Matic

Staking Contract

Security Assessment

February 2nd, 2021

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are 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 review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	**Matic Staking Contract**
Description	Smart contracts regarding staking, delegation and validator reward distribution
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. bf1f1ea9e7fbc1efee575107f557bcb450673e92

Audit Summary

Delivery Date	**January 20th, 2021**
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	January 6th, 2021 - January 20th, 2021

Vulnerability Summary

Total Issues	**10**
Total Critical	0
Total Major	1
Total Medium	0
Total Minor	0
Total Informational	9

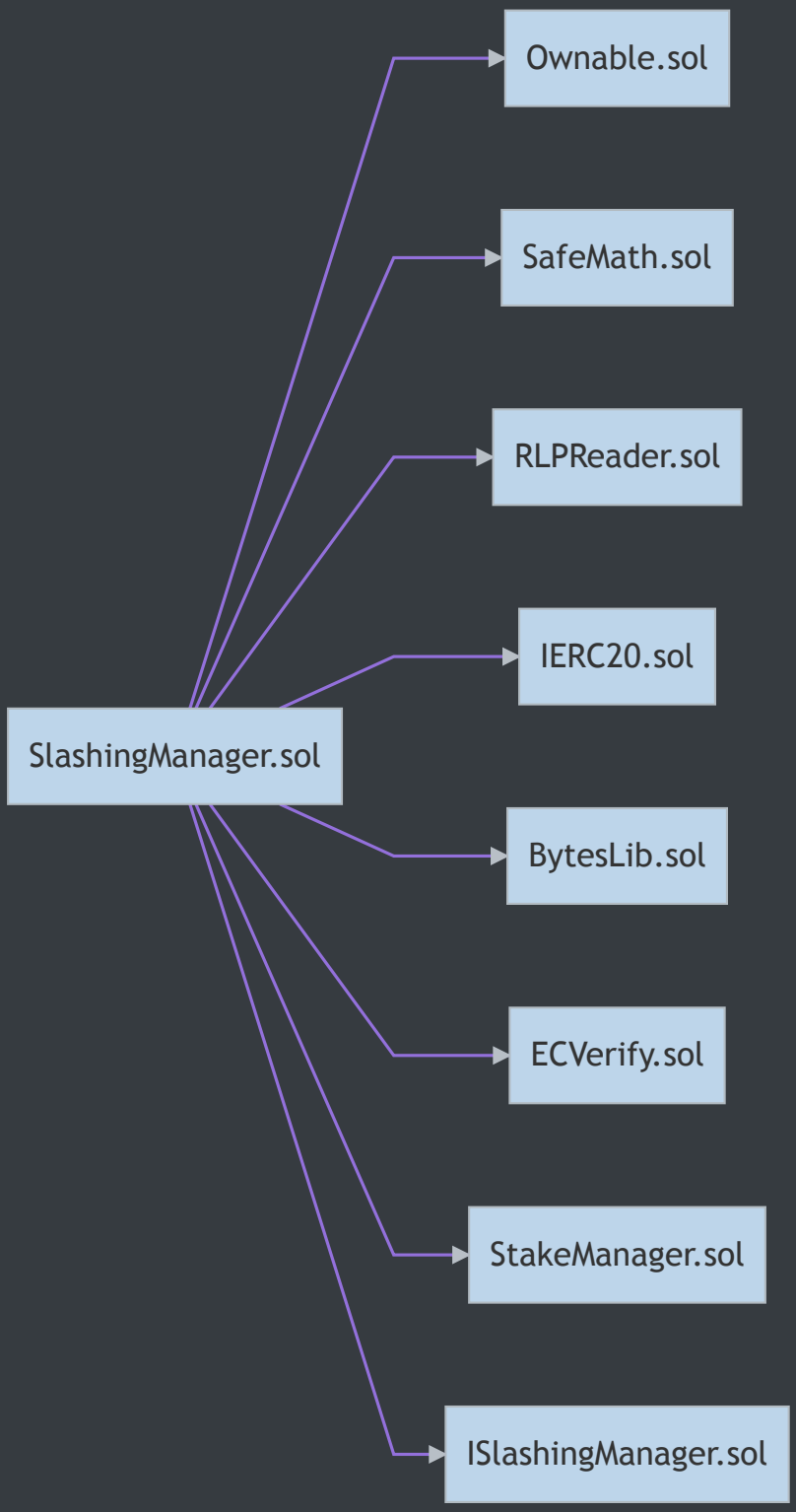
Executive Summary

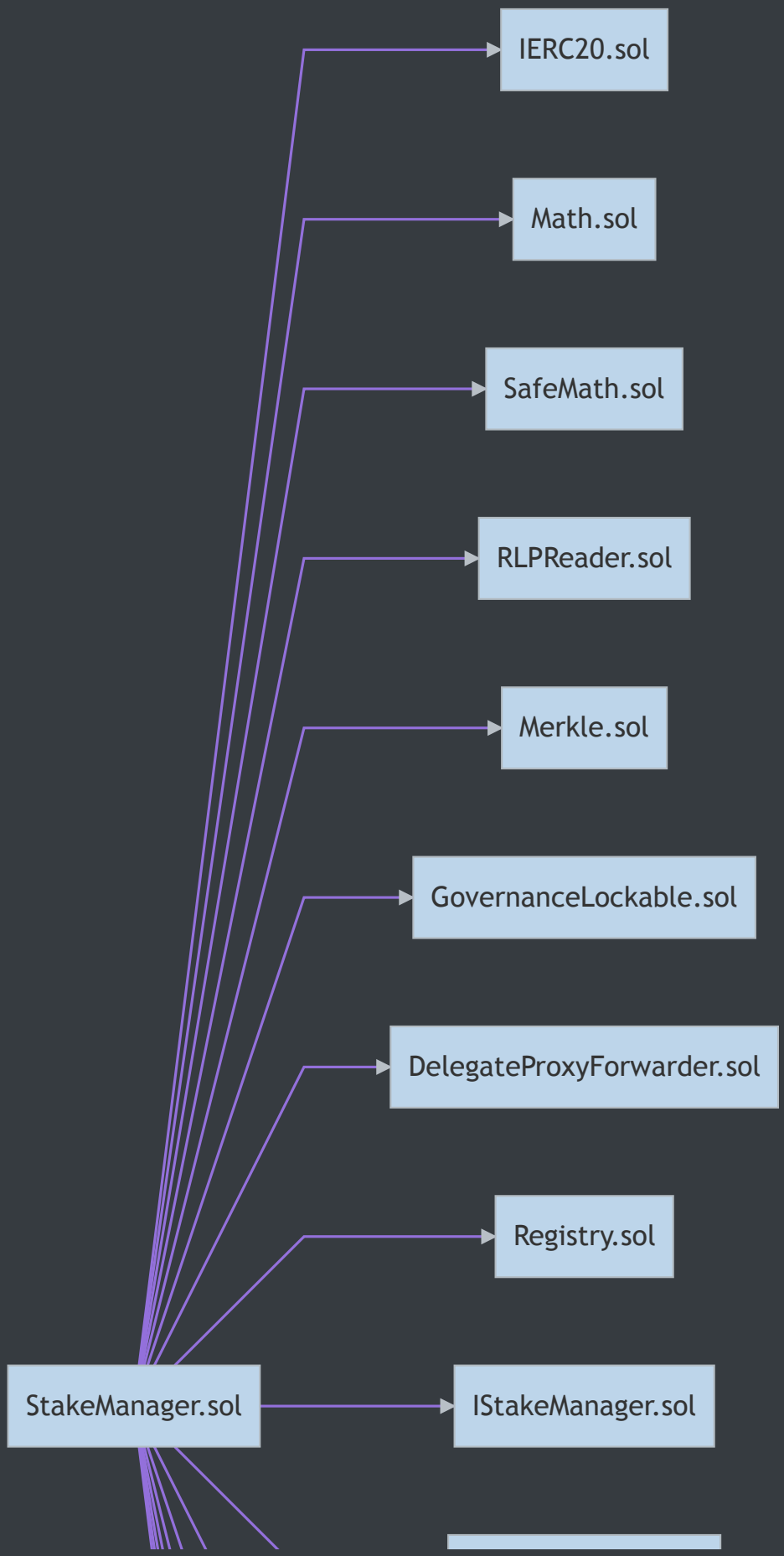
This section will represent the summary of the whole audit process once it has concluded.

