Ether Authority

SMART CONTRACT

Security Audit Report

Project: **Prime (PRIME)**
Website: **echelon.io**
Platform: **Base Chain Network**
Language: **Solidity**
Date: **June 7th, 2024**

Table of contents

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Prime (PRIME) smart contract from echelon.io was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on June 7th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

Website Details

Echelon is a platform focused on advancing web3 gaming by integrating blockchain features. It provides a frictionless and compelling utility for the PRIME token, enhancing game experiences. The platform aims to enable games to leverage blockchain technology, with the "Parallel TCG" game as an example of its capabilities. Echelon invites developers to build and enhance their games using its infrastructure.
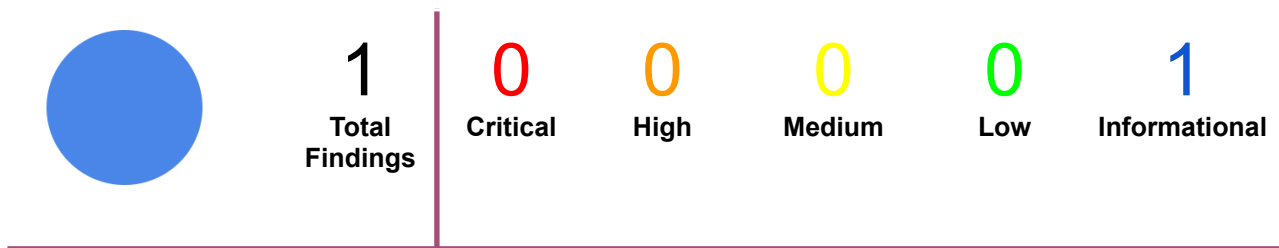
Code Details

- The `PrimeToken` smart contract continues to integrate various functionalities from OpenZeppelin, LayerZero's OFT, and custom logic specific to the Echelon ecosystem. Below is an explanation of the contract and its key components, along with some clarifications and improvements where necessary.
- Imports and Inheritance:
  - ReentrancyGuard: Protects against reentrant calls to functions.
  - OFT: Extends ERC20 functionality to support omnichain token transfers via LayerZero.
  - Context: Provides information about the current execution context (e.g., msg.sender and msg.data).
  - ERC2771Recipient: Allows the contract to accept meta-transactions.
  - EchelonGateways and IPrimeToken: Custom interfaces and contracts specific to the Echelon ecosystem.
- setTrustedForwarder Function: Allows the contract owner to update the trusted forwarder address.
- invokeEchelon Function: Enables users to send PRIME and native tokens to the Echelon ecosystem. It validates inputs, performs token transfers, and calls the handler's custom logic.
- addEchelonHandlerContract Function: Allows the owner to register new handler contracts, ensuring no existing gateways are overwritten.
- send Function: Uses LayerZero's OFT functionality to bridge tokens to another chain.
- Meta-Transaction Support: Overrides `_msgSender` and `_msgData` to correctly handle meta-transactions.
- The `PrimeToken` contract effectively integrates cross-chain token functionality with meta-transaction support and custom logic handling for the Echelon ecosystem. It ensures secure and efficient handling of token transfers and custom logic invocations, making it a versatile solution for decentralized applications.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Audit scope

Name	**Code Review and Security Analysis Report for Prime (PRIME) Smart Contract**
Platform	**Base Chain Network**
Language	**Solidity**
File	PrimeToken.sol
Smart Contract Code	0xfA980cEd6895AC314E7dE34Ef1bFAE90a5AdD21b
Audit Date	June 7th,2024
Audit Result	**Passed**

Code Audit History

1	0	0	0	0	1
Total Findings	**Critical**	**High**	**Medium**	**Low**	**Informational**

Severity Definitions

0 🟥	**Critical**		Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
0 🟧	**High**		High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. Public access is crucial.
0 🟨	**Medium**		Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
0 🟩	**Low**		Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
1 🟦	**Lowest / Informational / Best Practice**		Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: Prime ● Symbol: PRIME ● Decimals: 18	**YES, This is valid.**
Owner Specifications: ● Set config for LayerZero user Application. ● Update the send/receive version. ● Set the trusted path for cross-chain communication. ● Update the trusted remote address. ● Renouncing ownership. ● Allows the current owner to transfer control of the contract to a new owner.	**YES, This is valid. We advise renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Secured"**. Also, these contracts contain owner control, which does not make them fully decentralized.

