-next-line reentrancy-events
193:             emit DepositFailed(_l1Token, _l2Token, _from, _to, _amount, _data);
194:         }
195:     }
```

Description

yekong : The finalizeDeposit function in the provided smart contract relies on the _l2Token contract's l1Token() function to return the correct address of the corresponding L1 token. This is used to compare with the user-inputted _l1Token address.

If the _l2Token contract behaves unexpectedly, such as returning an incorrect address when the l1Token() function is called, the finalizeDeposit function may fail to perform its intended operations correctly. A user who mistakenly provides an incorrect _l1Token will be affected by this.

The function currently contains a check for this scenario, and a withdrawal message is created in case of failure, attempting to return the user's funds. However, this only mitigates the potential problem and does not eliminate it.

Recommendation

yekong : Consider implementing a mechanism to further ensure the reliability of the _l2Token contract. One possible measure is to use a whitelist mechanism. Only allow calls from _l2Token contracts that are known to be secure and have been thoroughly audited.

This would give you more control over which contracts can interact with your smart contract and reduce the risk of interacting with misbehaving _l2Token contracts.

Client Response

Mitigated.We have added a whitelist feature for token address pairs in the bridge UI. The majority of users will utilize the bridge UI for token deposits and withdrawals. Therefore, only those users who directly interact with the bridge contract need to be cautious about any potential issues with _l2Token. Typically, these users are technical developers who are expected to have a comprehensive understanding of all contract behaviors.

MTL-17:Reentrancy in BVM_EigenDataLayrChain.confirmData

Category	Severity	Client Response	Contributor
Reentrancy	Medium	Declined	Secure3

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L276-L316

```
276:     function confirmData(
277:         bytes calldata data,
278:         IDataLayrServiceManager.DataStoreSearchData memory searchData,
279:         uint256 startL2Block,
280:         uint256 endL2Block,
281:         uint32 originDataStoreId,
282:         uint256 reConfirmedBatchIndex,
283:         bool isReRollup
284:     ) external {
285:         require(msg.sender == sequencer, "Only the sequencer can store data");
286:         require(dataStoreIdToL2RollUpBlock[searchData.metadata.globalDataStoreId].startL2BlockNu
mber == startL2Block &&
287:             dataStoreIdToL2RollUpBlock[searchData.metadata.globalDataStoreId].endBL2BlockNumber
== endL2Block &&
288:             dataStoreIdToL2RollUpBlock[searchData.metadata.globalDataStoreId].isReRollup == isRe
Rollup,
289:             "Data store either was not initialized by the rollup contract, or is already confirm
ed"
290:         );
291:         require(
292:             dataStoreIdToRollupStoreNumber[searchData.metadata.globalDataStoreId] == DATA_STORE_
INITIALIZED_BUT_NOT_CONFIRMED,
293:             "Data store either was not initialized by the rollup contract, or is already confirm
ed"
294:         );
295:         IDataLayrServiceManager(dataManageAddress).confirmDataStore(data, searchData);
296:         if (!isReRollup) {
297:             rollupBatchIndexRollupStores[rollupBatchIndex] = RollupStore({
298:                 originDataStoreId: searchData.metadata.globalDataStoreId,
299:                 dataStoreId: searchData.metadata.globalDataStoreId,
300:                 confirmAt: uint32(block.timestamp + fraudProofPeriod),
301:                 status: RollupStoreStatus.COMMITTED
302:             });
303:             l2ConfirmedBlockNumber = endL2Block;
304:             dataStoreIdToRollupStoreNumber[searchData.metadata.globalDataStoreId] = rollupBatchI
ndex;
305:             emit RollupStoreConfirmed(uint32(rollupBatchIndex++), searchData.metadata.globalData
StoreId, startL2Block, endL2Block);
306:         } else {
307:             rollupBatchIndexRollupStores[reConfirmedBatchIndex] = RollupStore({
308:                 originDataStoreId: originDataStoreId,
```

```
309:                 dataStoreId: searchData.metadata.globalDataStoreId,
310:                 confirmAt: uint32(block.timestamp + fraudProofPeriod),
311:                 status: RollupStoreStatus.COMMITTED
312:             });
313:             dataStoreIdToRollupStoreNumber[searchData.metadata.globalDataStoreId] = reConfirmedB
atchIndex;
314:             emit RollupStoreConfirmed(reConfirmedBatchIndex, searchData.metadata.globalDataStore
Id, startL2Block, endL2Block);
315:         }
316:     }
```

Description

Secure3 : Detection of the reentrancy bug. Do not report reentrancies that involve Ether (see reentrancy-eth).

```
function confirmData(
        bytes calldata data,
        IDataLayrServiceManager.DataStoreSearchData memory searchData,
        uint256 startL2Block,
        uint256 endL2Block,
        uint32 originDataStoreId,
        uint256 reConfirmedBatchIndex,
        bool isReRollup
    ) external {
        require(msg.sender == sequencer, "Only the sequencer can store data");
        require(dataStoreIdToL2RollUpBlock[searchData.metadata.globalDataStoreId].startL2BlockNumber
== startL2Block &&
            dataStoreIdToL2RollUpBlock[searchData.metadata.globalDataStoreId].endBL2BlockNumber == e
ndL2Block &&
            dataStoreIdToL2RollUpBlock[searchData.metadata.globalDataStoreId].isReRollup == isReRoll
up,
            "Data store either was not initialized by the rollup contract, or is already confirmed"
        );
        require(
            dataStoreIdToRollupStoreNumber[searchData.metadata.globalDataStoreId] == DATA_STORE_INIT
IALIZED_BUT_NOT_CONFIRMED,
            "Data store either was not initialized by the rollup contract, or is already confirmed"
        );
        IDataLayrServiceManager(dataManageAddress).confirmDataStore(data, searchData);
        if (!isReRollup) {
            rollupBatchIndexRollupStores[rollupBatchIndex] = RollupStore({
                originDataStoreId: searchData.metadata.globalDataStoreId,
                dataStoreId: searchData.metadata.globalDataStoreId,
                confirmAt: uint32(block.timestamp + fraudProofPeriod),
                status: RollupStoreStatus.COMMITTED
            });
            l2ConfirmedBlockNumber = endL2Block;
            dataStoreIdToRollupStoreNumber[searchData.metadata.globalDataStoreId] = rollupBatchInde
x;
            emit RollupStoreConfirmed(uint32(rollupBatchIndex++), searchData.metadata.globalDataStor
eId, startL2Block, endL2Block);
        } else {
            rollupBatchIndexRollupStores[reConfirmedBatchIndex] = RollupStore({
                originDataStoreId: originDataStoreId,
                dataStoreId: searchData.metadata.globalDataStoreId,
                confirmAt: uint32(block.timestamp + fraudProofPeriod),
```

```
status: RollupStoreStatus.COMMITTED
            });
            dataStoreIdToRollupStoreNumber[searchData.metadata.globalDataStoreId] = reConfirmedBatch
Index;
            emit RollupStoreConfirmed(reConfirmedBatchIndex, searchData.metadata.globalDataStoreId,
startL2Block, endL2Block);
        }
    }
```

Recommendation

Secure3 : Use the Checks-Effects-Interactions best practice and make all state changes before calling external contracts. Also, consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.

Client Response

Declined.The storeData and confirmData functions can only be accessed by the Sequencer role. Furthermore, our MantleDA contract includes an extra layer of permission control to prevent regular users from executing reentrancy attacks. As a result, our contract is secure and free from any security risks.

```
function storeData(
    bytes calldata header,
    uint8 duration,
    uint32 blockNumber,
    uint256 startL2Block,
    uint256 endL2Block,
    uint32 totalOperatorsIndex,
    bool   isReRollup
) external onlySequencer {
  .....
}

function confirmData(
    bytes calldata data,
    IDataLayrServiceManager.DataStoreSearchData memory searchData,
    uint256 startL2Block,
    uint256 endL2Block,
    uint32 originDataStoreId,
    uint256 reConfirmedBatchIndex,
    bool isReRollup
) external onlySequencer {
    ......
}
```

MTL-18:Unchecked return value may make the assertion fail to complete

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	biakia

Code Reference

- code/packages/contracts/contracts/L1/fraud-proof/Rollup.sol#L489-L498

```
489:     function newAssertionDeadline() private returns (uint256) {
490:         // TODO: account for prev assertion, gas
491:         // return block.number + confirmationPeriod;
492:         address scc = resolve("StateCommitmentChain");
493:         (bool success, bytes memory data) = scc.call(
494:             abi.encodeWithSignature("FRAUD_PROOF_WINDOW()")
495:         );
496:         uint256 confirmationWindow = uint256(bytes32(data));
497:         return block.timestamp + confirmationWindow;
498:     }
```

Description

biakia : In contract `Rollup`, the function `newAssertionDeadline` is used to calculate deadline for a new assertion. It will do a low-level call to call the function `FRAUD_PROOF_WINDOW()` in contract `StateCommitmentChain`:

```
function newAssertionDeadline() private returns (uint256) {
        // TODO: account for prev assertion, gas
        // return block.number + confirmationPeriod;
        address scc = resolve("StateCommitmentChain");
        (bool success, bytes memory data) = scc.call(
            abi.encodeWithSignature("FRAUD_PROOF_WINDOW()")
        );
        uint256 confirmationWindow = uint256(bytes32(data));
        return block.timestamp + confirmationWindow;
    }
```

The issue here is that the return value `success` is not checked. If `success` is false, then the return value `data` will not be the correct fraud proof window. If the `data` is parsed as a big number, then the assertion will not be completed.

Recommendation

biakia : Consider checking the returned value `success`:

```
function newAssertionDeadline() private returns (uint256) {
        // TODO: account for prev assertion, gas
        // return block.number + confirmationPeriod;
        address scc = resolve("StateCommitmentChain");
        (bool success, bytes memory data) = scc.call(
            abi.encodeWithSignature("FRAUD_PROOF_WINDOW()")
        );
        require(success,"call FRAUD_PROOF_WINDOW() failed");
        uint256 confirmationWindow = uint256(bytes32(data));
        return block.timestamp + confirmationWindow;
    }
```

Client Response

Fixed. Please refer to the code in our development branch for details.

```
function newAssertionDeadline() private returns (uint256) {
    // TODO: account for prev assertion, gas
    // return block.number + confirmationPeriod;
    address scc = resolve("StateCommitmentChain");
    (bool success, bytes memory data) = scc.call(
        abi.encodeWithSignature("FRAUD_PROOF_WINDOW()")
    );
    require(success,"call FRAUD_PROOF_WINDOW() failed");
    uint256 confirmationWindow = uint256(bytes32(data));
    return block.timestamp + confirmationWindow;
}
```

MTL-19: `TssRewardContract::totalAmount` isn't reduced in `claimReward` function and `updateReward` may fail

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	biakia

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L91-L128
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L176-L189

```
91:     function claimReward(uint256 _blockStartHeight, uint32 _length, uint256 _batchTime, address[]
calldata _tssMembers)
92:     external
93:     virtual
94:     onlyFromCrossDomainAccount(sccAddress)
95:     {
96:         if (IBVM_GasPriceOracle(bvmGasPriceOracleAddress).IsBurning() != 1) {
97:             claimRewardByBlock(_blockStartHeight, _length, _tssMembers);
98:             return;
99:         }
100:         uint256 sendAmount = 0;
101:         uint256 batchAmount = 0;
102:         uint256 accu = 0;
103:         // sendAmount
104:         if (lastBatchTime == 0) {
105:             lastBatchTime = _batchTime;
106:             return;
107:         }
108:         require(_batchTime > lastBatchTime,"args _batchTime must gther than last lastBatchTim
e");
109:         batchAmount = (_batchTime - lastBatchTime) * querySendAmountPerSecond() + dust;
110:         dust = 0;
111:         sendAmount = batchAmount.div(_tssMembers.length);
112:         for (uint256 j = 0; j < _tssMembers.length; j++) {
113:             address payable addr = payable(_tssMembers[j]);
114:             accu = accu.add(sendAmount);
115:             addr.transfer(sendAmount);
116:         }
117:         uint256 reserved = batchAmount.sub(accu);
118:         if (reserved > 0) {
119:             dust = dust.add(reserved);
120:         }
121:         emit DistributeTssReward(
122:             lastBatchTime,
123:             _batchTime,
124:             sendAmount,
125:             _tssMembers
126:         );
127:         lastBatchTime = _batchTime;
128:     }
```

```
176:     function updateReward(uint256 _blockID, uint256 _amount)
177:     external
178:     onlyFromDeadAddress
179:     checkBalance
180:     returns (bool)
181:     {
182:         // check update block ID
183:         require(_blockID == bestBlockID + 1, "block id update illegal");
184:         // iter address to update balance
185:         bestBlockID = _blockID;
186:         totalAmount = totalAmount.add(_amount);
187:         ledger[_blockID] = _amount;
188:         return true;
189:     }
```

Description

biakia : The purpose of the contract `TssRewardContract` is to distribute rewards to tss members. Rewards are first sent to the current contract in each block by the `updateReward` function, and then distributed by the `claimReward` function.

In `claimReward` function, if the `IsBurning` flag is set to 1, then the following distribution process will take place:

```
        require(_batchTime > lastBatchTime,"args _batchTime must gther than last lastBatchTime");
        batchAmount = (_batchTime - lastBatchTime) * querySendAmountPerSecond() + dust;
        dust = 0;
        sendAmount = batchAmount.div(_tssMembers.length);
        for (uint256 j = 0; j < _tssMembers.length; j++) {
            address payable addr = payable(_tssMembers[j]);
            accu = accu.add(sendAmount);
            addr.transfer(sendAmount);
        }
        uint256 reserved = batchAmount.sub(accu);
        if (reserved > 0) {
            dust = dust.add(reserved);
        }
        emit DistributeTssReward(
            lastBatchTime,
            _batchTime,
            sendAmount,
            _tssMembers
        );
        lastBatchTime = _batchTime;
```

It first calculates the total rewards over time and then divides them equally among each tss member. The issue here is that the value of `totalAmount` is not deducted after the rewards are distributed, which causes `address(this).balance` to be less than `totalAmount`. When rewards are sent to the current contract in each block, the `updateReward` function is called, and this function applies the `checkBalance` modifier:

```
  function updateReward(uint256 _blockID, uint256 _amount)
      external
      onlyFromDeadAddress
      checkBalance
      returns (bool)
      {
          // check update block ID
          require(_blockID == bestBlockID + 1, "block id update illegal");
          // iter address to update balance
          bestBlockID = _blockID;
          totalAmount = totalAmount.add(_amount);
          ledger[_blockID] = _amount;
          return true;
      }
```

Let's see what `checkBalance` has done:

```
    modifier checkBalance() {
           require(
               address(this).balance >= totalAmount,
               "balance record and contract balance are not equal"
           );
           _;
       }
```

It guarantees that `address(this).balance` must be greater than `totalAmount`. However, after the above discussion about `claimReward` we know that `address(this).balance` is less than `totalAmount` at this time, so `checkBalance` will fail. This will cause all subsequent `updateReward` calls to fail.

Recommendation

biakia : Consider deducting `totalAmount` in the claimReward function:

```
        for (uint256 j = 0; j < _tssMembers.length; j++) {
             address payable addr = payable(_tssMembers[j]);
             accu = accu.add(sendAmount);
             addr.transfer(sendAmount);
         }
         totalAmount = totalAmount.sub(accu);
         uint256 reserved = batchAmount.sub(accu);
         if (reserved > 0) {
             dust = dust.add(reserved);
         }
```

Client Response

Fixed.The claimReward function has undergone significant logic changes, and this issue doesn't apply any more.

```
function claimReward(uint256 _blockStartHeight, uint32 _length, uint256 _batchTime, address[] callda
ta _tssMembers)
external
virtual
onlyFromCrossDomainAccount(sccAddress)
{
    uint256 batchAmount = 0;
    // sendAmount
    if (lastBatchTime == 0) {
        lastBatchTime = _batchTime;
        return;
    }
    require(_batchTime > lastBatchTime,"args _batchTime must gther than last lastBatchTime");
    batchAmount = (_batchTime - lastBatchTime) * querySendAmountPerSecond();
    _distributeReward(batchAmount, _tssMembers);
    emit DistributeTssReward(
        lastBatchTime,
        _batchTime,
        batchAmount,
        _tssMembers
    );
    lastBatchTime = _batchTime;
}
```

The following functions have been deprecated and have been removed from the current version.

