

**Stader Labs**

**MaticX**

**Smart Contract Audit  
Final Report**



**April 25, 2022**

<b>Introduction</b>	<b>3</b>
About Stader Labs	3
About ImmuneBytes	3
<b>Documentation Details</b>	<b>3</b>
<b>Audit Process &amp; Methodology</b>	<b>4</b>
<b>Audit Details</b>	<b>4</b>
<b>Audit Goals</b>	<b>5</b>
<b>Security Level Reference</b>	<b>5</b>
High Severity Issues	6
Medium Severity Issues	6
Low Severity Issues	7
<b>Recommendation / Informational</b>	<b>9</b>
<b>Unit Tests</b>	<b>10</b>
<b>Test Coverage</b>	<b>12</b>
<b>Automated Audit Result</b>	<b>13</b>
Maian	13
Mythril	13
MythX	14
Slither	15
<b>Concluding Remarks</b>	<b>16</b>
<b>Disclaimer</b>	<b>16</b>

## Introduction

### 1. About Stader Labs

Stader is building the key staking middleware infrastructure layer for multiple PoS networks that will power the above staking-related opportunities while solving the key challenges. We are taking an extremely modular approach to building our contracts so third parties can leverage our components to build several staking solutions on top of it.

In the short term, Stader is building native staking smart contracts across multiple chains including Terra, and Solana, among others, and building an economic ecosystem to grow and develop solutions like YFI-style farming with rewards, launchpads, gaming with rewards, liquid staking solutions, and more.

In the long term, Stader is focused on unlocking the platform approach and nurturing third parties to develop several staking-related applications on top of Stader infrastructure.

Visit <https://staderlabs.com/> to know more about it.

### 2. About ImmuneBytes

ImmuneBytes is a security start-up to provides professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-ups with a detailed analysis of the system ensuring security and managing the overall project.

Visit <http://immunebytes.com/> to know more about the services.

## Documentation Details

The Stader Labs team has provided the following doc for the purpose of audit:

1. [https://staderlabs-docs.s3.amazonaws.com/Stader\\_Litepaper.pdf](https://staderlabs-docs.s3.amazonaws.com/Stader_Litepaper.pdf)
2. <https://github.com/stader-labs/maticX#readme>

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

## Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

## Audit Details

- Project Name: Stader Labs
- Contracts Name: [MaticX.sol](#), [ValidatorRegistry.sol](#)
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Github commits for initial audit: 85023fa73a1325d0794dd3097a7efcbad72379bd
- Github commits for final audit: [8f914608ae40fdb35cfae281ff6c1dda9943b632](#)
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

## Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include
  - a. Correctness
  - b. Readability
  - c. Sections of code with high complexity
  - d. Quantity and quality of test coverage

## Security Level Reference

Every issue in this report were assigned a severity level from the following:

**High severity issues** will bring problems and should be fixed.

**Medium severity issues** could potentially bring problems and should eventually be fixed.

**Low severity issues** are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	-	-	-
Closed	1	1	3

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

## High Severity Issues

### 1. Missing Authentication

Contract: Missing implementation of contracts (state Manager and Validator share)

**Description:**

As there are external calls to the missing contracts so the contract methods which are dependent on external calls will fail.

**Recommendation:**

Add missing implementation

**Status:** Acknowledged

**Note by the team:**

*Both stakeManager and validatorShare contracts are implemented and deployed by Polygon. We only use the interface contract to interact with it.*

## Medium Severity Issues

### 1. Contract:

MaticX.sol

**Description:**

The reentrancy guard is missing in the MaticX.requestWithdraw.

The state variable is written after the external call of validatorRegistry.setLastWithdrawnValidatorId.

**Recommendation:**

Create or Import a nonReentrancy guard from OpenZeppelin and apply it to the method.

**Status:** Closed

**Note by the team:**

*It is calling \_burn function after the validatorRegistry.setLastWithdrawnValidatorId function call and validatorRegistry contract is our own contract. We can add reentrancy guard which will increase our gas cost but we don't see an issue unless our manager role gets hacked and the hacker changes the validatorRegistry contract address which itself is a separate concern. Please give us more details on reproducing this issue.*

## Low Severity Issues

### 1. **Contract:**

MaticX.sol

#### **Description:**

The rescue function for the by mistake sent funds is missing in the contract.

If a user sends the funds by mistake to the contract then there is no way to pull the funds out by the admin and give it back to the user.

#### **Recommendation:**

We recommend a rescue funds method which is only operated by Admin.