Unsecured	Poor Secured	Secured	Well Secured

You are here

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 1 very low-level issue.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	"Out of Gas" Issue	Moderated
	High consumption 'for/while' loop	Passed
	High consumption 'storage' storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Overall Audit Result: PASSED

Business Risk Analysis

Category	Result
🟢 Buy Tax	0%
🟢 Sell Tax	0%
🟢 Cannot Buy	No
🟢 Cannot Sell	No
🟢 Max Tax	0%
🟢 Modify Tax	No
🟢 Fee Check	Not Detected
🟢 Is Honeypot	Not Detected
🟢 Trading Cooldown	Not Detected
🟢 Can Pause Trade?	Not Detected
🟢 Pause Transfer?	No
🟢 Max Tax?	No
🟢 Is it Anti-whale?	Not Detected
🟢 Is Anti-bot?	Not Detected
🟢 Is it a Blacklist?	No
🟢 Blacklist Check	No
🟢 Can Mint?	No
🟢 Is it a Proxy Contract?	No
🟢 Is it used Open Source?	No
🟢 External Call Risk?	No
🟢 Balance Modifiable?	No
🟢 Can Take Ownership?	Yes
🟢 Ownership Renounce?	Yes
🟢 Hidden Owner?	Not Detected
🟢 Self Destruction?	Not Detected
🟢 Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Prime Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Prime Token.

The EtherAuthority team has not provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Prime Token smart contract code in the form of a [basescan](#) web link.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

PrimeToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	setTrustedForwarder	external	access only owner	No Issue
3	invokeEchelon	write	Gas Optimization	Refer Audit Findings
4	addEchelonHandlerContract	write	Gas Optimization	Refer Audit Findings
5	send	write	Gas Optimization	Refer Audit Findings
6	_msgSender	read	Passed	No Issue
7	_msgData	read	Passed	No Issue
8	supportsInterface	read	Passed	No Issue
9	token	read	Passed	No Issue
10	circulatingSupply	read	Passed	No Issue
11	_debitFrom	internal	Passed	No Issue
12	_creditTo	internal	Passed	No Issue
13	getTrustedForwarder	read	Passed	No Issue
14	_setTrustedForwarder	internal	Passed	No Issue
15	isTrustedForwarder	read	Passed	No Issue
16	_msgSender	internal	Passed	No Issue
17	_msgData	read	Passed	No Issue
18	nonReentrant	modifier	Passed	No Issue
19	_nonReentrantBefore	write	Passed	No Issue
20	_nonReentrantAfter	write	Passed	No Issue
21	_blockingLzReceive	internal	Passed	No Issue
22	_storeFailedMessage	internal	Passed	No Issue
23	nonblockingLzReceive	write	Passed	No Issue
24	_nonblockingLzReceive	internal	Passed	No Issue
25	retryMessage	write	Passed	No Issue

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

No Low-severity vulnerabilities were found.

Very Low / Informational / Best practices:

[I-01] Gas Optimization:

Description:

public' functions that are never called by the contract could be declared external functions in the contract like:

primeToken.sol

- invokeEchelon
- addEchelonHandlerContract
- send

Recommendation: We suggest declaring these all functions external for better Gas optimization.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble.

Centralized	Decentralized

You are here

The following are Admin functions:

PrimeToken.sol

- setTrustedForwarder: Updated trusted forwarder address only by the owner.
- addEchelonHandlerContract: Add the Echelon handler contract address only by the owner.