```
function updateReward(uint256 _blockID, uint256 _amount)
    external
    onlyFromDeadAddress
    checkBalance
    returns (bool)
    {
      // check update block ID
       require(_blockID == bestBlockID + 1, "block id update illegal");
    // iter address to update balance
    bestBlockID = _blockID;
    totalAmount = totalAmount.add(_amount);
    ledger[_blockID] = _amount;
    return true;
 }
```

MTL-20:Anyone can call `removeStake` in `Rollup` contract

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/packages/contracts/contracts/L1/fraud-proof/Rollup.sol#L197-L208
- code/packages/contracts/contracts/L1/fraud-proof/Rollup.sol#L259-L276

```
197:     function removeStake(address stakerAddress) external override {
198:         requireStaked(stakerAddress);
199:         // Require that staker is staked on a confirmed assertion.
200:         Staker storage staker = stakers[stakerAddress];
201:         if (staker.assertionID > lastConfirmedAssertionID) {
202:             revert("StakedOnUnconfirmedAssertion");
203:         }
204:         deleteStaker(stakerAddress);
205:         // Note: we don't need to modify assertion state because you can only unstake from a con
firmed assertion.
206:         (bool success,) = stakerAddress.call{value: staker.amountStaked}("");
207:         if (!success) revert("TransferFailed");
208:     }

259:     function createAssertionWithStateBatch(
260:         bytes32 vmHash,
261:         uint256 inboxSize,
262:         bytes32[] calldata _batch,
263:         uint256 _shouldStartAtElement,
264:         bytes calldata _signature
265:         ) external override stakedOnly { // todo batch submitter only
266:         // permissions only allow rollup proposer to submit assertion, only allow RollupContract
to append new batch
267:         require(msg.sender == resolve("BVM_Rolluper"), "msg.sender is not rollup proposer, can't
append batch");
268:         // create assertion
269:         createAssertion(vmHash, inboxSize);
270:         // append state batch
271:         address scc = resolve("StateCommitmentChain");
272:         (bool success, ) = scc.call(
273:             abi.encodeWithSignature("appendStateBatch(bytes32[],uint256,bytes)", _batch, _should
StartAtElement, _signature)
274:         );
275:         require(success, "scc append state batch failed, revert all");
276:     }
```

Description

biakia : In `Rollup` contract, the rollup proposer will submit assertion and append state batch to SCC by calling `createAssertionWithStateBatch` function. This function has a `stakedOnly` modifier, which means the rollup proposer should be a staker at first:

```
function createAssertionWithStateBatch(
        bytes32 vmHash,
        uint256 inboxSize,
        bytes32[] calldata _batch,
        uint256 _shouldStartAtElement,
        bytes calldata _signature
        ) external override stakedOnly {
...
...
}

modifier stakedOnly() {
        if (!isStaked(msg.sender)) {
            revert("NotStaked");
        }
        _;
 }
```

The function `removeStake` can be called by anyone to remove any staker, as long as the staker's assertion has been confirmed:

```
function removeStake(address stakerAddress) external override {
        requireStaked(stakerAddress);
        // Require that staker is staked on a confirmed assertion.
        Staker storage staker = stakers[stakerAddress];
        if (staker.assertionID > lastConfirmedAssertionID) {
            revert("StakedOnUnconfirmedAssertion");
        }
        deleteStaker(stakerAddress);
        // Note: we don't need to modify assertion state because you can only unstake from a confirm
ed assertion.
        (bool success,) = stakerAddress.call{value: staker.amountStaked}("");
        if (!success) revert("TransferFailed");
    }
```

Consider the following case:

1. The rollup proposer submits an assertion by calling `createAssertionWithStateBatch`, at this point it has an assertionID of 1
2. There were no new transactions during the fraud proof window, so that the latest assertionID of the rollup proposer is still 1.
3. After the fraud proof window, the assertion is confirmed, and the `lastConfirmedAssertionID` will be 1.

4. A malicious user calls the `removeStake` function before the rollup proposer creates the next assertion, passing in the address of the rollup proposer, at which point the `staker.assertionID` is 1 and the `lastConfirmedAssertionID` is also 1, so the `if` condition statement is skipped and the removal succeeds
5. The rollup proposer submits the next assertion now, and the call fails because it is no longer a staker.

Note: The above attack will only occur if there are no transactions for a long time, and considering the relatively small probability of occurrence, the severity of the issue is set to low.

Recommendation

biakia : Consider prohibiting the deletion of rollup proposer in `removeStake` function:

```
function removeStake(address stakerAddress) external override {
require(stakerAddress!= resolve("BVM_Rolluper"), "stakerAddress can not be rollup proposer");
...
...
```

Client Response

Fixed. The code has been updated with permission restrictions. Please find the modified code below. You can also take a look at the code in our development branch.

```
/// @inheritdoc IRollup
function removeStake(address stakerAddress) onlyOwner external override {
    requireStaked(stakerAddress);
    // Require that staker is staked on a confirmed assertion.
    Staker storage staker = stakers[stakerAddress];
    if (staker.assertionID > lastConfirmedAssertionID) {
        revert("StakedOnUnconfirmedAssertion");
    }
    uint256 amountToSent = staker.amountStaked;
    deleteStaker(stakerAddress);
    // send erc20 token to user
    require(
        IERC20(stakeToken).transfer(stakerAddress, amountToSent),
        "transfer erc20 token failed"
    );
}
/// @inheritdoc IRollup
function createAssertionWithStateBatch(
    bytes32 vmHash,
    uint256 inboxSize,
    bytes32[] calldata _batch,
    uint256 _shouldStartAtElement,
    bytes calldata _signature
    ) external override operatorOnly {
    // permissions only allow rollup proposer to submit assertion, only allow RollupContract to appe
nd new batch
    require(msg.sender == resolve("BVM_Rolluper"), "msg.sender is not rollup proposer, can't append
batch");
    // create assertion
    createAssertion(vmHash, inboxSize);
    // append state batch
    address scc = resolve("StateCommitmentChain");
    (bool success, ) = scc.call(
        abi.encodeWithSignature("appendStateBatch(bytes32[],uint256,bytes)", _batch, _shouldStartAtE
lement, _signature)
    );
    require(success, "scc append state batch failed, revert all");
}
```

MTL-21:BVM_EigenDataLayrChain.submitReRollUpInfo uses timestamp for comparisons

Category	Severity	Client Response	Contributor
Code Style	Low	Declined	Secure3

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L210-L224

```
210:    function submitReRollUpInfo(
211:        uint256 batchIndex
212:    ) external {
213:        require(msg.sender == reSubmitterAddress, "Only the re submitter can submit re rollup da
ta");
214:        RollupStore memory rStore = rollupBatchIndexRollupStores[batchIndex];
215:        if (rStore.dataStoreId > 0) {
216:            reRollupBatchIndex[reRollupIndex] = batchIndex;
217:            emit ReRollupBatchData(
218:                reRollupIndex++,
219:                batchIndex,
220:                dataStoreIdToL2RollUpBlock[rStore.dataStoreId].startL2BlockNumber,
221:                dataStoreIdToL2RollUpBlock[rStore.dataStoreId].endBL2BlockNumber
222:            );
223:        }
224:    }
```

Description

Secure3 : Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.

Consider below POC contract

```
 function submitReRollUpInfo(
        uint256 batchIndex
    ) external {
        require(msg.sender == reSubmitterAddress, "Only the re submitter can submit re rollup dat
a");
        RollupStore memory rStore = rollupBatchIndexRollupStores[batchIndex];
        if (rStore.dataStoreId > 0) {
            reRollupBatchIndex[reRollupIndex] = batchIndex;
            emit ReRollupBatchData(
                reRollupIndex++,
                batchIndex,
                dataStoreIdToL2RollUpBlock[rStore.dataStoreId].startL2BlockNumber,
                dataStoreIdToL2RollUpBlock[rStore.dataStoreId].endBL2BlockNumber
            );
        }
    }
    ...
}
```

Recommendation

Secure3 : Avoid relying on block.timestamp.

Client Response

Declined. The block.timestamp is not utilized in the code.

MTL-22:Inconsistency between comments and implementation in function `completeChallenge`

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	biakia

Code Reference

- code/packages/contracts/contracts/L1/fraud-proof/Rollup.sol#L420-L448

```
420:     function completeChallenge(address winner, address loser) external override {
421:         requireStaked(loser);
422:
423:         address challenge = getChallenge(winner, loser);
424:         if (msg.sender != challenge) {
425:             revert("NotChallenge");
426:         }
427:         uint256 amountWon;
428:         uint256 loserStake = stakers[loser].amountStaked;
429:         uint256 winnerStake = stakers[winner].amountStaked;
430:         if (loserStake > baseStakeAmount) {
431:             // If loser has a higher stake than the winner, refund the difference.
432:             // Loser gets deleted anyways, so maybe unnecessary to set amountStaked.
433:             // stakers[loser].amountStaked = winnerStake;
434:             withdrawableFunds[loser] += (loserStake - baseStakeAmount);
435:             amountWon = baseStakeAmount;
436:         } else {
437:             amountWon = loserStake;
438:         }
439:         // Reward the winner with half the remaining stake
440:         stakers[winner].amountStaked += amountWon; // why +stake instead of +withdrawable?
441:         stakers[winner].currentChallenge = address(0);
442:         // Turning loser into zombie renders the loser's remaining stake inaccessible.
443:         uint256 assertionID = stakers[loser].assertionID;
444:         deleteStaker(loser);
445:         // Track as zombie so we can account for it during assertion resolution.
446:         zombies.push(Zombie(loser, assertionID));
447:         challengeCtx.completed = true;
448:     }
```

Description

biakia : In the function `completeChallenge`, some comments are inconsistent with the implementation of the code:

```
        uint256 loserStake = stakers[loser].amountStaked;
        uint256 winnerStake = stakers[winner].amountStaked;
        if (loserStake > baseStakeAmount) {
            // If loser has a higher stake than the winner, refund the difference.
            // Loser gets deleted anyways, so maybe unnecessary to set amountStaked.
            // stakers[loser].amountStaked = winnerStake;
            withdrawableFunds[loser] += (loserStake - baseStakeAmount);
            amountWon = baseStakeAmount;
        } else {
            amountWon = loserStake;
        }
        // Reward the winner with half the remaining stake
        stakers[winner].amountStaked += amountWon; // why +stake instead of +withdrawable?
        stakers[winner].currentChallenge = address(0);
```

In comment, it says `If loser has a higher stake than the winner`, however, the code is `if (loserStake > baseStakeAmount)`, the variable `winnerStake` is not used. In comment, it says `Reward the winner with half the remaining stake`, however, the code is `stakers[winner].amountStaked += amountWon;`. The `amountWon` is not the half of the remaining stake.

Recommendation

biakia : Consider fixing comments to make them consistent with the code

Client Response

Fixed.The issue has been fixed as follows.

```
// uint256 winnerStake = stakers[winnerStaker].amountStaked;
if (loserStake > baseStakeAmount) {
    // If loser has a higher stake than the base stake amount, refund the difference.
    // Loser gets deleted anyways, so maybe unnecessary to set amountStaked.
    // stakers[loser].amountStaked = winnerStake;
    withdrawableFunds[loserStaker] += (loserStake - baseStakeAmount);
    amountWon = baseStakeAmount;
} else {
    amountWon = loserStake;
}
```

MTL-23:Incorrect Identification of Contract Accounts as EOAs in Address.isContract()

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	yekong

Code Reference

- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L69
- code/packages/contracts/contracts/test-helpers/L1StandardBridgeUpgrade.sol#L77

```
69:         require(!Address.isContract(msg.sender), "Account not EOA");

77:         require(!Address.isContract(msg.sender), "Account not EOA");
```

Description

yekong : The Address.isContract() function is used to check the code length of an address. However, during the construction of a contract, the code length is 0. This means that if the check is performed during the execution of the constructor, the isContract() method may incorrectly identify a contract account as an externally owned account (EOA).

Recommendation

yekong : To determine whether an address is an externally owned account (EOA), it is recommended to compare the msg.sender and tx.origin addresses. If they are equal, it indicates that the address is an EOA.

Client Response

Fixed. The issue has been fixed in this commit https://github.com/mantlenetworkio/mantle/pull/866/commits/7936855fc08f63827302fed9fa286495e10af51e

```
@@ -67,6 +70,7 @@ contract L1StandardBridge is IL1StandardBridge, CrossDomainEnabled {
67  70      modifier onlyEOA() {
68  71          // Used to stop deposits from contracts (avoid accidentally lost tokens)
69  72          require(!Address.isContract(msg.sender), "Account not EOA");
    73  +       require(tx.origin==msg.sender, "msg.sender is not ts origin");
70  74          _;
71  75      }
72  76
```

MTL-24:Incorrect buffer for gas limit

Category	Severity	Client Response	Contributor
Logical	Low	Declined	biakia

Code Reference

- code/tss/node/signer/keygen.go#L252-L309

```
252:func (p *Processor) EstimateGas(ctx context.Context, tx *etht.Transaction, rawContract *bind.Bou
ndContract, address ethc.Address) (*etht.Transaction, error) {
253:    header, err := p.l1Client.HeaderByNumber(ctx, nil)
254:    if err != nil {
255:            p.logger.Err(err).Msg("failed to get header by l1client")
256:            return nil, err
257:    }
258:    var gasPrice *big.Int
259:    var gasTipCap *big.Int
260:    var gasFeeCap *big.Int
261:    if header.BaseFee == nil {
262:            gasPrice, err = p.l1Client.SuggestGasPrice(ctx)
263:            if err != nil {
264:                    p.logger.Err(err).Msg("cannot fetch gas price")
265:                    return nil, err
266:            }
267:    } else {
268:            gasTipCap, err = p.l1Client.SuggestGasTipCap(ctx)
269:            if err != nil {
270:                    p.logger.Warn().Msg("failed to SuggestGasTipCap, FallbackGasTipCap = big.New
Int(1500000000) ")
271:                    gasTipCap = big.NewInt(1500000000)
272:            }
273:            gasFeeCap = new(big.Int).Add(
274:                    gasTipCap,
275:                    new(big.Int).Mul(header.BaseFee, big.NewInt(2)),
276:            )
277:    }
278:
279:    msg := ethereum.CallMsg{
280:            From:      p.address,
281:            To:        &address,
282:            GasPrice:  gasPrice,
283:            GasTipCap: gasTipCap,
284:            GasFeeCap: gasFeeCap,
285:            Value:     nil,
286:            Data:      tx.Data(),
287:    }
288:
289:    gasLimit, err := p.l1Client.EstimateGas(ctx, msg)
290:    if err != nil {
291:            p.logger.Err(err).Msg("failed to EstimateGas")
```

```
292:             return nil, err
293:     }
294:     opts, err := bind.NewKeyedTransactorWithChainID(p.privateKey, p.chainId)
295:     if err != nil {
296:             p.logger.Err(err).Msg("failed to new ops in estimate gas function")
297:             return nil, err
298:     }
299:     opts.Context = ctx
300:     opts.NoSend = true
301:     opts.Nonce = new(big.Int).SetUint64(tx.Nonce())
302:
303:     opts.GasTipCap = gasTipCap
304:     opts.GasFeeCap = gasFeeCap
305:     opts.GasLimit = 25 * gasLimit //add 20% buffer to gas limit
306:
307:     return rawContract.RawTransact(opts, tx.Data())
308:
309:}
```

Description

biakia : In module `keygen.go`, the function `setGroupPublicKey` will call `EstimateGas` to reset the gas limit before sending the transaction to the L1 blockchain:

```
func (p *Processor) setGroupPublicKey(localKey, poolPubkey []byte) error {
        ...
        ...
        newTx, err := p.EstimateGas(p.ctx, tx, rawTgmContract, address)

        if err != nil {
                p.logger.Err(err).Msg("got failed in estimate gas function ")
                return err
        }

        if err := p.l1Client.SendTransaction(p.ctx, newTx); err != nil {
                p.logger.Err(err).Msg("Unable to send transaction to l1 chain, need to retry ")
                for i := 0; i < 3; i++ {
                        p.logger.Info().Msgf("commit transaction retry %d times", i)
                        newTx, err = p.RetryTransaction(newTx, rawTgmContract)
                        if err == nil {
                                err = p.l1Client.SendTransaction(p.ctx, newTx)
                                if err == nil {
                                        break
                                }
                        }
                }
                return err
        }
}
```

The function `EstimateGas` will try to estimate the gas and add some buffer to gas limit:

```
func (p *Processor) EstimateGas(ctx context.Context, tx *etht.Transaction, rawContract *bind.BoundCo
ntract, address ethc.Address) (*etht.Transaction, error) {
         ...
         ...
         gasLimit, err := p.l1Client.EstimateGas(ctx, msg)
        if err != nil {
                p.logger.Err(err).Msg("failed to EstimateGas")
                return nil, err
        }
        opts, err := bind.NewKeyedTransactorWithChainID(p.privateKey, p.chainId)
        if err != nil {
                p.logger.Err(err).Msg("failed to new ops in estimate gas function")
                return nil, err
        }
        opts.Context = ctx
        opts.NoSend = true
        opts.Nonce = new(big.Int).SetUint64(tx.Nonce())

        opts.GasTipCap = gasTipCap
        opts.GasFeeCap = gasFeeCap
        opts.GasLimit = 25 * gasLimit //add 20% buffer to gas limit

        return rawContract.RawTransact(opts, tx.Data())

}
```

The comment shows that it will add 20% buffer to gas limit, however, the implementation is:

```
opts.GasLimit = 25 * gasLimit
```

It increases `gasLimit` by 25 times, which is unreasonable.

Recommendation

biakia : Consider below fix in the `EstimateGas` function

```
opts.GasLimit = 120 * gasLimit / 100
```

Client Response

Declined.Since the TssNode eventually invokes the setGroupPublicKey transaction in the TssGroupManager.sol contract, it's worth noting that the _updateTssMember function incurs a relatively high gas cost. The normal EstimateGas estimation may not be accurate in this case, and it may require a significantly larger gas limit to successfully execute the transaction

```
function setGroupPublicKey(bytes calldata _publicKey, bytes calldata _groupPublicKey)
    public
    override
{
    require(isInActiveMember[_publicKey], "your public key is not in InActiveMember");
    require(msg.sender == Lib_Address.publicKeyToAddress(_publicKey), "public key not match");
    require(_groupPublicKey.length == 64, "Invalid groupPublicKey length");

    if (!_isSubmitGroupKey[_publicKey]) {
        _isSubmitGroupKey[_publicKey] = true;
    }
    if (!_isEqual(_memberGroupKey[_publicKey], _groupPublicKey)) {
        _groupKeyCounter[_groupPublicKey] += 1;
        if (_memberGroupKey[_publicKey].length != 0) {
            _groupKeyCounter[_memberGroupKey[_publicKey]] -= 1;
        }
        _memberGroupKey[_publicKey] = _groupPublicKey;
    }
    if (_groupKeyCounter[_groupPublicKey] >= inActiveTssMembers.length) {
        _updateTssMember(_groupPublicKey);
    }
}
```

MTL-25:Incorrect check for `threshold`

Category	Severity	Client Response	Contributor
Logical	Low	Declined	biakia

Code Reference

- code/tss/node/tsslib/tss.go#L224-L257

```
224:func (t *TssServer) requestCheck(request interface{}) error {
225:    var threshold int
226:    switch value := request.(type) {
227:    case keygen.Request:
228:            threshold = value.ThresHold
229:            if len(value.Keys) <= threshold {
230:                    t.logger.Error().Msg("check params : pub_keys size is smaller than threshold
!")
231:                    return errors.New("check params : pub_keys size is smaller than threshold")
232:            }
233:    case keysign.Request:
234:            myPk, err := conversion.GetPubKeyFromPeerID(t.p2pCommunication.GetHost().ID().String
())
235:            if err != nil {
236:                    t.logger.Info().Msgf("fail to convert the p2p id(%s) to pubkey", t.p2pCommun
ication.GetHost().ID().String())
237:                    return err
238:            }
239:            isSignMember := false
240:            for _, el := range value.SignerPubKeys {
241:                    if myPk == el {
242:                            isSignMember = true
243:                            break
244:                    }
245:            }
246:            if !isSignMember {
247:                    t.logger.Info().Msgf("we(%s) are not the active signer", t.p2pCommunication.
GetHost().ID().String())
248:                    return errors.New("not active signer")
249:            }
250:
251:    default:
252:            t.logger.Error().Msg("unknown request type")
253:            return errors.New("unknown request type")
254:    }
255:    return nil
256:
257:}
```

Description

biakia : In `tss.go`, the function `requestCheck` is used to do checks for different request parameters. When it comes to `keygen.Request`, it will check for `threshold`:

```
case keygen.Request:
                threshold = value.ThresHold
                if len(value.Keys) <= threshold {
                        t.logger.Error().Msg("check params : pub_keys size is smaller than threshold
!")
                        return errors.New("check params : pub_keys size is smaller than threshold")
                }
```

The error message tells us that when the size of `pub_keys` is smaller than the `threshold`, the check will fail. However, the implementation is `len(value.Keys) <= threshold`, which means that when the size of `pub_keys` is equal to the `threshold`, the check will also fail. If the error message is accurate, then the comparison `len(value.Keys) <= threshold` here is incorrect.

Recommendation

biakia : Consider below fix in the `requestCheck` function

```
case keygen.Request:
                threshold = value.ThresHold
                if len(value.Keys) < threshold {
                        t.logger.Error().Msg("check params : pub_keys size is smaller than threshold
!")
                        return errors.New("check params : pub_keys size is smaller than threshold")
                }
```

Client Response

Declined.The requirement of len(value.Keys) <= threshold is accurate. However, the printed log message may not provide precise information. In future updates, it can be modified to "check params: pub_keys size should not exceed the threshold!" to enhance clarity.

```
if len(value.Keys) <= threshold {
   t.logger.Error().Msg("check params : pub_keys size is smaller than threshold !")
   return errors.New("check params : pub_keys size is smaller than threshold")
}
```

MTL-26:Lack of address zero Input Validation

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	Secure3, yekong

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrFee.sol#L24-L26
- code/packages/contracts/contracts/L1/deployment/ChugSplashDictator.sol#L45
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L50-L52
- code/packages/contracts/contracts/L2/predeploys/BVM_SequencerFeeVault.sol#L94-L96
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L119-L122
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/governance/Governor.sol#L480-L487

```
24:     function setFeeAddress(address _address) public onlyOwner {
25:         gasFeeAddress = _address;
26:     }

45:         finalOwner = _finalOwner;

50:     function setStakingSlash(address _address) public onlyOwner {
51:         stakingSlash = _address;
52:     }

94:     function setBurner(address _burner) public onlyOwner{
95:         burner = _burner;
96:     }

119:     function setFraudProofAddress(address _address) external {
120:         require(msg.sender == sequencer, "Only the sequencer can set fraud proof address unavail
able");
121:         fraudProofWhitelist[_address] = true;
122:     }

480:     function setMultisig(address _multisig) external onlyTimelock {
481:         _setMultisig(_multisig);
482:     }
483:
484:     function _setMultisig(address _multisig) internal {
485:         emit MultisigTransferred(multisig, _multisig);
486:         multisig = _multisig;
487:     }
```

Description

Secure3 : Detect missing zero address validation.

Consider below POC contract

```
finalOwner = _finalOwner
```

yekong : In some important set functions, the input address parameters are not verified. If the input parameter is 0x0, the function will not give any prompt, which may cause some serious impact

Recommendation

Secure3 : Check that the address is not zero.

yekong : To enhance the integrity and maintainability of the contract, it is advisable to incorporate suitable input

validation checks into these functions. In the event of identifying an invalid input, such as the presence of 0x0 as an address, it is recommended to promptly revert the transaction while providing a detailed error message. This proactive approach will effectively safeguard against erroneous updates to the contract's state and greatly facilitate the comprehension and troubleshooting of any potential problems that may arise.

Client Response

Fixed. The contract has been removed, and the remaining issues have been fixed.
code/packages/contracts/contracts/libraries/eigenda/lib/contracts/governance/Governor.sol#L480-L487

```
    function setFeeAddress(address _address) public onlyOwner {
        require(_address != address(0), "setFeeAddress: address is the zero address");
        address oldGasFeeAddress = gasFeeAddress;
        gasFeeAddress = _address;
        emit FeeAddressUpdated(oldGasFeeAddress, gasFeeAddress);
    }

    function setStakingSlash(address _address) public onlyOwner {
        require(_address != address(0), "param _address is the zero address");
        stakingSlash = _address;
    }

    function setFraudProofAddress(address _address) external onlySequencer {
        require(_address != address(0), "setFraudProofAddress: address is the zero address");
        fraudProofWhitelist[_address] = true;
    }
```

MTL-27:Lack of check for `db`

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	biakia

Code Reference

- code/tss/node/store/storage.go#L35-L37

```
35:func (s *Storage) Close() error {
36:     return s.db.Close()
37:}
```

Description

biakia : In `storage.go`, the function `Close()` is used to close the `db`. However, there is no check for `s.db`:

```
func (s *Storage) Close() error {
        return s.db.Close()
}
```

If the `s.db` is `nil`, calling this function will result in a panic error, which will cause the program to terminate.

Recommendation

biakia : Consider below fix in the `Close()` function

```
func (s *Storage) Close() error {
        if s.db!=nil{
                return s.db.Close()
        }
}
```

Client Response

Fixed.The issue has been resolved in the PR https://github.com/mantlenetworkio/mantle/pull/925/files

5 tss/manager/store/storage.go

```
@@ -32,7 +32,10 @@ func NewStorage(levelDbFolder string) (*Storage, error) {
 }

 func (s *Storage) Close() error {
-    return s.db.Close()
+    if s.db != nil {
+        return s.db.Close()
+    }
+    return nil
 }

 func handleError[T any](defaultValue T, err error) (T, error) {
```

MTL-28:Logic risk in `BVM_EigenDataLayrChain` contract `submitReRollUpInfo` function

Category	Severity	Client Response	Contributor
Logical	Low	Declined	yekong

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L210-L224

```
210:    function submitReRollUpInfo(
211:        uint256 batchIndex
212:    ) external {
213:        require(msg.sender == reSubmitterAddress, "Only the re submitter can submit re rollup da
ta");
214:        RollupStore memory rStore = rollupBatchIndexRollupStores[batchIndex];
215:        if (rStore.dataStoreId > 0) {
216:            reRollupBatchIndex[reRollupIndex] = batchIndex;
217:            emit ReRollupBatchData(
218:                reRollupIndex++,
219:                batchIndex,
220:                dataStoreIdToL2RollUpBlock[rStore.dataStoreId].startL2BlockNumber,
221:                dataStoreIdToL2RollUpBlock[rStore.dataStoreId].endBL2BlockNumber
222:            );
223:        }
224:    }
```

Description

yekong : We identified three potential vulnerabilities in the submitReRollUpInfo function in your Solidity contract. These issues may lead to unexpected behavior of the contract and could potentially be exploited.

Missing Input Validation: The function currently does not validate whether the input batchIndex corresponds to a valid RollupStore. Consequently, an invalid batchIndex could lead to the creation of a new RollupStore with uninitialized fields.

No Handling of Failure Condition: The function does not handle a potential failure condition where the dataStoreId of a RollupStore object is zero. In the current implementation, this condition would lead to an event emission with potentially incorrect values.

Duplicate batchIndex submissions: The function currently does not prevent the same batchIndex from being submitted more than once. This could lead to the same batchIndex being processed multiple times and could cause incorrect behavior.

Recommendation

yekong : To mitigate these vulnerabilities, we recommend the following:
Implement input validation to ensure the batchIndex corresponds to a valid RollupStore. Add a condition to handle the scenario where the dataStoreId of a RollupStore object is zero, and decide on the appropriate action based on your business logic. Keep track of already submitted batchIndex values and prevent the submission of a batchIndex that has already been submitted.

Client Response

Declined. The DataStoreId starts from 1 and increments sequentially. Therefore, this issue is not a concern.

MTL-29:Maximum Limit Issue with _batchPublicKey in setTssGroupMember Function

Category	Severity	Client Response	Contributor
Logical	Low	Mitigated	yekong

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L58-L85

```
58:     function setTssGroupMember(uint256 _threshold, bytes[] memory _batchPublicKey)
59:         public
60:         override
61:         onlyOwner
62:     {
63:         require((_batchPublicKey.length > 0), "batch public key is empty");
64:         require(_threshold < _batchPublicKey.length, "threshold must less than tss member");
65:         // require((inActiveTssMembers.length == 0), "inactive tss member array is not empty");
66:         if(inActiveTssMembers.length > 0) {
67:             for (uint256 i = 0; i < inActiveTssMembers.length; i++) {
68:                 // re-election clear data
69:                 delete groupKeyCounter[memberGroupKey[inActiveTssMembers[i]]];
70:                 delete memberGroupKey[inActiveTssMembers[i]];
71:                 delete isSubmitGroupKey[inActiveTssMembers[i]];
72:                 delete isInActiveMember[inActiveTssMembers[i]];
73:             }
74:             delete inActiveTssMembers;
75:         }
76:         for (uint256 i = 0; i < _batchPublicKey.length; i++) {
77:             inActiveTssMembers.push(_batchPublicKey[i]);
78:             isInActiveMember[_batchPublicKey[i]] = true;
79:             isSubmitGroupKey[_batchPublicKey[i]] = false;
80:         }
81:         threshold = _threshold;
82:         gRoundId = gRoundId + 1;
83:         confirmNumber = 0;
84:         emit tssGroupMemberAppend(gRoundId, _threshold, _batchPublicKey);
85:     }
```

Description

yekong : In the setTssGroupMember function, loops are used to process the _batchPublicKey array and the inActiveTssMembers array. If the size of these arrays is very large, this function could exceed the gas limit, making it impossible to execute. This could effectively freeze the functionality of the contract if this function is necessary for further operations.

Recommendation

yekong : A potential mitigation for this problem is to limit the size of the input arrays that the function will accept. By setting a maximum length for the _batchPublicKey array, the gas consumption of the function can be controlled to ensure

it doesn't exceed the block gas limit.

Here is an example of how you could implement this:

```
require(_batchPublicKey.length <= MAX_PUBLIC_KEYS, "Too many public keys");
```

In this line, MAX_PUBLIC_KEYS is a constant that you define in your contract to be the maximum allowable number of public keys. The specific value would depend on the gas costs of your operations, but it should be set to a value that ensures function execution doesn't get close to the block gas limit.

Client Response

Mitigated.While this issue does exist, the affected function is restricted to onlyOwner. It may occur during network initialization but can be avoided by devops diligence effort. Additionally, TSS network has a small number of nodes. We plan to fix it in future releases.

MTL-30:Parser.parse(bytes[],uint256,uint256).provenString is a local variable never initialized

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	Secure3

Code Reference

- code/packages/contracts/contracts/libraries/eigenda/Parse.sol#L6

```
6:         bytes memory provenString;
```

Description

Secure3 : Uninitialized local variables.

Consider below POC contract

```
function parse(bytes[] calldata polys, uint256 startIndex, uint256 length) public returns(bytes memo
ry)
    ...
}
```

Recommendation

Secure3 : Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

Client Response

Fixed. Code has been deprecated.

MTL-31:Response Id not checked in function `unmarshalResponseBytes`

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/tss/ws/client/tm/decode.go#L107-L115

```
107:func validateAndVerifyID(res *types.RPCResponse, expectedID types.JSONRPCIntID) error {
108:    //if err := validateResponseID(res.ID); err != nil {
109:    //      return err
110:    //}
111:    //if expectedID != res.ID.(types.JSONRPCIntID) { // validateResponseID ensured res.ID has th
e right type
112:    //      return fmt.Errorf("response ID (%d) does not match request ID (%d)", res.ID, expecte
dID)
113:    //}
114:    return nil
115:}
```

Description

biakia : In module `decode.go`, the functions `unmarshalResponseBytes` and `unmarshalResponseBytesArray` are used to parse the response. In the function `unmarshalResponseBytesArray`, ResponseIDs are batch-checked by `validateResponseIDs`:

```
func unmarshalResponseBytesArray(
        responseBytes []byte,
        expectedIDs []types.JSONRPCIntID,
        results []interface{},
) ([]interface{}, error) {

        ...
        ...
        if err := validateResponseIDs(ids, expectedIDs); err != nil {
                return nil, fmt.Errorf("wrong IDs: %w", err)
        }

        for i := 0; i < len(responses); i++ {
                if err := tmjson.Unmarshal(responses[i].Result, results[i]); err != nil {
                        return nil, fmt.Errorf("error unmarshalling #%d result: %w", i, err)
                }
        }

        return results, nil
}

func validateResponseIDs(ids, expectedIDs []types.JSONRPCIntID) error {
        m := make(map[types.JSONRPCIntID]bool, len(expectedIDs))
        for _, expectedID := range expectedIDs {
                m[expectedID] = true
        }

        for i, id := range ids {
                if m[id] {
                        delete(m, id)
                } else {
                        return fmt.Errorf("unsolicited ID #%d: %v", i, id)
                }
        }

        return nil
}
```

However, in function `unmarshalResponseBytes` the response id is not checked because the code in `validateAndVerifyID` is commented:

```
func unmarshalResponseBytes(
        responseBytes []byte,
        expectedID types.JSONRPCIntID,
        result interface{},
) (interface{}, error) {
        ....
        ....
        if err := validateAndVerifyID(response, expectedID); err != nil {
                return nil, fmt.Errorf("wrong ID: %w", err)
        }

        // Unmarshal the RawMessage into the result.
        if err := tmjson.Unmarshal(response.Result, result); err != nil {
                return nil, fmt.Errorf("error unmarshalling result: %w", err)
        }

        return result, nil
}

func validateAndVerifyID(res *types.RPCResponse, expectedID types.JSONRPCIntID) error {
        //if err := validateResponseID(res.ID); err != nil {
        //      return err
        //}
        //if expectedID != res.ID.(types.JSONRPCIntID) { // validateResponseID ensured res.ID has th
e right type
        //      return fmt.Errorf("response ID (%d) does not match request ID (%d)", res.ID, expecte
dID)
        //}
        return nil
}
```

The implementation of the two functions is inconsistent when it comes to checking the response id.

Recommendation

biakia : Consider uncommenting the code in the `validateAndVerifyID` function.

Client Response

Fixed. The code is deprecated and not used in production environment.