**Status:** Acknowledged

#### **Note by the team:**

*As of now we believe it is a user fault that is similar to sending tokens to a wrong address and is a nice to have feature which can be included later when it becomes a major issue.*

### 2. **Gas optimizations**

Contract: MaticX.sol, ValidatorRegistry.sol

#### **Description:**

In for loop on line 186 (MaticX) and 111 (Validator Registry) the .length is calculated again and again in the for loop which will use a SLOAD operation on each iteration

#### **Recommendation:**

We recommend calculating the length and store it in a variable and use that variable in the for loop iteration.

**Status:** Closed

Amended: Issue was fixed by the **Stader Labs** team and is no longer present in commit [8f914608ae40fdb35cfae281ff6c1dda9943b632](https://github.com/stader-labs/maticx/commit/8f914608ae40fdb35cfae281ff6c1dda9943b632)

### 3. Address zero check missing

Contract: MaticX.sol, ValidatorRegistry.sol

**Description:**

Zero check is missing in the input parameters.

functions : MaticX.initialize, MaticX.setTreasuryAddress, ValidatorRegistry.initialize, MaticX.setValidatorRegistry, ValidatorRegistry.setMaticX,

**Recommendation:**

We recommend including zero check.

**Status:** Acknowledged

**Note by the team:**

*Since we can recall this functions if we were to mistakenly set it to address(0), we believe it is not worth adding an additional check for it.*

## Recommendation / Informational

### 1. Log emit missing

Contract: MaticX.sol

#### **Description:**

Whenever the state of the contract gets changed then their event should be admitted.  
Some functions are missing the emitting of logs.

For eg: setValidatorRegistry, setVersion, setTreasury, setFees, setMaticX, setVersion

#### **Recommendation:**

At the time of state change events should be emitted properly.

**Status:** Closed

Amended: Issue was fixed by the **Stader Labs** team and is no longer present in commit [8f914608ae40fdb35cfae281ff6c1dda9943b632](https://github.com/stader-labs/MaticX/commit/8f914608ae40fdb35cfae281ff6c1dda9943b632)

## Unit Tests

MaticX contract

- ✓ Should submit successfully
- ✓ fails when non instant pool owner interacts with instant pool provision & withdrawal
- ✓ Should provide and withdraw matic to instant pool successfully
- ✓ Should provide and withdraw MaticX to instant pool successfully
- ✓ allows users to interact with instant pool swap
- ✓ fails when submit amount is greater than signer balance
- ✓ Should request withdraw from the contract successfully
- ✓ WithdrawalRequest should have correct share amount
- ✓ Should claim withdrawals after submitting to contract successfully
- ✓ Should stake rewards to a validator successfully without using instant pool matic
- ✓ Should migrate validator stake to another validator successfully
- ✓ Should send correct message from L1 to L2

ValidatorRegistry contract

- ✓ Should add new validators
- ✓ Should not add existing validator
- ✓ Should remove validators
- ✓ Should not remove an validator when it is preferred for deposits
- ✓ Should not remove an validator when it is preferred for withdrawals
- ✓ Should not remove non existing validator

Solc version: 0.8.7		Optimizer enabled: true		Runs: 200	Block limit: 30000000 gas	
Methods						
Contract	Method	Min	Max	Avg	# calls	usd (avg)
ERC20Upgradeable	approve	53605	53641	53634	11	-
ERC20Upgradeable	transfer	-	-	54092	1	-
FxStateChildTunnel	setFxRootTunnel	-	-	46326	12	-
FxStateRootTunnel	setFxChildTunnel	46263	46275	46273	12	-
FxStateRootTunnel	setMaticX	-	-	29176	12	-
MaticX	claimWithdrawal	-	-	119941	12	-
MaticX	migrateDelegation	-	-	82548	2	-
MaticX	provideInstantPoolMatic	72699	99399	87299	3	-