OFTCore.sol

- setUseCustomAdapterParams: The useCustomAdapterParams values can be updated by the owner.

LzApp.sol

- setConfig: Generic config for LayerZero user Application by the owner.
- setSendVersion: The owner can update the send version.
- setReceiveVersion: The owner can update the receive version.
- forceResumeReceive: The owner can update the receive address.
- setTrustedRemote: Set the trusted path for the cross-chain communication by the owner.
- setTrustedRemoteAddress: Update the trusted remote address by the owner.
- setPrecrime: The Precrime address can be set by the owner.
- setMinDstGas: The MinDstGas value can be set by the owner.
- setPayloadSizeLimit: The payload size limit value can be set by the owner.

Ownable.sol

- renounceOwnership: Renouncing ownership will leave the contract without an owner.
- transferOwnership: Allows the current owner to transfer control of the contract to a new owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a [basescan](#) web link. And we have used all possible tests based on given objects as files. We observed 1 Informational issue in the smart contracts. but those are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io
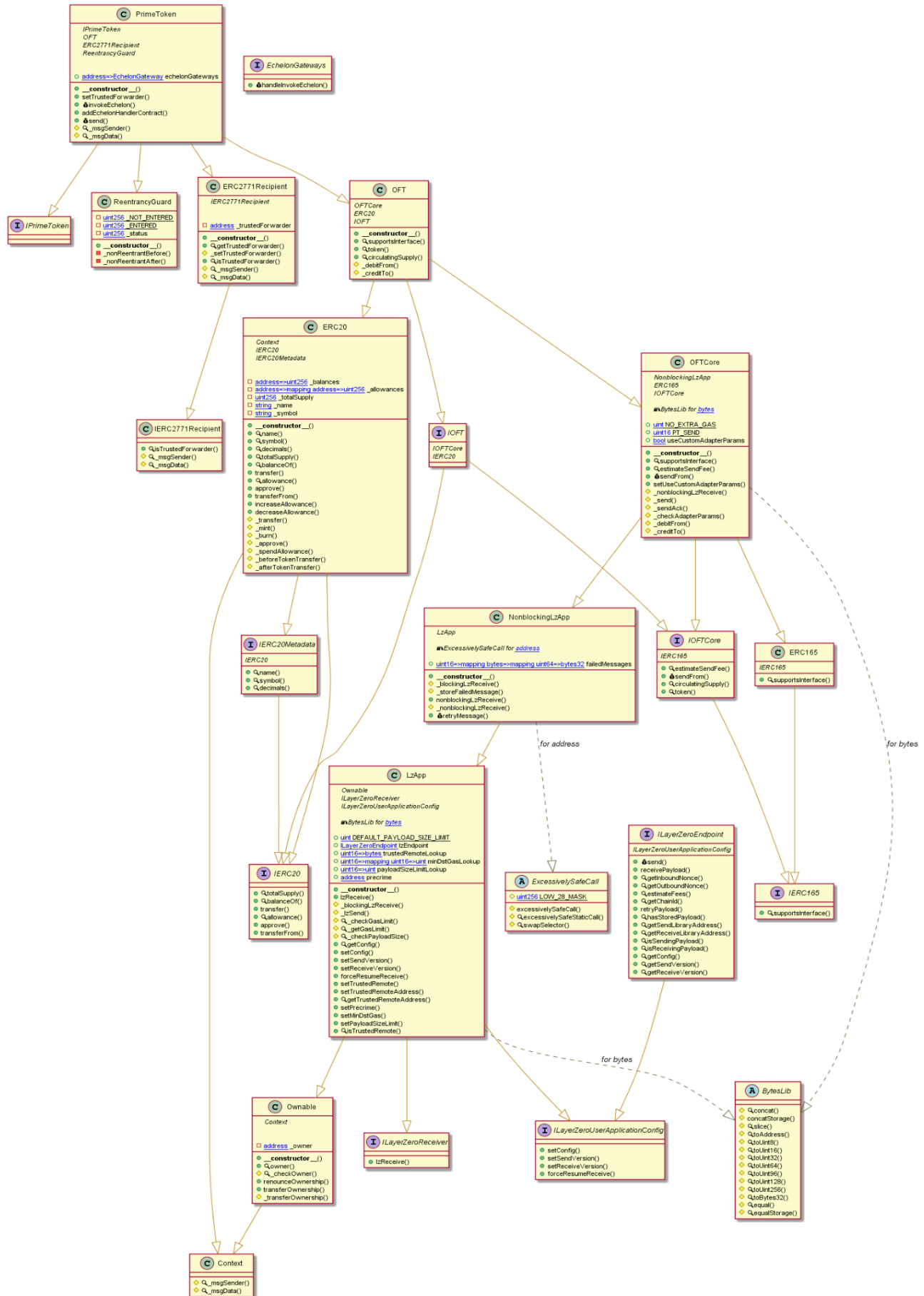
Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Prime Token