```
func validateAndVerifyID(res *types.RPCResponse, expectedID types.JSONRPCIntID) error {
    //if err := validateResponseID(res.ID); err != nil {
   //      return err
 //}
    //if expectedID != res.ID.(types.JSONRPCIntID) { // validateResponseID ensured res.ID has th
e right type
    //      return fmt.Errorf("response ID (%d) does not match request ID (%d)", res.ID, expecte
dID)
    //}
    return nil
}
```

MTL-32:Returned value of function `SendMsg` not checked

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/tss/node/signer/verify_slash.go#L31
- code/tss/node/signer/verify.go#L36
- code/tss/node/signer/sign_rollback.go#L38
- code/tss/node/signer/verify_rollback.go#L38
- code/tss/node/signer/sign_rollback.go#L45
- code/tss/node/signer/verify_rollback.go#L45
- code/tss/node/signer/sign_slash.go#L47
- code/tss/node/signer/sign.go#L48
- code/tss/node/signer/sign_rollback.go#L54
- code/tss/node/signer/sign_slash.go#L54
- code/tss/node/signer/sign.go#L55
- code/tss/node/signer/keygen.go#L56
- code/tss/node/signer/sign_slash.go#L63
- code/tss/node/signer/verify_slash.go#L65
- code/tss/node/signer/keygen.go#L69
- code/tss/node/signer/keygen.go#L76
- code/tss/node/signer/sign.go#L76
- code/tss/node/signer/sign_slash.go#L78
- code/tss/node/signer/verify.go#L79
- code/tss/node/signer/keygen.go#L84
- code/tss/node/signer/verify.go#L89
- code/tss/node/signer/verify.go#L95
- code/tss/node/signer/sign_slash.go#L106

```
31:                                     p.wsClient.SendMsg(RpcResponse)
36:                                     p.wsClient.SendMsg(RpcResponse)
38:                                     p.wsClient.SendMsg(RpcResponse)
38:                                     p.wsClient.SendMsg(RpcResponse)
45:                                     p.wsClient.SendMsg(RpcResponse)
45:                             p.wsClient.SendMsg(RpcResponse)
47:                                     p.wsClient.SendMsg(RpcResponse)
48:                                     p.wsClient.SendMsg(RpcResponse)
54:                                     p.wsClient.SendMsg(RpcResponse)
54:                                     p.wsClient.SendMsg(RpcResponse)
55:                                     p.wsClient.SendMsg(RpcResponse)
56:                                     p.wsClient.SendMsg(RpcResponse)
63:                                     p.wsClient.SendMsg(RpcResponse)
65:                             p.wsClient.SendMsg(RpcResponse)
69:                                     p.wsClient.SendMsg(RpcResponse)
76:                                             p.wsClient.SendMsg(RpcResponse)
76:             p.wsClient.SendMsg(RpcResponse)
78:                                     p.wsClient.SendMsg(RpcResponse)
79:                                             p.wsClient.SendMsg(RpcResponse)
84:                                             p.wsClient.SendMsg(RpcResponse)
89:                                     p.wsClient.SendMsg(RpcResponse)
```

```
95:                                     p.wsClient.SendMsg(RpcResponse)

106:                                    p.wsClient.SendMsg(errorRes)
```

Description

biakia : The returned value of function `SendMsg` is not checked in serval files:

```
p.wsClient.SendMsg(RpcResponse)
```

If `SendMsg` fails, the errors returned back are not handled, the program may run in the wrong way, causing serious problems.

Recommendation

biakia : Consider checking returned error in each call:

```
er := p.wsClient.SendMsg(RpcResponse)
if er != nil {
logger.Err(er).Msg("failed to send msg")
}
```

Client Response

Fixed.This issue has been fixed in the PR https://github.com/mantlenetworkio/mantle/pull/1293

MTL-33:Should avoid account nonce upper the limit `2^64–1`

Category	Severity	Client Response	Contributor
Integer Overflow and Underflow	Low	Fixed	Secure3

Code Reference

- code/l2geth/core/vm/evm.go#L418-L428

```
418:func (evm *EVM) create(caller ContractRef, codeAndHash *codeAndHash, gas uint64, value *big.Int,
address common.Address) ([]byte, common.Address, uint64, error) {
419:    // Depth check execution. Fail if we're trying to execute above the
420:    // limit.
421:    if evm.depth > int(params.CallCreateDepth) {
422:            return nil, common.Address{}, gas, ErrDepth
423:    }
424:    if !evm.CanTransfer(evm.StateDB, caller.Address(), value) {
425:            return nil, common.Address{}, gas, ErrInsufficientBalance
426:    }
427:    nonce := evm.StateDB.GetNonce(caller.Address())
428:    evm.StateDB.SetNonce(caller.Address(), nonce+1)
```

Description

Secure3 : At `code/l2geth/core/vm/evm.go`, function `func (evm *EVM) create()` does not limit account nonce according to EIP-2681. So if an account nonce is upper than `unit64` can hold, it will overflow.

Recommendation

Secure3 : Add a check to make sure nonce does not overflow.

Consider below fix

```
nonce := evm.StateDB.GetNonce(caller.Address())
        if nonce+1 < nonce {
                return nil, common.Address{}, gas, ErrNonceUintOverflow
        }
```

Client Response

Fixed. This issue has been fixed by PR https://github.com/mantlenetworkio/mantle/pull/1292

```
        if nonce+1 < nonce {
                return nil, common.Address{}, gas, ErrNonceUintOverflow
        }
```

MTL-34: `If` condition that can never be met in `advanceStake` function

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/packages/contracts/contracts/L1/fraud-proof/Rollup.sol#L211-L221

```
211:     function advanceStake(uint256 assertionID) external override stakedOnly {
212:         Staker storage staker = stakers[msg.sender];
213:         if (assertionID <= staker.assertionID && assertionID > lastCreatedAssertionID) {
214:             revert("AssertionOutOfRange");
215:         }
216:         // TODO: allow arbitrary descendant of current staked assertionID, not just child.
217:         if (staker.assertionID != assertions.getParentID(assertionID)) {
218:             revert("ParentAssertionUnstaked");
219:         }
220:         stakeOnAssertion(msg.sender, assertionID);
221:     }
```

Description

biakia : In contract `Rollup`, the function `advanceStake` is used to advance staker's current `assertionID` to descendant `assertionID`.

```
function advanceStake(uint256 assertionID) external override stakedOnly {
        Staker storage staker = stakers[msg.sender];
        if (assertionID <= staker.assertionID && assertionID > lastCreatedAssertionID) {
            revert("AssertionOutOfRange");
        }
        // TODO: allow arbitrary descendant of current staked assertionID, not just child.
        if (staker.assertionID != assertions.getParentID(assertionID)) {
            revert("ParentAssertionUnstaked");
        }
        stakeOnAssertion(msg.sender, assertionID);
 }
```

It will check the range of the passed parameter `assertionID`:

```
 if (assertionID <= staker.assertionID && assertionID > lastCreatedAssertionID) {
            revert("AssertionOutOfRange");
 }
```

This `if` condition will always be false because `staker.assertionID` is always be less than or equal to `lastCreatedAssertionID`. If `staker.assertionID< lastCreatedAssertionID`, consider `staker.assertionID` is 100 now and `lastCreatedAssertionID` is 102, then the `if` condition will be `(assertionID < 100 && assertionID> 102)`. No number can be both less than 100 and greater than 102, so this condition will always be false. If `staker.assertionID == lastCreatedAssertionID`, the `if` condition is still false. I suspect that `||` should be used here instead of `&&`.

Recommendation

biakia : Consider below fix in the `advanceStake` function

```
function advanceStake(uint256 assertionID) external override stakedOnly {
        Staker storage staker = stakers[msg.sender];
        if (assertionID <= staker.assertionID || assertionID > lastCreatedAssertionID) {
            revert("AssertionOutOfRange");
        }
        // TODO: allow arbitrary descendant of current staked assertionID, not just child.
        if (staker.assertionID != assertions.getParentID(assertionID)) {
            revert("ParentAssertionUnstaked");
        }
        stakeOnAssertion(msg.sender, assertionID);
 }
```

Client Response

Fixed.This issue has been fixed in PR: https://github.com/mantlenetworkio/mantle/pull/945

```
@@ -231,7 +238,7 @@ contract Rollup is Lib_AddressResolver, RollupBase, Whitelist {
231 238     function advanceStake(uint256 assertionID) external override operatorOnly {
232 239         address stakerAddr = registers[msg.sender];
233 240         Staker storage staker = stakers[stakerAddr];
234     -       if (assertionID <= staker.assertionID && assertionID > lastCreatedAssertionID) {
    241 +       if (assertionID <= staker.assertionID || assertionID > lastCreatedAssertionID) {
235 242             revert("AssertionOutOfRange");
236 243         }
237 244         // TODO: allow arbitrary descendant of current staked assertionID, not just child.
@@ -242,11 +249,13 @@ contract Rollup is Lib_AddressResolver, RollupBase, Whitelist {
242 249     }
```

MTL-35: `L1ChugSplashProxy::onlyWhenNotPaused` does not take effect.

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	Secure3

Code Reference

- code/packages/contracts/contracts/L1/deployment/ChugSplashDictator.sol#L18

```
18:    bool public isUpgrading = true;
```

Description

Secure3 : In the `L1ChugSplashProxy`, a modify named `onlyWhenNotPaused` use `iL1ChugSplashDeployer::isUpgrading` functions's return value to judge Pause this contracts.But in `ChugSplashDictator::isUpgrading` which is refer `iL1ChugSplashDeployer` is always true and has no function to change it.

```
    modifier onlyWhenNotPaused() {
        address owner = _getOwner();
        (bool success, bytes memory returndata) = owner.staticcall(
            abi.encodeWithSelector(iL1ChugSplashDeployer.isUpgrading.selector)
        );
        if (success && returndata.length == 32) {
            uint256 ret = abi.decode(returndata, (uint256));
            require(ret == 0, "L1ChugSplashProxy: system is currently being upgraded");
        }
        _;
    }
}
interface iL1ChugSplashDeployer {
    function isUpgrading() external view returns (bool);
}
contract ChugSplashDictator is iL1ChugSplashDeployer {
    bool public isUpgrading = true;
    ...
```

Recommendation

Secure3 : Creat a function which can change `isUpgrading`

```
function setisUpgrading(bool _isUpgrading) {
        require(msg.sender == finalOwner, "ChugSplashDictator: only callable by finalOwner");
        isUpgrading =_isUpgrading;
}
```

Client Response

Acknowledged. This is native code from Optimism and we haven't made any modifications to it. We will consider fixing it in the future.

MTL-36: `L1StandardBridge` does not support fee-on-transfer tokens

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	biakia

Code Reference

- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L177-L230

```
177:     function _initiateERC20Deposit(
178:         address _l1Token,
179:         address _l2Token,
180:         address _from,
181:         address _to,
182:         uint256 _amount,
183:         uint32 _l2Gas,
184:         bytes calldata _data
185:     ) internal {
186:         // When a deposit is initiated on L1, the L1 Bridge transfers the funds to itself for fu
ture
187:         // withdrawals. The use of safeTransferFrom enables support of "broken tokens" which do
not
188:         // return a boolean value.
189:         // slither-disable-next-line reentrancy-events, reentrancy-benign
190:         IERC20(_l1Token).safeTransferFrom(_from, address(this), _amount);
191:
192:         // Construct calldata for _l2Token.finalizeDeposit(_to, _amount)
193:         bytes memory message;
194:         if (_l1Token == l1BitAddress) {
195:             // Construct calldata for finalizeDeposit call
196:             _l2Token = Lib_PredeployAddresses.BVM_BIT;
197:             message = abi.encodeWithSelector(
198:                 IL2ERC20Bridge.finalizeDeposit.selector,
199:                 address(0x1A4b46696b2bB4794Eb3D4c26f1c55F9170fa4C5),
200:                 Lib_PredeployAddresses.BVM_BIT,
201:                 _from,
202:                 _to,
203:                 _amount,
204:                 _data
205:             );
206:
207:         } else {
208:             // Construct calldata for finalizeDeposit call
209:             message = abi.encodeWithSelector(
210:                 IL2ERC20Bridge.finalizeDeposit.selector,
211:                 _l1Token,
212:                 _l2Token,
213:                 _from,
214:                 _to,
215:                 _amount,
216:                 _data
```

```
217:             );
218:         }
219:
220:
221:         // Send calldata into L2
222:         // slither-disable-next-line reentrancy-events, reentrancy-benign
223:         sendCrossDomainMessage(l2TokenBridge, _l2Gas, message);
224:
225:         // slither-disable-next-line reentrancy-benign
226:         deposits[_l1Token][_l2Token] = deposits[_l1Token][_l2Token] + _amount;
227:
228:         // slither-disable-next-line reentrancy-events
229:         emit ERC20DepositInitiated(_l1Token, _l2Token, _from, _to, _amount, _data);
230:     }
```

Description

biakia : In contract `L1StandardBridge`, the function `_initiateERC20Deposit` will deposit `_l1Token` to current contract and record the amount:

```
IERC20(_l1Token).safeTransferFrom(_from, address(this), _amount);
...
...
// slither-disable-next-line reentrancy-benign
deposits[_l1Token][_l2Token] = deposits[_l1Token][_l2Token] + _amount;
```

If the `_l1Token` is a fee-on-transfer token, the actual amount the contract received will be less than `_amount`. When user withdraws tokens from L2, it will finally call `finalizeERC20Withdrawal` function in `L1StandardBridge`:

```
function finalizeERC20Withdrawal(
        address _l1Token,
        address _l2Token,
        address _from,
        address _to,
        uint256 _amount,
        bytes calldata _data
    ) public onlyFromCrossDomainAccount(l2TokenBridge) {
        deposits[_l1Token][_l2Token] = deposits[_l1Token][_l2Token] - _amount;

        // When a withdrawal is finalized on L1, the L1 Bridge transfers the funds to the withdrawer
        // slither-disable-next-line reentrancy-events
        IERC20(_l1Token).safeTransfer(_to, _amount);