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

MaticX	provideInstantPoolMatic	72699	99399	87299	3	-
MaticX	provideInstantPoolMaticX	68922	86022	77472	2	-
MaticX	requestWithdraw	287682	343758	310291	17	-
MaticX	setFeePercent	-	-	37373	2	-
MaticX	setFxStateRootTunnel	-	-	54758	12	-
MaticX	stakeRewardsAndDistributeFees	220519	225319	224119	4	-
MaticX	submit	223689	419172	302474	40	-
MaticX	swapMaticForMaticXViaInstantPool	-	-	136396	1	-
MaticX	withdrawInstantPoolMatic	-	-	72095	1	-
MaticX	withdrawInstantPoolMaticX	-	-	68350	1	-
PolygonMock	approve	26356	46256	45538	28	-
PolygonMock	mint	50785	67897	58999	25	-
PolygonMock	mintTo	34135	51235	42685	2	-
PolygonMock	transfer	-	-	51580	1	-
StakeManagerMock	createValidator	-	-	1340427	32	-
StakeManagerMock	setEpoch	26474	43562	35018	2	-
ValidatorRegistry	addValidator	116967	134067	126210	37	-
ValidatorRegistry	removeValidator	55838	65849	60844	4	-
ValidatorRegistry	setMaticX	-	-	54779	18	-
ValidatorRegistry	setPreferredDepositValidatorId	-	-	58772	14	-
ValidatorRegistry	setPreferredWithdrawalValidatorId	-	-	58792	14	-
Deployments					% of limit	
FxRootMock		-	-	188551	0.6 %	-
FxStateChildTunnel		-	-	851187	2.8 %	-
FxStateRootTunnel		-	-	2185299	7.3 %	-
MaticX		-	-	4068668	13.6 %	-
PolygonMock		-	-	666389	2.2 %	-
RateProvider		590923	590935	590933	2 %	-
StakeManagerMock		1369621	1369645	1369643	4.6 %	-
ValidatorRegistry		-	-	1488007	5 %	-
18 passing (10s)						

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

## Test Coverage

### MaticX contract

- ✓ Should submit successfully (185ms)
- ✓ fails when non instant pool owner interacts with instant pool provision & withdrawal (257ms)
- ✓ Should provide and withdraw matic to instant pool successfully (99ms)
- ✓ Should provide and withdraw MaticX to instant pool successfully (212ms)
- ✓ allows users to interact with instant pool swap (170ms)
- ✓ fails when submit amount is greater than signer balance (57ms)
- ✓ Should request withdraw from the contract successfully (172ms)
- ✓ WithdrawalRequest should have correct share amount (184ms)
- ✓ Should claim withdrawals after submitting to contract successfully (1026ms)
- ✓ Should stake rewards to a validator successfully without using instant pool matic (665ms)
- ✓ Should migrate validator stake to another validator successfully (153ms)
- ✓ Should send correct message from L1 to L2 (369ms)

### ValidatorRegistry contract

- ✓ Should add new validators (112ms)
- ✓ Should not add existing validator (48ms)
- ✓ Should remove validators (123ms)
- ✓ Should not remove an validator when it is preferred for deposits (56ms)
- ✓ Should not remove an validator when it is preferred for withdrawals (60ms)
- ✓ Should not remove non existing validator

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	82.38	53.7	68.18	82.74	
MaticX.sol	82.47	44.74	67.74	82.47	... 724,725,726
ValidatorRegistry.sol	82.05	75	69.23	83.72	... 193,194,205
contracts/interfaces/	100	100	100	100	
IFxStateChildTunnel.sol	100	100	100	100	
IFxStateRootTunnel.sol	100	100	100	100	
IMaticX.sol	100	100	100	100	
IRateProvider.sol	100	100	100	100	
IStakeManager.sol	100	100	100	100	
IValidatorRegistry.sol	100	100	100	100	
IValidatorShare.sol	100	100	100	100	
contracts/lib/	0	0	0	0	
ExitPayloadReader.sol	0	0	0	0	... 207,211,220
Merkle.sol	0	0	0	0	... 25,29,34,36
MerklePatriciaProof.sol	0	0	0	0	... 144,147,155
RLPReader.sol	0	0	0	0	... 361,364,366
contracts/mocks/	67.19	25	61.29	67.69	
FxRootMock.sol	100	100	100	100	
PolygonMock.sol	100	100	100	100	
StakeManagerMock.sol	62.5	100	60	62.5	... ,97,120,121
ValidatorShareMock.sol	67.57	25	50	67.57	... 124,125,127
contracts/state-transfer/	90	50	84.62	90	
FxStateChildTunnel.sol	100	100	100	100	
FxStateRootTunnel.sol	85.71	50	80	85.71	26
RateProvider.sol	75	100	66.67	75	30
contracts/tunnel/	24.14	14.29	55.56	26.67	
FxBaseChildTunnel.sol	80	50	75	83.33	60
FxBaseRootTunnel.sol	12.5	0	40	12.5	... 164,184,185
All files	43.49	21.43	45.45	43.88	