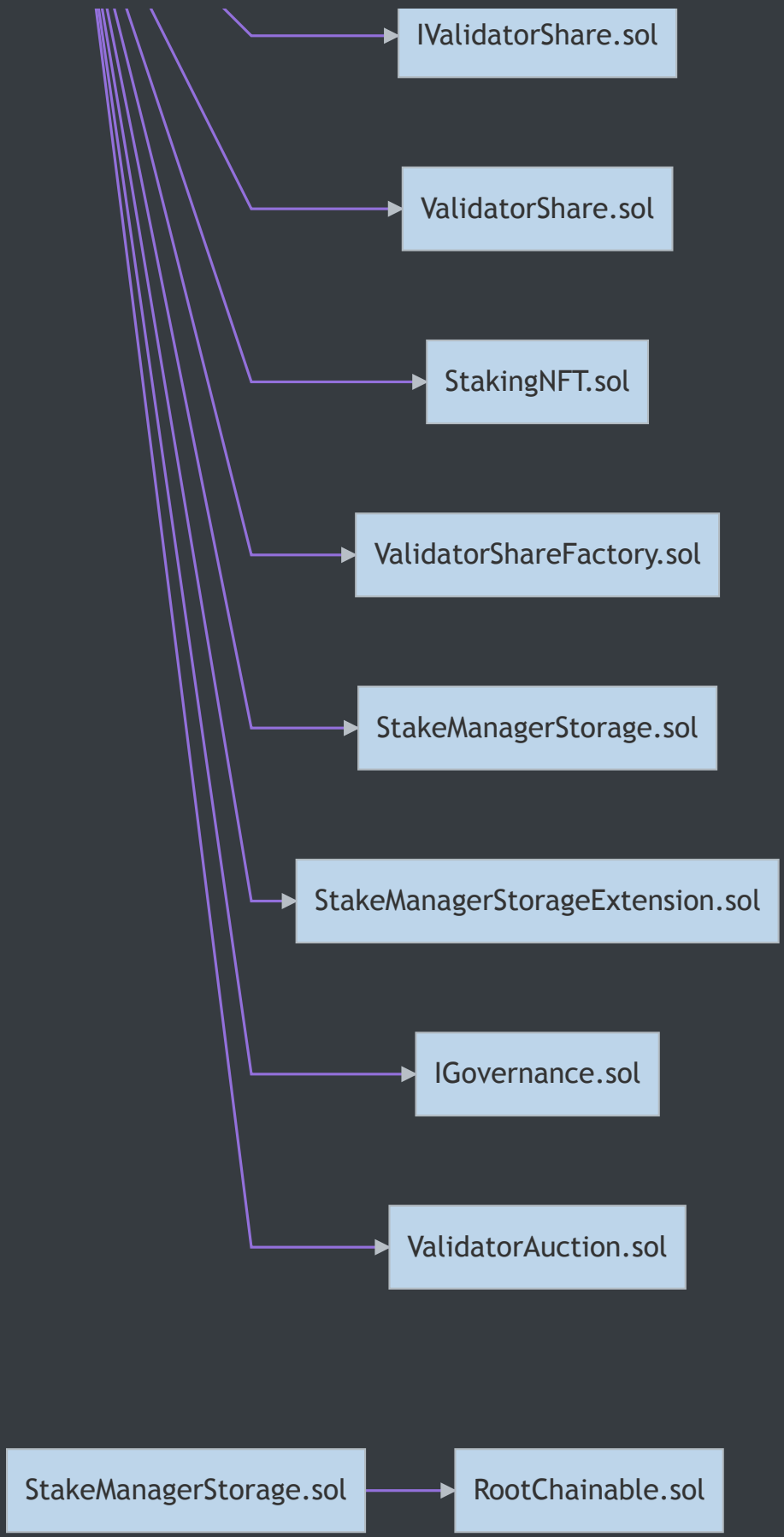
Files In Scope

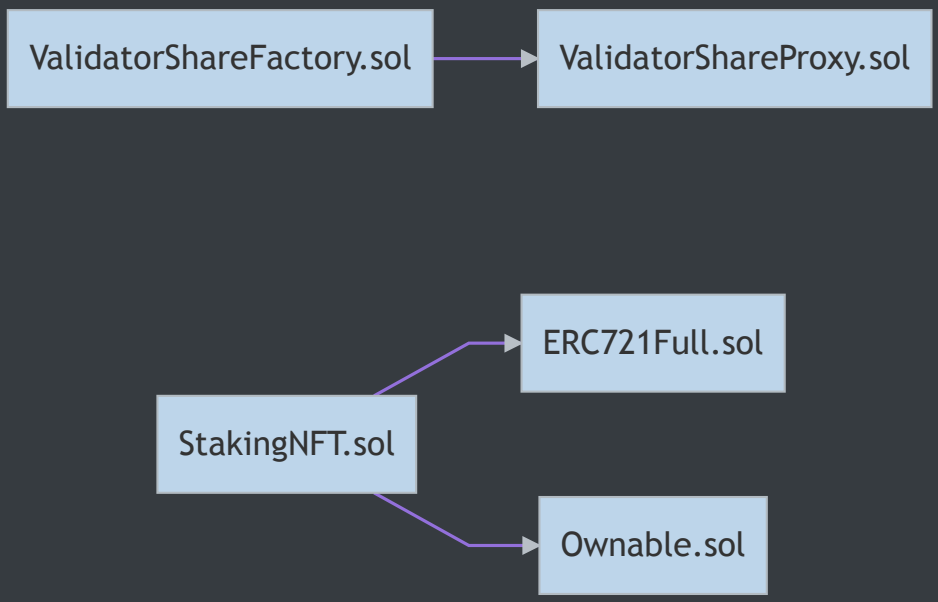
ID	Contract	Location
EHB	EventsHub.sol	contracts/staking/EventsHub.sol
EHP	EventsHubProxy.sol	contracts/staking/EventsHubProxy.sol
ISM	IStakeManager.sol	contracts/staking/stakeManager/IStakeManager.sol
IVS	IValidatorShare.sol	contracts/staking/validatorShare/IValidatorShare.sol
CON	ISlashingManager.sol	contracts/staking/slashing/ISlashingManager.sol
SNF	StakingNFT.sol	contracts/staking/stakeManager/StakingNFT.sol
SIO	StakingInfo.sol	contracts/staking/StakingInfo.sol
SMR	StakeManager.sol	contracts/staking/stakeManager/StakeManager.sol
SME	SlashingManager.sol	contracts/staking/slashing/SlashingManager.sol
SMP	StakeManagerProxy.sol	contracts/staking/stakeManager/StakeManagerProxy.sol
SMS	StakeManagerStorage.sol	contracts/staking/stakeManager/StakeManagerStorage.sol
SSE	StakeManagerStorageExtension.sol	contracts/staking/stakeManager/StakeManagerStorageExtension.sol
VSE	ValidatorShare.sol	contracts/staking/validatorShare/ValidatorShare.sol
VAN	ValidatorAuction.sol	contracts/staking/stakeManager/ValidatorAuction.sol
VSP	ValidatorShareProxy.sol	contracts/staking/validatorShare/ValidatorShareProxy.sol
VSF	ValidatorShareFactory.sol	contracts/staking/validatorShare/ValidatorShareFactory.sol
VSS	ValidatorShareStorageExtension.sol	contracts/staking/validatorShare/ValidatorShareStorageExtension.sol