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysi s of the project altogether. Below are the results.

PrimeToken.sol

```
INFO:Detectors:
Reentrancy in PrimeToken.invokeEchelon(address,uint256,uint256,bytes)
(PrimeToken.sol#1887-1937):
     External calls:
     - (sent) = gateway.nativeTokenDestinationAddress.call{value: msg.value}()
(PrimeToken.sol#1906-1908)
     State variables written after the call(s):
     - transfer(gateway.primeDestinationAddress,_primeValue) (PrimeToken.sol#1914)
          - _balances[from] = fromBalance - amount (PrimeToken.sol#1401)
          - _balances[to] += amount (PrimeToken.sol#1404)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
     Event emitted after the call(s):
     - SendToChain(_dstChainId,_from,_toAddress,amount) (PrimeToken.sol#1783)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
PrimeToken._msgData() (PrimeToken.sol#2022-2030) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (PrimeToken.sol#4) allows old versions
solc-0.8.25 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in PrimeToken.invokeEchelon(address,uint256,uint256,bytes)
(PrimeToken.sol#1887-1937):
     - (sent) = gateway.nativeTokenDestinationAddress.call{value: msg.value}()
(PrimeToken.sol#1906-1908)
```

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter PrimeToken.setTrustedForwarder(address)._forwarder (PrimeToken.sol#1862) is not in mixedCase
Parameter PrimeToken.invokeEchelon(address,uint256,uint256,bytes)._handlerAddress (PrimeToken.sol#1888) is not in mixedCase
Parameter PrimeToken.invokeEchelon(address,uint256,uint256,bytes)._id (PrimeToken.sol#1889) is not in mixedCase
Parameter PrimeToken.invokeEchelon(address,uint256,uint256,bytes)._primeValue (PrimeToken.sol#1890) is not in mixedCase
Parameter PrimeToken.invokeEchelon(address,uint256,uint256,bytes)._data (PrimeToken.sol#1891) is not in mixedCase
Parameter PrimeToken.addEchelonHandlerContract(address,address,address)._contractAddress (PrimeToken.sol#1948) is not in mixedCase
Parameter PrimeToken.addEchelonHandlerContract(address,address,address)._nativeTokenDestinationAddress (PrimeToken.sol#1949) is not in mixedCase
Parameter PrimeToken.addEchelonHandlerContract(address,address,address)._primeDestinationAddress (PrimeToken.sol#1950) is not in mixedCase
Parameter PrimeToken.send(address,uint16,uint256,address,address,bytes)._from (PrimeToken.sol#1988) is not in mixedCase
Parameter PrimeToken.send(address,uint16,uint256,address,address,bytes)._dstChainId (PrimeToken.sol#1989) is not in mixedCase
Parameter PrimeToken.send(address,uint16,uint256,address,address,bytes)._amount (PrimeToken.sol#1990) is not in mixedCase
Parameter PrimeToken.send(address,uint16,uint256,address,address,bytes)._refundAddress (PrimeToken.sol#1991) is not in mixedCase
Parameter PrimeToken.send(address,uint16,uint256,address,address,bytes)._zroPaymentAddress (PrimeToken.sol#1992) is not in mixedCase
Parameter PrimeToken.send(address,uint16,uint256,address,address,bytes)._adapterParams (PrimeToken.sol#1993) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
BytesLib.toAddress(bytes,uint256) (PrimeToken.sol#464-473) uses literals with too many digits:
    - tempAddress = mload(uint256)(_bytes + 0x20 + _start) / 0x1000000000000000000000000 (PrimeToken.sol#469)
ExcessivelySafeCall.slitherConstructorConstantVariables() (PrimeToken.sol#679-811) uses literals with too many digits:
    - LOW_28_MASK = 0x00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff (PrimeToken.sol#680-681)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Slither:PrimeToken.sol analyzed (24 contracts with 93 detectors), 155 result(s) found