        // slither-disable-next-line reentrancy-events
        emit ERC20WithdrawalFinalized(_l1Token, _l2Token, _from, _to, _amount, _data);
    }
```

It will fail to transfer `_amount` tokens to the user because there is no enough tokens in the current contract.

Recommendation

biakia : Consider using the actual amount the contract received:

```
uint256 before = IERC20(_l1Token).balanceOf(address(this));
IERC20(_l1Token).safeTransferFrom(_from, address(this), _amount);
uint256 after = IERC20(_l1Token).balanceOf(address(this));
_amount = after - before;
```

Client Response

Acknowledged. Risk is mitigated by the standard practice of manually approving tokens to be added to the canonical bridge through a registration and due diligence process. Future iterations of the canonical bridge may enable automatic addition of tokens, which requires further checks and balances.

MTL-37: `checkBalance` is not working as expected for `TssRewardContract` contract `updateReward` function

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	alansh

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L179

```
179:    checkBalance
```

Description

alansh : The flow is:

1. balance of `TssRewardContract` is incremented in `StateTransition`.
2. `updateReward` is called to increment `totalAmount`

`checkBalance` in fact wants to check `this.balance>= totalAmount` **after** `totalAmount` is incremented. But currently it's checked **before** `totalAmount` is incremented. This may cause the first place to trigger a bug to be missed.

Recommendation

alansh : check balance after `totalAmount` is incremented

Client Response

Fixed.Code has been deprecated

MTL-38:should not ignore error in NewStateTransition

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	alansh

Code Reference

- code/l2geth/core/state_transition.go#L141
- code/l2geth/core/state_transition.go#L150

```
141:                    l1Fee, _ = fees.CalculateL1MsgFee(msg, evm.StateDB, nil)

150:                            daFee, _ = fees.CalculateDAMsgFee(msg, evm.StateDB, nil)
```

Description

alansh : When there's an error, it's safer to return that error instead of ignoring directly.

Recommendation

alansh : Handle error case.

Client Response

Fixed.This issue has been fixed in PR: https://github.com/mantlenetworkio/mantle/pull/1295

MTL-39: no receive ether function with fallback

Category	Severity	Client Response	Contributor
Code Style	Informational	Declined	Secure3

Code Reference

- code/packages/contracts/contracts/chugsplash/L1ChugSplashProxy.sol#L17

```
17:contract L1ChugSplashProxy {
```

Description

Secure3 : This contract has a payable fallback function, but no receive ether function.

Consider below POC contract

```
contract L1ChugSplashProxy {
  (Relevant source part starts here and spans across multiple lines)

 fallback() external payable {
   (Relevant source part starts here and spans across multiple lines)
```

Recommendation

Secure3 : Consider adding a receive ether function.

Client Response

Declined.Implementing the receive method in the proxy contract is not required. If required, the implementation contract will support the receive ether function.

MTL-40:Division Before Multiplication

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	biakia

Code Reference

- code/mt-batcher/services/sequencer/driver.go#L564-L576

```
564:func (d *Driver) CalcUserFeeByRules(rollupDateSize *big.Int) (*big.Int, error) {
565:    seconds := new(big.Int).Div(big.NewInt(int64(d.Cfg.MainWorkerPollInterval)), big.NewInt(1000
000000))
566:    rollSizeSec := new(big.Int).Div(rollupDateSize, seconds)
567:    feeSs, ok := new(big.Int).SetString(d.Cfg.FeeSizeSec, 10)
568:    if !ok {
569:            log.Error("FeeSizeSec from string to big.int fail")
570:            return big.NewInt(0), nil
571:    }
572:    if rollSizeSec.Cmp(feeSs) < 0 {
573:            return big.NewInt(0), nil
574:    }
575:    return new(big.Int).Mul(new(big.Int).Div(rollSizeSec, feeSs), new(big.Int).SetUint64(d.Cfg.F
eePerBytePerTime)), nil
576:}
```

Description

biakia :

```
return new(big.Int).Mul(new(big.Int).Div(rollSizeSec, feeSs), new(big.Int).SetUint64(d.Cfg.FeePerByt
ePerTime)), nil
```

The returned value in the function `CalcUserFeeByRules` is `rollSizeSec / feeSs * d.Cfg.FeePerBytePerT ime`. It performs divisions before multiplications. It can sometimes cause loss of precision.

Recommendation

biakia : Consider applying multiplications before divisions:

```
feePerBytePerTime =  new(big.Int).SetUint64(d.Cfg.FeePerBytePerTime)
return new(big.Int).Div(new(big.Int).Mul(rollSizeSec,feePerBytePerTime),feeSs),nil
```

Client Response

Acknowledged. Our calculations are measured in Wei. We can tolerate minor loss of accuracy.

MTL-41:Gas Optimization : Replace Hardcoded require with revert

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	yekong

Code Reference

- code/packages/contracts/contracts/test-helpers/FailingReceiver.sol#L6
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L259

```
6:        require(false, "FailingReceiver");

259:            require(false, "err type for slashing");
```

Description

yekong : In the original implementation, the require statement is used to throw an error message with a hardcoded false condition. However, this condition will always evaluate to false, resulting in an immediate error and reverting the transaction. In such cases, it is more appropriate and gas-efficient to use the revert statement instead.

By using revert directly, you can achieve the same result of throwing an error message while avoiding unnecessary condition checks, thus optimizing gas usage.

Recommendation

yekong : Replace the require statement with revert to streamline the error-throwing behavior and improve gas efficiency.
example:

```
contract FailingReceiver {
    receive() external payable {
        require(false, "FailingReceiver");
    }
}
```

you can change it to

```
contract FailingReceiver {
    error FailingReceiver();
    receive() external payable {
        revert FailingReceiver()
    }
}
```

Client Response

Fixed. Please refer the following for details.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/test-helpers/FailingReceiver.sol#L6
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L267

```
FailingReceiver.sol

contract FailingReceiver {
    receive() external payable {
        revert("FailingReceiver");
    }
}


TssStakingSlashing.sol

revert("err type for slashing");
```

MTL-42:Gas Optimization : Streamline require Statements and Boolean Usage

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	yekong

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L95
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L338

```
95:         require(isInActiveMember[_publicKey] == true, "your public key is not in InActiveMembe
r");

338:         require(fraudProofWhitelist[msg.sender] == true, "Only fraud proof white list can challe
nge data");
```

Description

yekong : In conditional statements, boolean values can be used directly without the need to compare them explicitly to true. This comparison is redundant and can be simplified for better readability and gas optimization.

Recommendation

yekong : Instead of comparing boolean values to true, use the boolean values directly in conditional statements.

```
require(fraudProofWhitelist[msg.sender], "Only fraud proof white list can challenge data");

require(isInActiveMember[_publicKey], "your public key is not in InActiveMember");
```

By directly using boolean values, the code becomes more concise and eliminates unnecessary comparisons.

Client Response

Fixed.Please refer to the following for details.

```
function proveFraud(
    uint256 fraudulentStoreNumber,
    uint256 startIndex,
    IDataLayrServiceManager.DataStoreSearchData memory searchData,
    DisclosureProofs calldata disclosureProofs
) external {
    require(fraudProofWhitelist[msg.sender], "proveFraud: Only fraud proof white list can challenge
data");
    RollupStore memory rollupStore = rollupBatchIndexRollupStores[fraudulentStoreNumber];
    require(rollupStore.status == RollupStoreStatus.COMMITTED && rollupStore.confirmAt > block.times
tamp, "RollupStore must be committed and unconfirmed");
    ......
 }
function setGroupPublicKey(bytes calldata _publicKey, bytes calldata _groupPublicKey)
    public
    override
{
    require(isInActiveMember[_publicKey], "your public key is not in InActiveMember");
    require(msg.sender == Lib_Address.publicKeyToAddress(_publicKey), "public key not match");
    require(_groupPublicKey.length == 64, "Invalid groupPublicKey length");
    ......
}
```

MTL-43:Gas Optimization: ChugSplashDictator.bitAddressSlotKey should be immutable

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	Secure3

Code Reference

- code/packages/contracts/contracts/L1/deployment/ChugSplashDictator.sol#L22-L27

```
22:     bytes32 public messengerSlotKey;
23:     bytes32 public messengerSlotVal;
24:     bytes32 public bridgeSlotKey;
25:     bytes32 public bridgeSlotVal;
26:     bytes32 public bitAddressSlotKey;
27:     bytes32 public bitAddressSlotVal;
```

Description

Secure3 : State variables that are not updated following deployment should be declared immutable to save gas.

Consider below POC contract

```
bytes32 public codeHash;
    bytes32 public messengerSlotKey;
    bytes32 public messengerSlotVal;
    bytes32 public bridgeSlotKey;
    bytes32 public bridgeSlotVal;
    bytes32 public bitAddressSlotKey;
    bytes32 public bitAddressSlotVal;
```

Recommendation

Secure3 : Add the immutable attribute to state variables that never change or are set only in the constructor.

Client Response

Fixed. PR:https://github.com/mantlenetworkio/mantle/pull/1114
issue:https://github.com/mantlenetworkio/mantle/issues/1113

```
bytes32 immutable public messengerSlotKey;
bytes32 immutable public messengerSlotVal;
bytes32 immutable public bridgeSlotKey;
bytes32 immutable public bridgeSlotVal;
bytes32 immutable public mantleAddressSlotKey;
bytes32 immutable public mantleAddressSlotVal;
```

MTL-44:Gas Optimization: Inefficient Deletion of Elements in Large Arrays

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	yekong

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L318-L324

```
318:    function removeActiveTssMembers(uint256 _index) private {
319:        require(_index < activeTssMembers.length, "index out of bound");
320:        for (uint256 i = _index; i < activeTssMembers.length - 1; i++) {
321:            activeTssMembers[i] = activeTssMembers[i + 1];
322:        }
323:        activeTssMembers.pop();
324:    }
```

Description

yekong : The current implementation deletes specific elements from an array by moving all subsequent elements up one position, which can be very expensive in terms of gas consumption, especially when the array is very large. This approach results in high computational costs and can lead to performance issues.

Recommendation

yekong : To optimize gas usage and improve efficiency, it is recommended to implement a different approach for deleting elements from the array. One possible optimization is to swap the element that needs to be removed with the last element in the array, and then delete the last element directly. This approach avoids the need to move all subsequent elements and reduces the overall gas cost.

Client Response

Fixed.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L344C1-L347C6

```
    function _removeActiveTssMembers(uint256 _index) private {
        activeTssMembers[_index] = activeTssMembers[activeTssMembers.length - 1];
        activeTssMembers.pop();
    }
```

MTL-45:Gas Optimization: It is not necessary to use SafeMath in solidity version 0.8+

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	biakia, yekong

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrFee.sol#L6
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L6
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L7
- code/packages/contracts/contracts/da/BVM_EigenDataLayrFee.sol#L10
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L11
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L16
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L17
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L21-L22
- code/packages/contracts/contracts/L1/fraud-proof/WhiteList.sol#L23
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/BloomLib.sol#L36
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/MemoryLib.sol#L36
- code/packages/contracts/contracts/L1/fraud-proof/libraries/MPT.sol#L42
- code/packages/contracts/contracts/L1/deployment/AddressDictator.sol#L52
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/MemoryLib.sol#L87
- code/packages/contracts/contracts/L1/fraud-proof/libraries/RLPWriter.sol#L113
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/MemoryLib.sol#L115
- code/packages/contracts/contracts/L1/fraud-proof/libraries/RLPReader.sol#L124
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/EVMTypesLib.sol#L132
- code/packages/contracts/contracts/L1/fraud-proof/Rollup.sol#L145
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/MemoryLib.sol#L169
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/MemoryLib.sol#L191
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L196-L204
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/MemoryLib.sol#L262
- code/packages/contracts/contracts/L1/fraud-proof/libraries/MerkleLib.sol#L278
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/MemoryLib.sol#L292

```
6:import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";

6:import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";

7:import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";

10:    using SafeMathUpgradeable for uint256;

11:import {SafeMath} from "@openzeppelin/contracts/utils/math/SafeMath.sol";

16:    using SafeMathUpgradeable for uint256;

17:    using SafeMathUpgradeable for uint256;

21:contract TssRewardContract is Ownable,ITssRewardContract,CrossDomainEnabled {
22:    using SafeMath for uint256;

23:        for (uint i = 0; i < toAddAddresses.length; i++) {

36:        for (uint256 i = 0; i < 8; i++) {

36:        for (uint256 i = 0; i < cellNum - 1; i++) {

42:        for (uint256 i = 0; i < stack.length; i++) {

52:        for (uint256 i = 0; i < _names.length; i++) {

87:                        for (uint64 i = 0; i < cellNum; i++) {

113:            for (i = 1; i <= lenLen; i++) {

115:                for (uint64 i = 0; i < existCellNum; i++) {

124:        for (uint256 i = 0; i < items; i++) {

132:        for (uint256 i = 0; i < topics.length; i++) {

145:        for (uint i = 0; i < whitelists.length; i++) {

169:                        for (uint64 i = 0; i < cellNum; i++) {

191:                for (uint64 i = 0; i < cellNum; i++) {
```

```
196:     function resetRollupBatchData(uint256 _rollupBatchIndex) external {
197:         require(msg.sender == sequencer, "Only the sequencer can update sequencer address");
198:         for (uint256 i = 0; i < rollupBatchIndex; i++) {
199:             delete rollupBatchIndexRollupStores[i];
200:         }
201:         rollupBatchIndex = _rollupBatchIndex;
202:         l2StoredBlockNumber = 1;
203:         l2ConfirmedBlockNumber = 1;
204:     }

262:                         for (uint64 i = 0; i < cellNum; i++) {

278:         for (uint64 i = 0; i < element_count; i++) {

292:                 for (uint64 i = 0; i < cellNum; i++) {
```

Description

biakia : Solidity provides overflow checking for version above 0.8. The contract `TssRewardContract` does not need to import SafeMath library for overflow checking, which can save gas.
yekong : Starting from Solidity version 0.8, overflow and underflow checking are enabled by default for arithmetic operations. Therefore, there is no need to import and use the SafeMath library specifically for overflow checking. Removing the SafeMath library can help save gas in the contract.
yekong : After solidity version 0.8, solidity will perform a check every time the data changes to determine whether it overflows, so as to decide whether to throw an exception.

At the same time, because of the additional gas consumption caused by the detection, through the reasonable use of unchecked, the intermediate detection link can be effectively removed, so as to achieve the purpose of gas saving.

I also recommend using `++i` instead of `i++` to keep the code uniform, as in other contracts of this project, and it will also save gas.

Recommendation

biakia : Consider removing SafeMath library.
yekong : Remove the SafeMath library import and usages in the contract to optimize gas usage.
yekong : It is recommended to make modifications according to the following example： change before

```
function resetRollupBatchData(uint256 _rollupBatchIndex) external {
        require(msg.sender == sequencer, "Only the sequencer can update sequencer address");
        for (uint256 i = 0; i < rollupBatchIndex; i++) {
            delete rollupBatchIndexRollupStores[i];
        }
        rollupBatchIndex = _rollupBatchIndex;
        l2StoredBlockNumber = 1;
        l2ConfirmedBlockNumber = 1;
    }
```

change after

```
function resetRollupBatchData(uint256 _rollupBatchIndex) external {
        require(msg.sender == sequencer, "Only the sequencer can update sequencer address");
        for (uint256 i = 0; i < rollupBatchIndex; ) {
            delete rollupBatchIndexRollupStores[i];
            unchecked {
                  ++i;
            }
        }
        rollupBatchIndex = _rollupBatchIndex;
        l2StoredBlockNumber = 1;
        l2ConfirmedBlockN umber = 1;

    }
```

There may be omissions in the annotation, please use the global search to check again

Client Response

Acknowledged. This optimization will be addressed in future releases.

MTL-46:Gas Optimization: Remove unused Hardhat imports

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	biakia

Code Reference

- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/DataLayr/DataLayrServiceManager.sol#L20
- code/packages/contracts/contracts/L1/fraud-proof/Rollup.sol#L28

```
20:import "hardhat/console.sol";

28:import "hardhat/console.sol";
```

Description

biakia : The contract `Rollup` includes the following unnecessary Hardhat imports:

```
import "hardhat/console.sol";
```

Recommendation

biakia : Consider removing the import statement to save on deployment gas costs.

Client Response

Acknowledged. This optimization will be addressed in future releases.

MTL-47:Gas Optimization: `SlashType.nothing` is useless in `TssStakingSlashing` contract

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L17

```
17:         nothing,
```

Description

alansh : `SlashType.nothing` is useless since it's not accepted by `slash` function.(will trigger `require(false, "err type for slashing")`)

Recommendation

alansh : Remove `SlashType.nothing`

Client Response

Fixed.Please refer to the fix as follows.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L33

```
    enum SlashType {
        uptime,
        animus
    }
```

MTL-48:Gas Optimization: `_distributeTssReward` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	biakia

Code Reference

- code/packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol#L360-L388

```
360:     function _distributeTssReward(bytes32[] memory _batch, uint256 _shouldStartAtElement) intern
al {
361:         // get address of tss group member
362:         address[] memory tssMembers = ITssGroupManager(resolve("Proxy__TSS_GroupManager")).getTs
sGroupUnJailMembers();
363:         require(tssMembers.length > 0, "get tss members in error");
364:
365:         // construct calldata for claimReward call
366:         bytes memory message = abi.encodeWithSelector(
367:             ITssRewardContract.claimReward.selector,
368:             _shouldStartAtElement,
369:             _batch.length,
370:             block.timestamp,
371:             tssMembers
372:         );
373:
374:         // send call data into L2, hardcode address
375:         sendCrossDomainMessage(
376:             address(0x4200000000000000000000000000000000000020),
377:             2000000,
378:             message
379:         );
380:
381:         // emit message
382:         emit DistributeTssReward(
383:             _shouldStartAtElement,
384:             _batch.length,
385:             block.timestamp,
386:             tssMembers
387:         );
388:     }
```

Description

biakia : The param `_batch` in `_distributeTssReward` function is an array that holds all rollup batches. However, only the length of it is used:

```
function _distributeTssReward(bytes32[] memory _batch, uint256 _shouldStartAtElement) internal {
        // get address of tss group member
        address[] memory tssMembers = ITssGroupManager(resolve("Proxy__TSS_GroupManager")).getTssGro
upUnJailMembers();
        require(tssMembers.length > 0, "get tss members in error");

        // construct calldata for claimReward call
        bytes memory message = abi.encodeWithSelector(
            ITssRewardContract.claimReward.selector,
            _shouldStartAtElement,
            _batch.length,
            block.timestamp,
            tssMembers
        );

        // send call data into L2, hardcode address
        sendCrossDomainMessage(
            address(0x4200000000000000000000000000000000000020),
            2000000,
            message
        );

        // emit message
        emit DistributeTssReward(
            _shouldStartAtElement,
            _batch.length,
            block.timestamp,
            tssMembers
        );
    }
```

There is no need to input all rollup batches to the function if it is not intended to be used. It's better to input `_batch.length` instead of `_batch` to save gas.

Recommendation

biakia : Consider below fix in the `_distributeTssReward` function

```
function _distributeTssReward(uint256 _batch_length, uint256 _shouldStartAtElement) internal {
        // get address of tss group member
        address[] memory tssMembers = ITssGroupManager(resolve("Proxy__TSS_GroupManager")).getTssGro
upUnJailMembers();
        require(tssMembers.length > 0, "get tss members in error");

        // construct calldata for claimReward call
        bytes memory message = abi.encodeWithSelector(
            ITssRewardContract.claimReward.selector,
            _shouldStartAtElement,
            _batch_length,
            block.timestamp,
            tssMembers
        );

        // send call data into L2, hardcode address
        sendCrossDomainMessage(
            address(0x4200000000000000000000000000000000000020),
            2000000,
            message
        );

        // emit message
        emit DistributeTssReward(
            _shouldStartAtElement,
            _batch_length,
            block.timestamp,
            tssMembers
        );
    }
```

Client Response

Fixed. Please refer to the fix as follows.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol#L131

```
function _distributeTssReward(uint256 _batch_length, uint256 _shouldStartAtElement) internal {
}
```

MTL-49:Gas Optimization: `firstSupportedInterface` and `secondSupportedInterface` should be declared as constant in `L2StandardERC20` contract `supportsInterface` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	alansh

Code Reference

- code/packages/contracts/contracts/standards/L2StandardERC20.sol#L37-L40

```
37:         bytes4 firstSupportedInterface = bytes4(keccak256("supportsInterface(bytes4)")); // ERC16
5
38:         bytes4 secondSupportedInterface = IL2StandardERC20.l1Token.selector ^
39:             IL2StandardERC20.mint.selector ^
40:             IL2StandardERC20.burn.selector;
```

Description

alansh : These two variable never change, so should be declared as constant instead of recalculating each time.

Recommendation

alansh : Make `firstSupportedInterface` and `secondSupportedInterface` constants.

Client Response

Acknowledged. This optimization will be addressed in future releases.

MTL-50:Gas Optimization: `isEqual` function doesn't need to be public in `TssStakingSlashing`/`TssGroupManager` contract

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L310-L316
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L373-L379

```
310:    function isEqual(bytes memory byteListA, bytes memory byteListB) public pure returns (bool)
{
311:        if (byteListA.length != byteListB.length) return false;
312:        for (uint256 i = 0; i < byteListA.length; i++) {
313:            if (byteListA[i] != byteListB[i]) return false;
314:        }
315:        return true;
316:    }

373:    function isEqual(bytes memory byteListA, bytes memory byteListB) public pure returns (bool)
{
374:        if (byteListA.length != byteListB.length) return false;
375:        for (uint256 i = 0; i < byteListA.length; i++) {
376:            if (byteListA[i] != byteListB[i]) return false;
377:        }
378:        return true;
379:    }
```

Description

alansh : `isEqual` is a pure function that appears in both contracts, it doesn't need to be public, which will incur some additional cost. It's more fit to be put into a library and reuse it in both contracts.

Recommendation

alansh : Put `isEqual` into a library for reuse.

Client Response

Fixed. We have removed the isEqual function in the TssStakingSlashing.sol contract. The latest version of TssGroupManager.sol has addressed the existing issues.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L336

```
    function _isEqual(bytes memory byteListA, bytes memory byteListB) private pure returns (bool) {
        if (byteListA.length != byteListB.length) return false;
        for (uint256 i = 0; i < byteListA.length; i++) {
            if (byteListA[i] != byteListB[i]) return false;
        }
        return true;
    }
```

MTL-51:Gas Optimization: `publicKeyToAddress` is more fit for a library function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L142

```
142:                 msg.sender == ITssGroupManager(tssGroupContract).publicKeyToAddress(_pubKey),
```

Description

alansh : Currently `TssStakingSlashing::stake` calls `ITssGroupManager(tssGroupContract).publicKeyToAddress(_pubKey),` solely for calculating the address of pubkey, this will incur some extra cost for `call` opcode, it's better to put `publicKeyToAddress` in a library function since it's a pure function.

Recommendation

alansh : Put utility function into library to avoid contract call.

Client Response

Fixed. This code has been deprecated.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/libraries/utils/Lib_Address.sol#L8

MTL-52:Gas Optimization: could quit the loop earlier in `TssGroupManager` contract `removeMember` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L166

```
166:                removeActiveTssMembers(i);
```

Description

alansh : Currently the implementation continues to find even if the `_publicKey` has already been found and removed.

Recommendation

alansh : Consider below fix in the `TssGroupManager.removeMember()` function

```
    function removeMember(bytes memory _publicKey) public override onlyStakingSlash {
        for (uint256 i = 0; i < activeTssMembers.length; i++) {
            if (isEqual(activeTssMembers[i], _publicKey)) {
                removeActiveTssMembers(i);
               break;
            }
        }
        delete tssActiveMemberInfo[_publicKey];
    }
```

Client Response

Fixed. Please refer to the fix as follows.

```
    function removeMember(bytes calldata _publicKey) public override onlyOwner {
        require(
            activeTssMembers.length > threshold + 1,
            "TssGroupManager removeMember: active members must more than threshold plus one"
        );
        for (uint256 i = 0; i < activeTssMembers.length; i++) {
            if (_isEqual(activeTssMembers[i], _publicKey)) {
                _removeActiveTssMembers(i);
                break;
            }
        }
        delete tssActiveMemberInfo[_publicKey];
    }
```

MTL-53:Gas Optimization: in `TssStakingSlashing` contract `transformDeposit` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L279
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L307

```
279:        uint256 _exIncome = 0;

307:        totalTransfer = exIncome[slashType] + remainder;
```

Description

alansh : No need to initialize `_exIncome` since it will be assigned anyway, just declaring is enough:

```
uint256 _exIncome = 0;
```

Here `exIncome[slashType]` can be replaced with `_exIncome` to avoid loading from storage:

```
totalTransfer = exIncome[slashType] + remainder;
```

Recommendation

alansh : Apply the gas saving tips.

Client Response

Fixed. There have been significant changes made to code logic, and the mentioned issue no longer exists. Please refer to the following for details.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L277

MTL-54:Gas Optimization: no need to call `_isValidBatchHeader` in `StateCommitmentChain` contract `deleteStateBatch` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	alansh

Code Reference

- code/packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol#L145

```
145:         require(_isValidBatchHeader(_batchHeader), "Invalid batch header.");
```

Description

alansh : Since `_deleteBatch` already calls `_isValidBatchHeader` internally, there's no need to call it twice.

Recommendation

alansh : Remove calling `_isValidBatchHeader` in `deleteStateBatch` function.

Client Response

Acknowledged. We will remove unnecessary require statements in future releases.

MTL-55:Gas Optimization: no need to check all four variables in `TssStakingSlashing` contract `staking` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L125-L128
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L245-L248
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L322-L325

```
125:         for (uint256 i = 0; i < 2; i++) {
126:             require(slashAmount[i] > 0, "have not set the slash amount");
127:             require(exIncome[i] > 0, "have not set the extra income amount");
128:         }

245:         for (uint256 i = 0; i < 2; i++) {
246:             require(slashAmount[i] > 0, "have not set the slash amount");
247:             require(exIncome[i] > 0, "have not set the extra income amount");
248:         }

322:         for (uint256 i = 0; i < 2; i++) {
323:             require(slashAmount[i] > 0, "have not set the slash amount");
324:             require(exIncome[i] > 0, "have not set the extra income amount");
325:         }
```

Description

alansh :

```
function staking(uint256 _amount, bytes calldata _pubKey) public nonReentrant {
        // slashing params check
        for (uint256 i = 0; i < 2; i++) {
            require(slashAmount[i] > 0, "have not set the slash amount");
            require(exIncome[i] > 0, "have not set the extra income amount");
        }
       ...
```

Currently each `staking` call checks `4` variables, but in fact only need to check `1` variable, since `setSlashingParams` guarantees that all `4` variables must be valid together, say, it's impossible for `1` variable to be zero if some variable is positive.

The same for `slash`/`unJail` function.

Recommendation

alansh : Only check 1 variable is enough, this will save some gas for each user.

Client Response

Fixed. Please refer to the fix as follows.

https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L75

```
isSetParam
require(isSetParam,"have not set the slash amount");
```

MTL-56:Gas Optimization: no need to inherit `Initializable` in `TssStakingSlashing` contract

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	alansh

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L11

```
11:     Initializable,
```

Description

alansh : `TssStakingSlashing` inherits `OwnableUpgradeable`, which already inherits `Initializable`.

Recommendation

alansh : Remove `Initializable` from inherit list.

Client Response

Acknowledged.We will resolve the issue in future releases.

MTL-57:Gas Optimization: no need to save `pledgor` into storage in `TssStakingSlashing` contract `staking` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

- code/packages/contracts/contracts/L1/tss/ITssStakingSlashing.sol#L6
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L146

```
6:        address pledgor;

146:            deposits[msg.sender].pledgor = msg.sender;
```

Description

alansh : Since `pledgor` can be calculated by `publicKeyToAddress(_pubKey)`, there's no need to store it at all.

Recommendation

alansh : Don't store redundant info on chain.

Client Response

Fixed.The code related to the pledgor variable has been deprecated.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L24

MTL-58:Gas Optimization: unnecessary variable `sentMessages` in `L2CrossDomainMessenger` contract `sendMessage` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Declined	alansh

Code Reference

- code/packages/contracts/contracts/L2/messaging/L2CrossDomainMessenger.sol#L27
- code/packages/contracts/contracts/L2/messaging/L2CrossDomainMessenger.sol#L72

```
27:    mapping(bytes32 => bool) public sentMessages;

72:        sentMessages[keccak256(xDomainCalldata)] = true;
```

Description

alansh : `BVM_L2ToL1MessagePasser` already has a `sentMessages`, no need for `L2CrossDomainMessenger` to store such a variable.

Recommendation

alansh : Remove `L2CrossDomainMessenger.sentMessages`

Client Response

Declined.The sentMessages variables in the two contracts serve different purposes. One is for cross-chain messages from L1 to L2, while the other is for messages from L2 to L1. The keys generated in their respective maps are distinct and not the same.

MTL-59:Gas Optimization:Cache array length out of the loop

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	biakia

Code Reference

- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L195

```
195:         for (uint256 i = 0; i < quitRequestList.length; i++) {
```

Description

biakia : In `TssStakingSlashing`, the function `quitRequest` will always read the length of the array in each iteration:

```
        // is active member
        for (uint256 i = 0; i < quitRequestList.length; i++) {
            require(quitRequestList[i] != msg.sender, "already in quitRequestList");
        }
```

Since `quitRequestList` is a storage array, there is an extra sload operation which will cost 100 addition gas for each operation.In the code, it is so many issues like this.You can use IDE to search `.length;` for all. Issue List:

- code/packages/contracts/contracts/L1/fraud-proof/Rollup.sol#L145
- code/packages/contracts/contracts/L1/fraud-proof/Rollup.sol#L525
- code/packages/contracts/contracts/L1/fraud-proof/WhiteList.sol#L23
- code/packages/contracts/contracts/L1/fraud-proof/WhiteList.sol#L32
- code/packages/contracts/contracts/L1/fraud-proof/challenge/Challenge.sol#L37
- code/packages/contracts/contracts/L1/fraud-proof/libraries/MPT.sol#L42
- code/packages/contracts/contracts/L1/fraud-proof/libraries/MPT.sol#L238
- code/packages/contracts/contracts/L1/fraud-proof/libraries/MPT.sol#L252
- code/packages/contracts/contracts/L1/fraud-proof/libraries/RLPWriter.sol#L139
- code/packages/contracts/contracts/L1/fraud-proof/libraries/RLPWriter.sol#L193
- code/packages/contracts/contracts/L1/fraud-proof/libraries/RLPWriter.sol#L203
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/EVMTypesLib.sol#L132
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/OneStepProof.sol#L362
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/VerifierHelper.sol#L69
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/VerifierHelper.sol#L99
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/VerifierHelper.sol#L113
- code/packages/contracts/contracts/L1/fraud-proof/verifier/libraries/VerifierHelper.sol#L134
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L67

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L76
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L164
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L178
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L185
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L257
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L262
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L312
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L195
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L309
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L344
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L375
- code/packages/contracts/contracts/L2/predeploys/BVM_GasPriceOracle.sol#L208
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L112
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L143
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L152
- code/packages/contracts/contracts/libraries/eigenda/DataLayrDisclosureLogic.sol#L381
- code/packages/contracts/contracts/libraries/eigenda/DataLayrDisclosureLogic.sol#L406
- code/packages/contracts/contracts/libraries/eigenda/DataLayrDisclosureLogic.sol#L501
- code/packages/contracts/contracts/libraries/eigenda/DataLayrDisclosureLogic.sol#L543
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/core/EigenLayrDelegation.sol#L178
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/core/EigenLayrDelegation.sol#L318
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/governance/Governor.sol#L286
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/governance/Governor.sol#L311
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/governance/Governor.sol#L342
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/investment/InvestmentManager.sol#L234
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/investment/InvestmentManager.sol#L363
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/investment/InvestmentManager.sol#L380
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/investment/InvestmentManager.sol#L477
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/investment/InvestmentManager.sol#L534
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/investment/InvestmentManager.sol#L644
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/investment/InvestmentManager.sol#L708
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/investment/Slasher.sol#L98
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/investment/Slasher.sol#L111
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/investment/Slasher.sol#L124
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/libraries/Merkle.sol#L45
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/libraries/Merkle.sol#L130
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/BLSRegistry.sol#L205
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/VoteWeigherBase.sol#L48
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/DataLayr/DataLayrBombVerifier.sol#L686
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/DataLayr/DataLayrChallengeUtils.sol#L412
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/DataLayr/DataLayrChallengeUtils.sol#L437

- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/DataLayr/DataLayrChallengeUtils.sol#L532
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/DataLayr/DataLayrChallengeUtils.sol#L574
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/DataLayr/DataLayrLowDegreeChallenge.sol#L
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/DataLayr/DataLayrPaymentManager.sol#L94
- code/packages/contracts/contracts/libraries/rlp/Lib_RLPWriter.sol#L129
- code/packages/contracts/contracts/libraries/rlp/Lib_RLPWriter.sol#L184
- code/packages/contracts/contracts/libraries/rlp/Lib_RLPWriter.sol#L194
- code/packages/contracts/contracts/libraries/trie/Lib_MerkleTrie.sol#L143
- code/packages/contracts/contracts/libraries/trie/Lib_MerkleTrie.sol#L239
- code/packages/contracts/contracts/libraries/utils/Lib_BytesUtils.sol#L106
- code/packages/contracts/contracts/libraries/utils/Lib_BytesUtils.sol#L117
- code/packages/contracts/contracts/libraries/utils/Lib_MerkleTree.sol#L128

Recommendation

biakia : Consider caching the length out of the loop:

```
        // is active member
        uint256 length = quitRequestList.length;
        for (uint256 i = 0; i < length; i++) {
            require(quitRequestList[i] != msg.sender, "already in quitRequestList");
        }
```

Client Response

Acknowledged.We will resolve the issue in future releases.

MTL-60:Gas Optimization:Unnecessary Gas Consumption for Zero ERC20 Deposits

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	yekong

Code Reference

- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L177-L230

```
177:     function _initiateERC20Deposit(
178:         address _l1Token,
179:         address _l2Token,
180:         address _from,
181:         address _to,
182:         uint256 _amount,
183:         uint32 _l2Gas,
184:         bytes calldata _data
185:     ) internal {
186:         // When a deposit is initiated on L1, the L1 Bridge transfers the funds to itself for fu
ture
187:         // withdrawals. The use of safeTransferFrom enables support of "broken tokens" which do
not
188:         // return a boolean value.
189:         // slither-disable-next-line reentrancy-events, reentrancy-benign
190:         IERC20(_l1Token).safeTransferFrom(_from, address(this), _amount);
191: 
192:         // Construct calldata for _l2Token.finalizeDeposit(_to, _amount)
193:         bytes memory message;
194:         if (_l1Token == l1BitAddress) {
195:             // Construct calldata for finalizeDeposit call
196:             _l2Token = Lib_PredeployAddresses.BVM_BIT;
197:             message = abi.encodeWithSelector(
198:                 IL2ERC20Bridge.finalizeDeposit.selector,
199:                 address(0x1A4b46696b2bB4794Eb3D4c26f1c55F9170fa4C5),
200:                 Lib_PredeployAddresses.BVM_BIT,
201:                 _from,
202:                 _to,
203:                 _amount,
204:                 _data
205:             );
206: 
207:         } else {
208:             // Construct calldata for finalizeDeposit call
209:             message = abi.encodeWithSelector(
210:                 IL2ERC20Bridge.finalizeDeposit.selector,
211:                 _l1Token,
212:                 _l2Token,
213:                 _from,
214:                 _to,
215:                 _amount,
216:                 _data
```

```
217:             );
218:         }
219:
220:
221:         // Send calldata into L2
222:         // slither-disable-next-line reentrancy-events, reentrancy-benign
223:         sendCrossDomainMessage(l2TokenBridge, _l2Gas, message);
224:
225:         // slither-disable-next-line reentrancy-benign
226:         deposits[_l1Token][_l2Token] = deposits[_l1Token][_l2Token] + _amount;
227:
228:         // slither-disable-next-line reentrancy-events
229:         emit ERC20DepositInitiated(_l1Token, _l2Token, _from, _to, _amount, _data);
230:     }
```

Description

yekong : In the current implementation of the _initiateERC20Deposit function in the contract, it doesn't check if the deposit amount _amount is greater than zero. Even though ERC20 contracts permit transfers of zero value, the execution of this function in the case of zero-value deposits results in unnecessary gas consumption. These operations include the invocation of the safeTransferFrom function, the encoding of the data, the sending of the cross-domain message, the increase of the deposits record and the emission of the ERC20DepositInitiated event. All these operations are executed, consuming gas, but effectively nothing meaningful is accomplished when _amount is zero.

Recommendation

yekong : To prevent unnecessary gas consumption, add a require statement at the beginning of the _initiateERC20Deposit function to ensure that _amount is not zero. Here's how you can do this:

```
require(_amount > 0, "Deposit amount must be greater than zero");
```

This modification will ensure that if _amount is zero, the function will revert at the start, saving unnecessary gas consumption. Furthermore, this check will prevent accidental deposits of zero amount by users.

Client Response

Acknowledged.We will solve the issue in our future updates.

MTL-61:Gas optimization: `TssGroupManager::removeActiveTssMembers` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	biakia

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L318-L324

```
318:     function removeActiveTssMembers(uint256 _index) private {
319:         require(_index < activeTssMembers.length, "index out of bound");
320:         for (uint256 i = _index; i < activeTssMembers.length - 1; i++) {
321:             activeTssMembers[i] = activeTssMembers[i + 1];
322:         }
323:         activeTssMembers.pop();
324:     }
```

Description

biakia : In contract `TssGroupManager`, the function `removeActiveTssMembers` is used to remove a tss member based on its index. It starts from the index position and keeps moving each element of the array to the left position:

```
function removeActiveTssMembers(uint256 _index) private {
        require(_index < activeTssMembers.length, "index out of bound");
        for (uint256 i = _index; i < activeTssMembers.length - 1; i++) {
            activeTssMembers[i] = activeTssMembers[i + 1];
        }
        activeTssMembers.pop();
    }
```

When the array is very large and the index is very small, this operation takes more gas.

Recommendation

biakia : Consider swapping the element in the last position of the array to the index position and popping the last element out:

```
function removeActiveTssMembers(uint256 _index) private {
        require(_index < activeTssMembers.length, "index out of bound");
        activeTssMembers[index] = activeTssMembers[activeTssMembers.length - 1];
        activeTssMembers.pop();
    }
```

Client Response

Fixed. The issue has been fixed as follows.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L344C1-L347C6

```
    function _removeActiveTssMembers(uint256 _index) private {
        activeTssMembers[_index] = activeTssMembers[activeTssMembers.length - 1];
        activeTssMembers.pop();
    }
```

MTL-62:Incorrect error message in `accessors_chain.go`

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	biakia

Code Reference

- code/l2geth/core/rawdb/accessors_chain.go#L54-L58

```
54:func DeleteFPValidatorChallengeCtx(db ethdb.Writer) {
55:     if err := db.Delete(FPValidatorChallengeCtx); err != nil {
56:             log.Crit("Failed to store fp challenge ctx", "err", err)
57:     }
58:}
```

Description

biakia : The function `DeleteFPValidatorChallengeCtx` in `accessors_chain.go` will remove the fp challenge ctx:

```
func DeleteFPValidatorChallengeCtx(db ethdb.Writer) {
        if err := db.Delete(FPValidatorChallengeCtx); err != nil {
                log.Crit("Failed to store fp challenge ctx", "err", err)
        }
}
```

When deletion fails, it logs an error message, however, the error message here is not accurate. The error message should be `Failed to delete fp challenge ctx` instead of `Failed to store fp challenge ctx`.

Recommendation

biakia : Consider correcting the error message:

```
func DeleteFPValidatorChallengeCtx(db ethdb.Writer) {
        if err := db.Delete(FPValidatorChallengeCtx); err != nil {
                log.Crit("Failed to delete fp challenge ctx", "err", err)
        }
}
```

Client Response

Acknowledged.We will improve it in future releases.

MTL-63:Miss emitting event in important state-changing setter functions

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	yekong, biakia

Code Reference

- code/packages/contracts/contracts/da/BVM_EigenDataLayrFee.sol#L24
- code/packages/contracts/contracts/L1/fraud-proof/verifier/VerifierEntry.sol#L49
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L50
- code/packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol#L52
- code/packages/contracts/contracts/L1/fraud-proof/AssertionMap.sol#L60
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L70
- code/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L85
- code/packages/contracts/contracts/L2/predeploys/BVM_SequencerFeeVault.sol#L94-L104
- code/packages/contracts/contracts/da/BVM_EigenDataLayrChain.sol#L119-L190
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/PaymentManager.sol#L122
- code/packages/contracts/contracts/L1/fraud-proof/challenge/Challenge.sol#L157-L159
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/middleware/RegistryBase.sol#L161-L166
- code/packages/contracts/contracts/L1/fraud-proof/challenge/Challenge.sol#L188-L189
- code/packages/contracts/contracts/L1/fraud-proof/challenge/Challenge.sol#L226
- code/packages/contracts/contracts/L1/fraud-proof/challenge/Challenge.sol#L230
- code/packages/contracts/contracts/libraries/eigenda/lib/contracts/governance/Governor.sol#L443-L478

```
24:     function setFeeAddress(address _address) public onlyOwner {

49:     function setVerifier(uint8 verifier, IVerifier impl)

50:     function setStakingSlash(address _address) public onlyOwner {

52:     function setFraudProofWindow(uint256 _fraudProofWindow) public {

60:     function setRollupAddress(address _rollupAddress) public {

70:     function setSendAmountPerYear(uint256 _sendAmountPerYear) public onlyOwner {

85:     function setAddress(address _token, address _tssGroup) public onlyOwner {

94:     function setBurner(address _burner) public onlyOwner{
95:         burner = _burner;
96:     }
97:
98:     function setMinWithdrawalAmount(uint256 _minWithdrawalAmount) public onlyOwner{
99:         minWithdrawalAmount = _minWithdrawalAmount;
100:     }
101:
102:     function setL1FeeWallet(address _l1FeeWallet) public onlyOwner{
103:         l1FeeWallet = _l1FeeWallet;
104:     }

119:     function setFraudProofAddress(address _address) external {
120:         require(msg.sender == sequencer, "Only the sequencer can set fraud proof address unavail
able");
121:         fraudProofWhitelist[_address] = true;
122:     }
123:
124:     /**
125:     * @notice unavailable fraud proof address
126:     * @param _address for fraud proof
127:     */
128:     function unavailableFraudProofAddress(address _address) external {
129:         require(msg.sender == sequencer, "Only the sequencer can remove fraud proof address");
130:         fraudProofWhitelist[_address] = false;
131:     }
132:
```

```
133:     /**
134:     * @notice remove fraud proof address
135:     * @param _address for fraud proof
136:     */
137:     function removeFraudProofAddress(address _address) external {
138:         require(msg.sender == sequencer, "Only the sequencer can remove fraud proof address");
139:         delete fraudProofWhitelist[_address];
140:     }
141:
142:     /**
143:     * @notice update fraud proof period
144:     * @param _fraudProofPeriod fraud proof period
145:     */
146:     function updateFraudProofPeriod(uint256 _fraudProofPeriod) external {
147:         require(msg.sender == sequencer, "Only the sequencer can update fraud proof period");
148:         fraudProofPeriod = _fraudProofPeriod;
149:     }
150:
151:     /**
152:     * @notice update dlsm address
153:     * @param _dataManageAddress dlsm address
154:     */
155:     function updateDataLayrManagerAddress(address _dataManageAddress) external {
156:         require(msg.sender == sequencer, "Only the sequencer can update dlsm address");
157:         dataManageAddress = _dataManageAddress;
158:     }
159:
160:     /**
161:     * @notice update l2 latest store block number
162:     * @param _l2StoredBlockNumber l2 latest block number
163:     */
164:     function updateL2StoredBlockNumber(uint256 _l2StoredBlockNumber) external {
165:         require(msg.sender == sequencer, "Only the sequencer can set latest l2 block number");
166:         l2StoredBlockNumber = _l2StoredBlockNumber;
167:     }
168:
169:     /**
170:     * @notice update l2 latest confirm block number
171:     * @param _l2ConfirmedBlockNumber l2 latest block number
```

```
172:     */
173:     function updateL2ConfirmedBlockNumber(uint256 _l2ConfirmedBlockNumber) external {
174:         require(msg.sender == sequencer, "Only the sequencer can set latest l2 block number");
175:         l2ConfirmedBlockNumber = _l2ConfirmedBlockNumber;
176:     }
177:
178:     /**
179:     * @notice update sequencer address
180:     * @param _sequencer update sequencer address
181:     */
182:     function updateSequencerAddress(address _sequencer) external {
183:         require(msg.sender == sequencer, "Only the sequencer can update sequencer address");
184:         sequencer = _sequencer;
185:     }
186:
187:     function updateReSubmitterAddress(address _reSubmitterAddress) external {
188:         require(msg.sender == sequencer, "Only the sequencer can update re submitter address");
189:         reSubmitterAddress = _reSubmitterAddress;
190:     }

122:     function setAllowance(address allowed, uint256 amount) external {

157:         bisectionHash = ChallengeLib.computeBisectionHash(0, _numSteps);
158:         // TODO: consider emitting a different event?
159:         currentBisected = BisectedStore(startStateHash, checkStateHash, endStateHash, block.numb
er, block.timestamp, 0, _numSteps);

161:     function setMinimumStakeFirstQuorum(uint128 _minimumStakeFirstQuorum) external onlyRepositor
yGovernance {
162:         minimumStakeFirstQuorum = _minimumStakeFirstQuorum;
163:     }
164:
165:     /// @notice Adjusts the `minimumStakeSecondQuorum` -- i.e. the node stake (weight) requireme
nt for inclusion in the 2nd quorum.
166:     function setMinimumStakeSecondQuorum(uint128 _minimumStakeSecondQuorum) external onlyReposit
oryGovernance {

188:         bisectionHash = ChallengeLib.computeBisectionHash(challengedSegmentStart, challengedSegm
entLength);
189:         currentBisected = BisectedStore(bisection[0], bisection[1], bisection[2], blo-
```

```
ck.number, block.timestamp, challengedSegmentStart, challengedSegmentLength);

226:    function setRollback() public {

230:        rollback = true;

443:    function setQuorumsAndThresholds(
444:        uint16 _firstQuorumPercentage,
445:        uint16 _secondQuorumPercentage,
446:        uint16 _proposalThresholdFirstQuorumPercentage,
447:        uint16 _proposalThresholdSecondQuorumPercentage
448:    ) external onlyTimelock {
449:        _setQuorumsAndThresholds(
450:            _firstQuorumPercentage,
451:            _secondQuorumPercentage,
452:            _proposalThresholdFirstQuorumPercentage,
453:            _proposalThresholdSecondQuorumPercentage
454:        );
455:    }
456:
457:    function _setQuorumsAndThresholds(
458:        uint16 _firstQuorumPercentage,
459:        uint16 _secondQuorumPercentage,
460:        uint16 _proposalThresholdFirstQuorumPercentage,
461:        uint16 _proposalThresholdSecondQuorumPercentage
462:    ) internal {
463:        require(_firstQuorumPercentage > 0 && _firstQuorumPercentage < 100, "bad _firstQuorumPer
centage");
464:        require(_secondQuorumPercentage > 0 && _secondQuorumPercentage < 100, "bad _secondQuorum
Percentage");
465:        require(
466:            _proposalThresholdFirstQuorumPercentage > 0 && _proposalThresholdFirstQuorumPercenta
ge < 100,
467:            "bad _proposalThresholdFirstQuorumPercentage"
468:        );
469:        require(
470:            _proposalThresholdSecondQuorumPercentage > 0 && _proposalThresholdSecondQuorumPercen
tage < 100,
471:            "bad _proposalThresholdSecondQuorumPercentage"
472:        );
473:
474:        firstQuorumPercentage = _firstQuorumPercentage;
475:        secondQuorumPercentage = _secondQuorumPercentage;
```