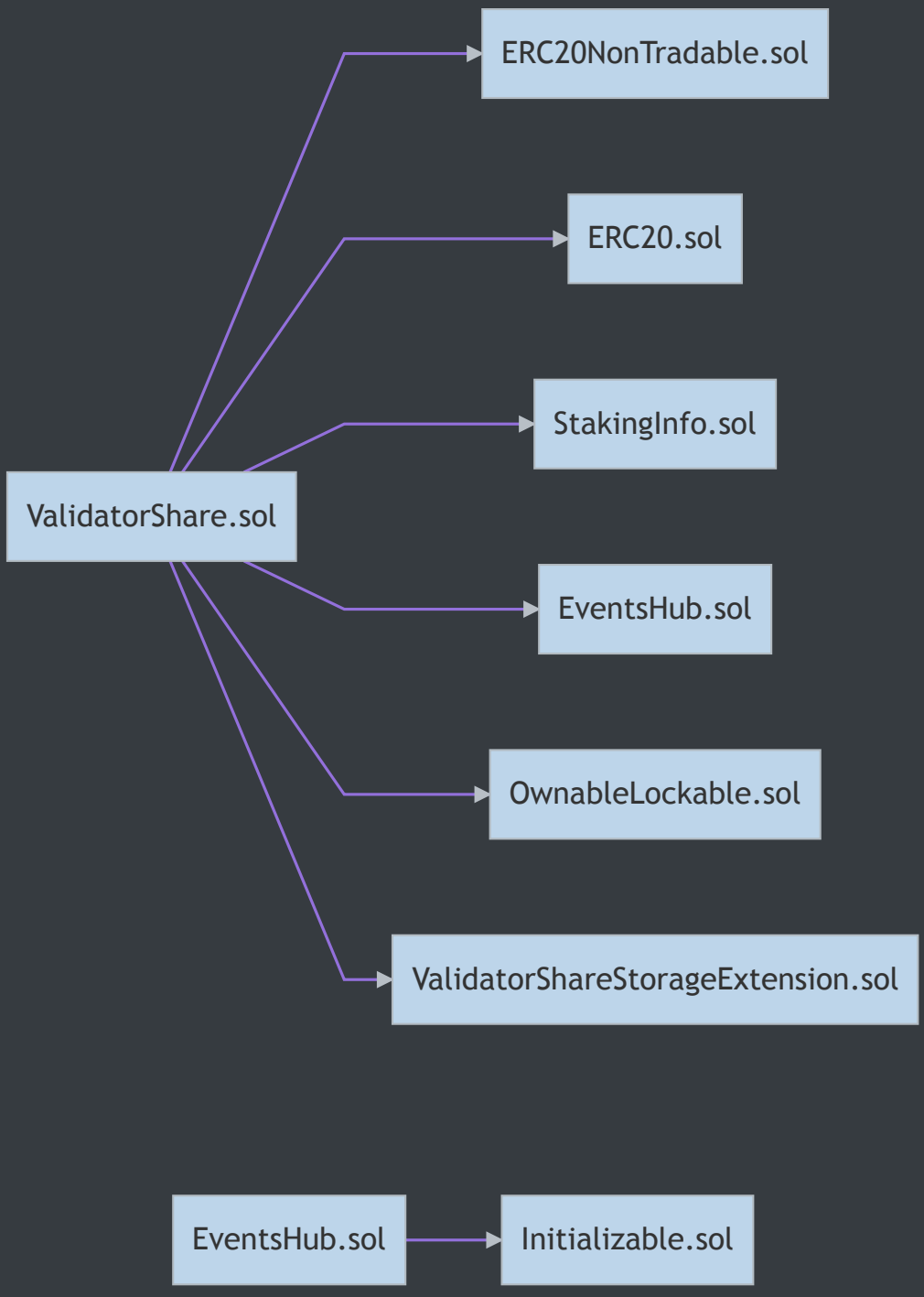
File Dependency Graph (BETA)