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

PrimeToken.sol

Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in PrimeToken.invokeEchelon(address,uint256,uint256,bytes): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
Pos: 1887:4:

Inline assembly:
The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
Pos: 1764:8:

Low level calls:
Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
Pos: 1906:28:

Gas costs:
Gas requirement of function OFT.setMinDstGas is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1663:4:

Gas costs:
Gas requirement of function PrimeToken.send is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1987:4:

This on local calls:
Use of "this" for local functions: Never use "this" to call functions in the same contract, it only consumes more gas than normal local calls.
Pos: 1694:119:

Constant/View/Pure functions:

OFTCore._creditTo(uint16,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
Pos: 1805:4:

No return:
OFTCore._creditTo(uint16,address,uint256): Defines a return type but never explicitly returns a value.
Pos: 1805:4:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 1719:8:

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

PrimeToken.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:3
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:189
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:258
Variable "mlengthmod" is unused
Pos: 21:369
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:408
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:467
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:478
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:489
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:500
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:511
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:522
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:533
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:544
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:555
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:565
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:615
Explicitly mark visibility of state
Pos: 5:679
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:711
```

Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:763
Provide an error message for require
Pos: 9:798
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:800
Code contains empty blocks
Pos: 1:994
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1016
Error message for require is too long
Pos: 9:1058
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1099
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1137
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 13:1172
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1215
Error message for require is too long
Pos: 9:1365
Error message for require is too long
Pos: 9:1392
Error message for require is too long
Pos: 9:1393
Error message for require is too long
Pos: 9:1398
Error message for require is too long
Pos: 9:1447
Error message for require is too long
Pos: 9:1452
Error message for require is too long
Pos: 9:1482
Error message for require is too long
Pos: 9:1483
Code contains empty blocks
Pos: 24:1529
Code contains empty blocks
Pos: 24:1549
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1570
Error message for require is too long
Pos: 9:1580
Error message for require is too long
Pos: 9:1590
Check result of "send" call
Pos: 9:1592
Avoid to use inline assembly. It is acceptable only in rare cases

Pos: 9:1604
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1684
Code contains empty blocks
Pos: 53:1684
Error message for require is too long
Pos: 9:1707
Error message for require is too long
Pos: 9:1717
Error message for require is too long
Pos: 9:1718
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1740
Code contains empty blocks
Pos: 68:1740
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1763
Error message for require is too long
Pos: 13:1798
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1809
Code contains empty blocks
Pos: 125:1809
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1852
Code contains empty blocks
Pos: 71:1855
Error message for require is too long
Pos: 9:1892
Error message for require is too long
Pos: 9:1951

Software analysis result:

This software reported many false positive results and some were informational issues. So, those issues can be safely ignored.