```
476:         proposalThresholdFirstQuorumPercentage = _proposalThresholdFirstQuorumPercentage;
477:         proposalThresholdSecondQuorumPercentage = _proposalThresholdSecondQuorumPercentage;
478:     }
```

Description

yekong : Miss emitting event in important state-changing setter functions. example:

```
function setFraudProofAddress(address _address) external {
        require(msg.sender == sequencer, "Only the sequencer can set fraud proof address unavailabl
e");
        fraudProofWhitelist[_address] = true;
    }
```

biakia : Functions that update state variables should emit relevant events as notifications.

Recommendation

yekong : Add emitting event.

Due to the large quantity, it is recommended that the project party check again to prevent my marking omissions

biakia : Recommend adding events for state-changing actions, and emitting them in their relevant functions.

Client Response

Acknowledged.We will resolve it in future releases.

MTL-64:Redundant function

Category	Severity	Client Response	Contributor
Logical	Informational	Acknowledged	biakia

Code Reference

- code/l2geth/core/rawdb/rollup_indexes.go#L94-L101
- code/l2geth/core/rawdb/rollup_indexes.go#L113-L120

```
94:func ReadLatestEigenBatchIndex(db ethdb.KeyValueReader) *uint64 {
95:     data, _ := db.Get(eigenBatchKey)
96:     if len(data) == 0 {
97:             return nil
98:     }
99:     ret := new(big.Int).SetBytes(data).Uint64()
100:    return &ret
101:}

113:func ReadEigenBatchIndex(db ethdb.KeyValueReader) *uint64 {
114:    data, _ := db.Get(eigenBatchKey)
115:    if len(data) == 0 {
116:            return nil
117:    }
118:    ret := new(big.Int).SetBytes(data).Uint64()
119:    return &ret
120:}
```

Description

biakia : In `rollup_indexes.go`, the implementation of the functions `ReadLatestEigenBatchIndex` and `ReadEigenBatchIndex` is the same:

```
func ReadLatestEigenBatchIndex(db ethdb.KeyValueReader) *uint64 {
        data, _ := db.Get(eigenBatchKey)
        if len(data) == 0 {
                return nil
        }
        ret := new(big.Int).SetBytes(data).Uint64()
        return &ret
}

func ReadEigenBatchIndex(db ethdb.KeyValueReader) *uint64 {
        data, _ := db.Get(eigenBatchKey)
        if len(data) == 0 {
                return nil
        }
        ret := new(big.Int).SetBytes(data).Uint64()
        return &ret
}
```

There is no need to implement two different functions for the same logic.

Recommendation

biakia : Consider keeping just one function.

Client Response

Acknowledged.We will resolve it in future releases.

MTL-65:Unused variable: `BVM_GasPriceOracle::sccAddress`

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	biakia

Code Reference

- code/packages/contracts/contracts/L2/predeploys/BVM_GasPriceOracle.sol#L38-L39

```
38:     // sccAddress l1 sccAddress
39:     address public sccAddress;
```

Description

biakia : In the contract `BVM_GasPriceOracle` the variable `sccAddress` is neither initialized nor used, it is recommended to remove this variable to save gas.

Recommendation

biakia : Consider removing the variable `sccAddress` in `BVM_GasPriceOracle` if it is not intended to be used.

Client Response

Acknowledged. This is native code from Optimism and we haven't made any modifications to it. We will consider fixing it in the future.

MTL-66:Use `call` instead of `transfer` to transfer ETH

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	yekong, biakia, Secure3

Code Reference

- code/packages/contracts/contracts/libraries/eigenda/lib/test/mocks/LiquidStakingToken.sol#L22
- code/packages/contracts/contracts/L2/tokens/wbit.sol#L43
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L115
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L156
- code/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L239-L251

```
22:         payable(msg.sender).transfer(_amount);

43:         payable(msg.sender).transfer(_amount);

115:             addr.transfer(sendAmount);

156:                 addr.transfer(sendAmount);

239:     function finalizeETHWithdrawal(
240:         address _from,
241:         address _to,
242:         uint256 _amount,
243:         bytes calldata _data
244:     ) external onlyFromCrossDomainAccount(l2TokenBridge) {
245:         // slither-disable-next-line reentrancy-events
246:         (bool success, ) = _to.call{ value: _amount }(new bytes(0));
247:         require(success, "TransferHelper::safeTransferETH: ETH transfer failed");
248:
249:         // slither-disable-next-line reentrancy-events
250:         emit ETHWithdrawalFinalized(_from, _to, _amount, _data);
251:     }
```

Description

yekong : With the upgrade of the network, the gas of the underlying operation may change. Currently, it is recommended to use call for ether transfers

biakia : It is not recommended to use Solidity's `transfer()` function for transferring Ether, since some contracts may not be able to receive the funds. Those functions forward only a fixed amount of gas (2300 specifically) and the receiving

contracts may run out of gas before finishing the transfer. Also, EVM instructions' gas costs may increase in the future. Thus, some contracts that can receive now may stop working in the future due to the gas limitation.

Secure3 : In `L1StandardBridge` contract , it use `finalizeETHWithdrawal` function to withdraw some eth to a specific address.However, at the low level, it use `call` method to do the transfer, which could cause Reentrancy if the target address is malicious.

Consider below POC contract

```
contract hack {
    function reenter(address _from, address _to, uint256 _amount, bytes calldata _data) external {
        victim.finalizeETHWithdrawal(_from, _to,  _amount, calldata _data);
    }
}
```

Recommendation

yekong : it is recommended to use `call` instead of `transfer`

About why transfer is not recommended, you can refer to this article： https://consensys.net/diligence/blog/2019/09/stop-using-soliditys-transfer-now/

biakia : We recommend using the `Address.sendValue()` function from OpenZeppelin.

Since `Address.sendValue()` may allow reentrancy, we also recommend guarding against reentrancy attacks by utilizing the Checks-Effects-Interactions Pattern or applying OpenZeppelin ReentrancyGuard.

Secure3 : Use the Checks-Effects-Interactions best practice and make all state changes before calling external contracts. Also, consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.

Consider below fix in the function

```
function finalizeETHWithdrawal(
        address _from,
        address _to,
        uint256 _amount,
        bytes calldata _data
    ) external onlyFromCrossDomainAccount(l2TokenBridge) nonReentrant{
        // slither-disable-next-line reentrancy-events
        (bool success, ) = _to.call{ value: _amount }(new bytes(0));
        require(success, "TransferHelper::safeTransferETH: ETH transfer failed");

        // slither-disable-next-line reentrancy-events
        emit ETHWithdrawalFinalized(_from, _to, _amount, _data);
    }
```

Client Response

Fixed.

https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/co-

ntracts/L1/messaging/L1StandardBridge.sol#L247
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L127
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L199

MTL-67:Use `calldata` instead of `memory`

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	biakia

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L58
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L91
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L147
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L155
- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L163

```
58:    function setTssGroupMember(uint256 _threshold, bytes[] memory _batchPublicKey)

91:    function setGroupPublicKey(bytes memory _publicKey, bytes memory _groupPublicKey)

147:    function memberJail(bytes memory _publicKey) public override onlyStakingSlash {

155:    function memberUnJail(bytes memory _publicKey) public override onlyStakingSlash {

163:    function removeMember(bytes memory _publicKey) public override onlyStakingSlash {
```

Description

biakia : It's better to use `calldata` instead of `memory` for function parameters that represent variables that will not be modified.

Recommendation

biakia : Consider using `calldata` instead of `memory` to save gas.

Client Response

Fixed.https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L14

```
function setTssGroupMember(uint256 _threshold, bytes[] calldata _batchPublicKey)
function setGroupPublicKey(bytes calldata _publicKey, bytes calldata _groupPublicKey)

function memberJail(bytes calldata _publicKey) public override onlyStakingSlash {
    tssActiveMemberInfo[_publicKey].status = MemberStatus.jail;
}

function memberUnJail(bytes calldata _publicKey) public override onlyStakingSlash {
    tssActiveMemberInfo[_publicKey].status = MemberStatus.unJail;
}

function removeMember(bytes calldata _publicKey) public override onlyOwner {
```

MTL-68: `dust` / `accu` calculation can be simplified in `TssRewardContract` contract `claimReward` function

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	alansh

Code Reference

- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L110-L120
- code/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L150-L161

```
110:         dust = 0;
111:         sendAmount = batchAmount.div(_tssMembers.length);
112:         for (uint256 j = 0; j < _tssMembers.length; j++) {
113:             address payable addr = payable(_tssMembers[j]);
114:             accu = accu.add(sendAmount);
115:             addr.transfer(sendAmount);
116:         }
117:         uint256 reserved = batchAmount.sub(accu);
118:         if (reserved > 0) {
119:             dust = dust.add(reserved);
120:         }

150:             dust = 0;
151:             sendAmount = batchAmount.div(_tssMembers.length);
152:             for (uint256 j = 0; j < _tssMembers.length; j++) {
153:                 address payable addr = payable(_tssMembers[j]);
154:                 accu = accu.add(sendAmount);
155:                 totalAmount = totalAmount.sub(sendAmount);
156:                 addr.transfer(sendAmount);
157:             }
158:             uint256 reserved = batchAmount.sub(accu);
159:             if (reserved > 0) {
160:                 dust = dust.add(reserved);
161:             }
```

Description

alansh :

```
        dust = 0;
        sendAmount = batchAmount.div(_tssMembers.length);
        for (uint256 j = 0; j < _tssMembers.length; j++) {
            address payable addr = payable(_tssMembers[j]);
            accu = accu.add(sendAmount);
            addr.transfer(sendAmount);
        }
        uint256 reserved = batchAmount.sub(accu);
        if (reserved > 0) {
            dust = dust.add(reserved);
        }
```

can be simplified as

```
        sendAmount = batchAmount.div(_tssMembers.length);
        for (uint256 j = 0; j < _tssMembers.length; j++) {
            address payable addr = payable(_tssMembers[j]);
            addr.transfer(sendAmount);
        }
        accu = sendAmount * _tssMembers.length
        dust = batchAmount.sub(accu);
```

Similarly for `claimRewardByBlock` function.

Recommendation

alansh : Apply the fix .

Client Response

Fixed.The TssRewardContract.sol contract has undergone significant logic adjustments, and the "dust" and "accu" variables are no longer in use.
https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L2/predeploys/TssRewardContract.sol#L22

MTL-69:missing calling `__ReentrancyGuard_init` in `TssGroupManager` contract `initialize` function

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	alansh

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L38-L39

```
38:     function initialize() public initializer {
39:         __Ownable_init();
```

Description

alansh : `TssStakingSlashing::initialize` calls `__ReentrancyGuard_init`, but `TssGroupManager::initialize` doesn't. It's better to keep consistent.

Recommendation

alansh : Add a call to `__ReentrancyGuard_init` in `TssGroupManager::initialize`

Client Response

Acknowledged. For the TssGroupManager.sol contract, the __ReentrancyGuard_init() function was not called during contract initialization. However, there are no security concerns associated with this omission. We will address this issue in future releases.

MTL-70:should check `_publicKey` exist before updating `tssActiveMemberInfo` in `TssGroupManager` contract `memberUnJail` function

Category	Severity	Client Response	Contributor
Logical	Informational	Declined	alansh

Code Reference

- code/packages/contracts/contracts/L1/tss/TssGroupManager.sol#L148

```
148:         tssActiveMemberInfo[_publicKey].status = MemberStatus.jail;
```

Description

alansh : `memberUnJail` can be call from `TssStakingSlashing` by a single member, should check `_publicKey` is in `tssActiveMemberInfo` otherwise `tssActiveMemberInfo` may be filled with arbitrary `_publicKey`.

Recommendation

alansh : check `_publicKey` in `tssActiveMemberInfo` before update.

Client Response

Declined. The reported issue does not exist. In the TssStakingSlashing.sol contract, we have a require statement require(isJailed(msg.sender)) before calling memberUnJail, which ensures that _publicKey is present and valid. https://github.com/mantlenetworkio/mantle/blob/0b7733c10a44e93edbf49b27e056ebe2e5279c63/packages/contracts/contracts/L1/tss/TssStakingSlashing.sol#L317C1-L326C6

```
    function unJail() public {
        // slashing params check
        require(isSetParam, "have not set the slash amount");
        require(isJailed(msg.sender), "An unjailed user doesn't need to call this method");

        uint256 totalBalance = _tokenBalance();

        require((delegation.operatorShares(msg.sender, this) * totalBalance) / totalShares >= slashA
mount[1], "Insufficient balance");
        ITssGroupManager(tssGroupContract).memberUnJail(operators[msg.sender]);
    }
```

MTL-71:should rename `RANDOM` to `PREVRANDAO` according to eip-4399

Category	Severity	Client Response	Contributor
Language Specific	Informational	Acknowledged	Secure3

Code Reference

- code/l2geth/core/vm/evm.go#L96

```
96:     Random      *common.Hash   // Provides information for RANDOM
```

Description

Secure3 : According to eip-4399, `Supplant DIFFICULTY opcode with PREVRANDAO`, so you should rename `RANDOM` to `PREVRANDAO` in code/l2geth/core/vm/evm.go#L96.

Recommendation

Secure3 : rename `RANDOM` to `PREVRANDAO` according to eip-4399

Consider below fix in the `sample.test()` function

```
type BlockContext struct {
        // CanTransfer returns whether the account contains
        // sufficient ether to transfer the value
        CanTransfer CanTransferFunc
        // Transfer transfers ether from one account to the other
        Transfer TransferFunc
        // GetHash returns the hash corresponding to n
        GetHash GetHashFunc

        // Block information
        Coinbase    common.Address // Provides information for COINBASE
        GasLimit    uint64         // Provides information for GASLIMIT
        BlockNumber *big.Int       // Provides information for NUMBER
        Time        *big.Int       // Provides information for TIME
        Difficulty  *big.Int       // Provides information for DIFFICULTY
        BaseFee     *big.Int       // Provides information for BASEFEE
        Random      *common.Hash   // Provides information for PREVRANDAO
}
```

Client Response

Acknowledged. This is native code from Optimism and we haven't made any modifications to it. We will fix it in the future.

MTL-72:uses assembly in libraries

Category	Severity	Client Response	Contributor
Code Style	Informational	Declined	Secure3

Code Reference

- code/packages/contracts/contracts/libraries/rlp/Lib_RLPReader.sol#L44-L46
- code/packages/contracts/contracts/libraries/rlp/Lib_RLPReader.sol#L144-L163

```
44:         assembly {
45:             ptr := add(_in, 32)
46:         }

144:     function readBytes32(RLPItem memory _in) internal pure returns (bytes32) {
145:         require(_in.length <= 33, "Invalid RLP bytes32 value.");
146:
147:         (uint256 itemOffset, uint256 itemLength, RLPItemType itemType) = _decodeLength(_in);
148:
149:         require(itemType == RLPItemType.DATA_ITEM, "Invalid RLP bytes32 value.");
150:
151:         uint256 ptr = _in.ptr + itemOffset;
152:         bytes32 out;
153:         assembly {
154:             out := mload(ptr)
155:
156:             // Shift the bytes over to match the item size.
157:             if lt(itemLength, 32) {
158:                 out := div(out, exp(256, sub(32, itemLength)))
159:             }
160:         }
161:
162:         return out;
163:     }
```

Description

Secure3 : Solidity defines an assembly language, which can be embedded in the source code of solidity and used in the way of inline assembly. The inline assembly provided by solidity is a language close to the EVM alternative, allowing for use with solidity. Because EVM is stackable, sometimes it is difficult to locate the stack. Solidty's inline assembly provides

alternative features to solve various problems caused by handwritten underlying code. However, inline assembly language does not have many security mechanisms provided by solidity, and it is difficult to write inline compilation language.

```
contract ERC777Hack {
   function toRLPItem(bytes memory _in) internal pure returns (RLPItem memory) {
        uint256 ptr;
        assembly {
            ptr := add(_in, 32)
        }

        return RLPItem({ length: _in.length, ptr: ptr });
    }
```

Recommendation

Secure3 : When writing smart contracts, try not to use inline assembly language.

Client Response

Declined.The usage of inline assembly logic in this context is straightforward and does not imply any security risks. The code does not have logic issues.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. 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.