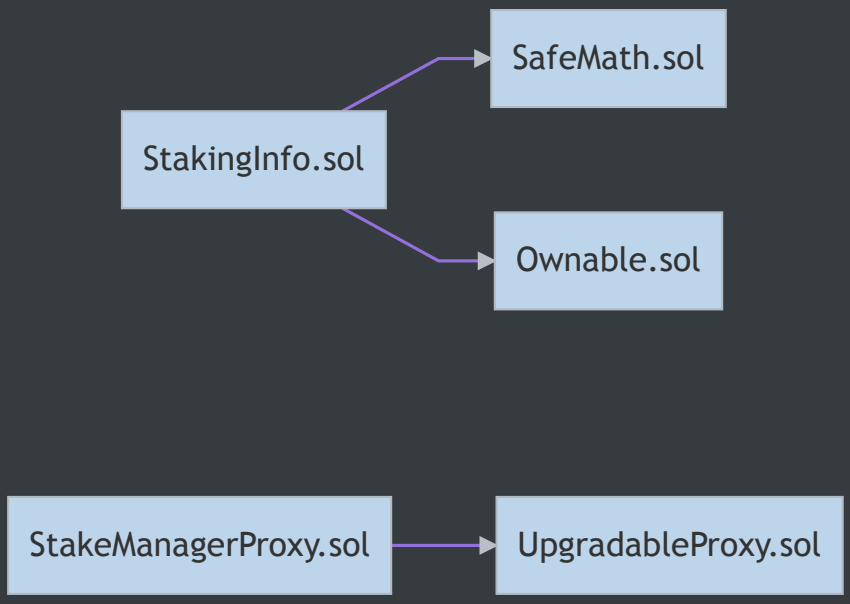



IERC20.sol
Math.sol
SafeMath.sol
RLPReader.sol
Merkle.sol
GovernanceLockable.sol
DelegateProxyForwarder.sol
Registry.sol
StakeManager.sol
IStakeManager.sol

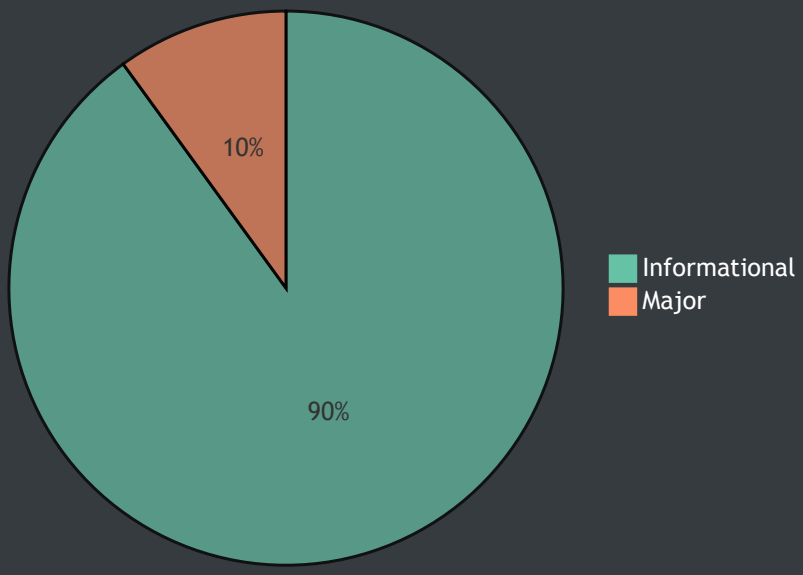

IValidatorShare.sol
ValidatorShare.sol
StakingNFT.sol
ValidatorShareFactory.sol
StakeManagerStorage.sol
StakeManagerStorageExtension.sol
IGovernance.sol
ValidatorAuction.sol
StakeManagerStorage.sol
RootChainable.sol


ValidatorShareFactory.sol
ValidatorShareProxy.sol
StakingNFT.sol
ERC721Full.sol
Ownable.sol


ERC20NonTradable.sol
ERC20.sol
StakingInfo.sol
ValidatorShare.sol
EventsHub.sol
OwnableLockable.sol
ValidatorShareStorageExtension.sol
EventsHub.sol
Initializable.sol


SafeMath.sol
StakingInfo.sol
Ownable.sol
StakeManagerProxy.sol
UpgradableProxy.sol

Findings



ID	Title	Type	Severity	Resolved
SMR-01	Confusing function name	Control Flow	Informational	✓
SMR-02	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
SMR-03	Jailed validator can appear twice in the unsigned validator list	Logical Issue	Major	✓
SMR-04	Confusing variable name	Logical Issue	Informational	✓
SMR-05	Function visibility	Gas Optimization	Informational	✓
SMR-06	Gas optimization in element insertion	Gas Optimization	Informational	✓
SMR-07	Incompatible array resize with newer Solidity version	Language Specific	Informational	⟳
SMR-08	Incorrect data allocation	Language Specific	Informational	✓
VSE-01	Unnecessary computation	Logical Issue	Informational	✓
VSE-02	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓

SMR-01: Confusing function name

Type	Severity	Location
Control Flow	Informational	StakeManager.sol L759

Description:

By standard this should be a pure function that derives the address from public key but in our code it also has additional functionality which checks whether or not the address appears in the signers array. I think we should rename the function to reflect that.