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

## Automated Audit Result

### Maian

```

=====
[ ] Check if contract is SUICIDAL

[ ] Contract address : 0xaFFeCAFEaFFeCaFEaFFeCaFEaFFeCaFEaFFeCaFE
[ ] Contract bytecode : 608060405234801561001057600080fd5b50612b3a80610020...
[ ] Bytecode length : 22196
[ ] Blockchain contract: False
[ ] Debug : False

[-] The code does not contain SUICIDE instructions, hence it is not vulnerable
root@a05bb3151e0a:/MAIAN/tool# python3 maian.py -b /share/contracts/validator.bytecode -c 0

=====
[ ] Check if contract is SUICIDAL

[ ] Contract address : 0xaFFeCAFEaFFeCaFEaFFeCaFEaFFeCaFEaFFeCaFE
[ ] Contract bytecode : 608060405234801561001057600080fd5b50612b3a80610020...
[ ] Bytecode length : 22196
[ ] Blockchain contract: False
[ ] Debug : False

[-] The code does not contain SUICIDE instructions, hence it is not vulnerable
root@a05bb3151e0a:/MAIAN/tool# python3 maian.py -b /share/contracts/validator.bytecode -c 1

=====
[ ] Check if contract is PRODIGAL

[ ] Contract address : 0xaFFeCAFEaFFeCaFEaFFeCaFEaFFeCaFEaFFeCaFE
[ ] Contract bytecode : 608060405234801561001057600080fd5b50612b3a80610020...
[ ] Bytecode length : 22196
[ ] Blockchain contract: False
[ ] Debug : False
[+] The code does not have CALL/SUICIDE, hence it is not prodigal
root@a05bb3151e0a:/MAIAN/tool# python3 maian.py -b /share/contracts/validator.bytecode -c 2

=====
[ ] Check if contract is GREEDY

[ ] Contract address : 0xaFFeCAFEaFFeCaFEaFFeCaFEaFFeCaFEaFFeCaFE
[ ] Contract bytecode : 608060405234801561001057600080fd5b50612b3a80610020...
[ ] Bytecode length : 22196
[ ] Debug : False
[-] Contract can receive Ether

[-] No lock vulnerability found because the contract cannot receive Ether

```

### Mythril

```
The analysis was completed successfully. No issues were detected.
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

## MythX

Line	SWC Title	Severity	Short Description
8	(SWC-103) Floating Pragma	Low	A floating pragma is set.
314	(SWC-103) Floating Pragma	Low	A floating pragma is set.
344	(SWC-103) Floating Pragma	Low	A floating pragma is set.
477	(SWC-123) Requirement Violation	Low	Requirement violation.
543	(SWC-103) Floating Pragma	Low	A floating pragma is set.
625	(SWC-103) Floating Pragma	Low	A floating pragma is set.
665	(SWC-103) Floating Pragma	Low	A floating pragma is set.
759	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
1062	(SWC-103) Floating Pragma	Low	A floating pragma is set.
1154	(SWC-103) Floating Pragma	Low	A floating pragma is set.
1225	(SWC-103) Floating Pragma	Low	A floating pragma is set.
1254	(SWC-103) Floating Pragma	Low	A floating pragma is set.
1298	(SWC-103) Floating Pragma	Low	A floating pragma is set.
1460	(SWC-123) Requirement Violation	Low	Requirement violation.
1536	(SWC-103) Floating Pragma	Low	A floating pragma is set.
1637	(SWC-103) Floating Pragma	Low	A floating pragma is set.
2130	(SWC-123) Requirement Violation	Low	Requirement violation.
2150	(SWC-113) DoS with Failed Call	Medium	Multiple calls are executed in the same transaction.
2150	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
2320	(SWC-103) Floating Pragma	Low	A floating pragma is set.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

## Slither

```
Compiled with solc
Number of lines: 2619 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 20 (+ 0 in dependencies, + 0 tests)
```

```
Number of optimization issues: 12
Number of informational issues: 103
Number of low issues: 26
Number of medium issues: 1
Number of high issues: 3
ERCs: ERC165, ERC20
```

Name	# functions	ERCs	ERC20 info	Complex code	Features
InvalidatorShare	23			No	
AddressUpgradeable	9			No	Send ETH Assembly
StringsUpgradeable	4			Yes	
SafeERC20Upgradeable	6			No	Send ETH Tokens interaction
IStakeManager	13			No	
MaticX	101	ERC20,ERC165	No Minting Approve Race Cond.	No	Send ETH Tokens interaction Upgradeable
ValidatorRegistry	60	ERC165		No	Upgradeable

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

## Concluding Remarks

While conducting the audits of the Stader Labs smart contracts, it was observed that the contracts contain High, Medium, and Low severity issues.

Our auditors suggest that High, Medium, and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

**Note: *Stader Labs team has Acknowledged/fixed the issues based on the auditor's recommendation. Stader Labs contracts do not have any issues present in the contract.***

## Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Stader Labs platform or its product nor this audit is investment advice.

Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

***ImmuneBytes***