Recommendation:

We recommend renaming the function to reflect its functionality.

Alleviation:

The team has followed the recommendation and changed `_pubToAddress()` to `_getAndAssertSigner()`

SMR-02: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	StakeManager.sol L199, L205, L210

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The team has followed the recommendation.

SMR-03: Jailed validator can appear twice in the unsigned validator list

Type	Severity	Location
Logical Issue	Major	StakeManager.sol L598

Description:

In the main loop of the function `checkSignatures()` if the validator status is locked it is added to the unsigned validator array, but the main validator id is not updated, hence it will be added again to the unsigned validator array in the next iteration. Unlike unstaked validator this can happen because jailed validator's signer still appears in the array `signers`.

Recommendation:

We recommend update the validator index in the `status == Status.Locked` branch or remove the signer from the array `signers` when the corresponding validator is jailed.

Alleviation:

This will be fixed when slashing becomes functional.

SMR-04: Confusing variable name

Type	Severity	Location
Logical Issue	Informational	StakeManager.sol L1121

Description:

In the function `_updateSigner` the second variable is `signerToDelete` even though we want to add it to our array of signers.

Recommendation:

We recommend changing the variable name accordingly.

Alleviation:

The team has followed the recommendation.

SMR-05: Function visibility

Type	Severity	Location
Gas Optimization	● Informational	StakeManager.sol L520, L524

Description:

Functions `increaseValidatorDelegatedAmount()` and `decreaseValidatorDelegatedAmount` can only be called by the corresponding delegation contract even though this contract (ValidatorShare.sol) has no method to call it. Throughout the codebase they are only used in `updateValidatorState()` so I believe it can be changed to private or internal.

Recommendation:

We recommend changing the function visibility as it also makes deployment cheaper.

Alleviation:

The team has followed the recommendation and changed the function visibility.

SMR-06: Gas optimization in element insertion

Type	Severity	Location
Gas Optimization	Informational	StakeManager.sol L1106

Description:

In function `_insertSigner()` we want to add a new element into the function and keep the order of elements in it. When we find the index for the new element we assign it at line 1115, but in the beginning at 1107 we push it temporarily to the end of the array so we can make an `if` check to see whether it is already in the correct place. This saves gas because assignment to the same value costs gas as a normal assignment.

Recommendation:

We recommend adding the if check to the function.

Alleviation:

The team has followed the recommendation.

SMR-07: Incompatible array resize with newer Solidity version

Type	Severity	Location
Language Specific	Informational	StakeManager.sol L1126

Description:

In the function `_removeSigner()` line 1140 a manual array resizing is made. One should be careful because such an operation is not possible with Solidity version 0.6.0 forwards where low-level assembly must be utilized.

Recommendation:

We recommend paying attention if the codebase is planned to be updated to newer Solidity version.

Alleviation:

The Matic Staking Contract development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

SMR-08: Incorrect data allocation

Type	Severity	Location
Language Specific	Informational	StakeManager.sol L234

Description:

The function `insertSigners()` does not appear anywhere in the codebase, so it is meant to be called externally, which means the input data allocation should be `calldata` instead of `memory`.

Recommendation:

We recommend changing the data allocation

Alleviation:

The team has followed the recommendation and changed data allocation.

VSE-01: Unnecessary computation

Type	Severity	Location
Logical Issue	Informational	ValidatorShare.sol L363

Description:

On the right hand side of line 363 we want to calculate the exact paid amount for `shares` so we can use `rate.mul(shares).div(precision` directly.

Recommendation:

Change the formula accordingly.

Alleviation:

The team has followed the recommendation and fixed the formula.

VSE-02: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	ValidatorShare.sol L118, L260, L282, L305, L340

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The team has followed the recommendation.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Arithmetic

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.