

May 16, 2023

SMART CONTRACT

AUDIT REPORT

EtherFi ETH2.0 Staking



omniscia.io



info@omniscia.io



Online report: etherfi-eth-2.0-staking

ETH 2.0 Staking Security Audit

Audit Overview

We were tasked with performing an audit of the EtherFi codebase and in particular their novel ETH2.0 staking mechanism that matches node operators and potential fund providers using a bidding system along with a customized reward distribution model for each node that makes use of NFTs.

Over the course of the audit, we identified multiple errors of significant severity the most crucial of which arise from the Beacon chain deposit mechanism and its susceptibility to a front-run attack with different withdrawal credentials.

We advise the EtherFi team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The EtherFi team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by EtherFi and have identified that certain exhibits have not been adequately dealt with. We advise the EtherFi team to revisit the following exhibits: SME-04M, EFM-05M, EFN-05M

Additionally, we advise these informational / static analysis exhibits to be re-visited as they have been remediated either partially or improperly: EFM-01S, TYR-01C, SMR-01C, SMR-04C, SMR-02C, EFN-01C, EFN-11C, EFN-04C, EFN-10C, EFN-03C, EFN-09C, PRM-02C, PRM-04C, CRP-02C, AMR-01C, AMR-03C, AMR-02C, EFM-02C, EFM-05C, SME-02C

Contracts Assessed

Files in Scope	Repository	Commit(s)
AuctionManager.sol (AMR)	dappContracts	0f9df283aa, 3a52fa3a5d
BNFT.sol (BNF)	dappContracts	0f9df283aa, 3a52fa3a5d
ClaimReceiverPool.sol (CRP)	dappContracts	0f9df283aa, 3a52fa3a5d
EtherFiNode.sol (EFN)	dappContracts	0f9df283aa, 3a52fa3a5d
EarlyAdopterPool.sol (EAP)	dappContracts	0f9df283aa, 3a52fa3a5d
EtherFiNodesManager.sol (EFM)	dappContracts	0f9df283aa, 3a52fa3a5d
NodeOperatorManager.sol (NOM)	dappContracts	0f9df283aa, 3a52fa3a5d
ProtocolRevenueManager.sol (PRM)	dappContracts	0f9df283aa, 3a52fa3a5d
ScoreManager.sol (SMR)	dappContracts	0f9df283aa, 3a52fa3a5d
StakingManager.sol (SME)	dappContracts	0f9df283aa, 3a52fa3a5d
TNFT.sol (TNF)	dappContracts	0f9df283aa, 3a52fa3a5d
Treasury.sol (TYR)	dappContracts	0f9df283aa, 3a52fa3a5d
UUPSProxy.sol (UUP)	dappContracts	0f9df283aa, 3a52fa3a5d

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	6	6	0	0
Informational	73	53	7	13
Minor	22	22	0	0
Medium	2	2	0	0
Major	11	8	0	3

During the audit, we filtered and validated a total of **27 findings utilizing static analysis** tools as well as identified a total of **87 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

Compilation

The project utilizes hardhat as its development pipeline tool, containing an array of tests and scripts coded in TypeScript.

To compile the project, the compile command needs to be issued via the npx CLI tool to hardhat:

```
npx hardhat compile
```

The hardhat tool automatically selects Solidity version 0.8.13 based on the version specified within the hardhat.config.ts file.

The project contains discrepancies with regards to the Solidity version used as the pragma statements of the contracts are open-ended (^0.8.13).

We advise them to be locked to 0.8.13 (=0.8.13), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the hardhat pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **386 potential issues** within the codebase of which **325** were ruled out to be false positives or negligible findings.

The remaining 61 issues were validated and grouped and formalized into the 27 exhibits that follow:

ID	Severity	Addressed	Title
AMR-01S	Informational	Yes	Inexistent Event Emissions
AMR-02S	Informational	Yes	Inexistent Visibility Specifier
AMR-03S	Informational	Yes	Literal Equality of bool Variables
AMR-04S	Minor	Yes	Inexistent Sanitization of Input Addresses
BNF-01S	Informational	Yes	Inexistent Visibility Specifier
BNF-02S	Minor	Yes	Inexistent Sanitization of Input Address
CRP-01S	Informational	Yes	Illegible Numeric Value Representation
CRP-02S	Informational	Yes	Inexistent Visibility Specifiers
CRP-03S	Minor	Yes	Inexistent Sanitization of Input Addresses
EAP-01S	Informational	Nullified	Illegible Numeric Value Representations
EAP-02S	Minor	Nullified	Inexistent Sanitization of Input Addresses
EAP-03S	Minor	Nullified	Potential Lock of Native Assets
EAP-04S	Medium	Nullified	Improper Invocations of EIP-20 transfer / transferFrom

	•		
EFN-01S	Informational	✓ Yes	Illegible Numeric Value Representation
EFN-013	momatorial	Tes	megible Numeric value representation
EFN-02S	Minor	✓ Yes	Inexistent Sanitization of Input Address
EFM-01S	Informational	• Partial	Illegible Numeric Value Representations
EFM-02S	Informational	✓ Yes	Inexistent Visibility Specifier
EFM-03S	Minor	✓ Yes	Inexistent Sanitization of Input Addresses
NOM-01S	Informational	✓ Yes	Literal Equality of bool Variable
NOM-02S	Minor	✓ Yes	Inexistent Sanitization of Input Address
PRM-01S	Informational	Yes	Inexistent Visibility Specifier
PRM-02S	Minor	✓ Yes	Inexistent Sanitization of Input Addresses
SMR-01S	Informational	✓ Yes	Inexistent Visibility Specifier
SME-01S	Informational	✓ Yes	Inexistent Visibility Specifier
SME-02S	Minor	✓ Yes	Inexistent Sanitization of Input Addresses
TNF-01S	Informational	Yes	Inexistent Visibility Specifier
TNF-02S	Minor	✓ Yes	Inexistent Sanitization of Input Address

Severity

ID

Addressed

Title

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in EtherFi's ETH2.0 staking system.

As the project at hand implements a novel ETH2.0 node operation system, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed multiple high-severity vulnerabilities** within the system which could have had **severe ramifications** to its overall operation the most crucial of which revolved around the notion of withdrawal credentials and how they can be manipulated to point to a different address than the one EtherFi expects.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to a certain extent, however, we strongly recommend it to be expanded at certain complex points such as the multi-branch fund distribution mechanism in

EtherFiNode::getFullWithdrawalPayouts using arbitrary value literals.

A total of **87 findings** were identified over the course of the manual review of which **40 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
AMR-01M	Unknown	Yes	Inexplicable Capability of Re-Invocation
AMR-02M	Informational	✓ Yes	Inexistent Disable of Initializer
AMR-03M	Informational	✓ Yes	Insufficient Validation of Bid Size
AMR-04M	Minor	✓ Yes	Improper Entry Clean-Up
AMR-05M	Minor	✓ Yes	Insufficient Validation of Minimum Bid Amount

ID	Severity	Addressed	Title
BNF-01M	Informational	✓ Yes	Inexistent Disable of Initializer
BNF-02M	Major	✓ Yes	Incorrect Override of Functionality
CRP-01M	Informational	✓ Yes	Inexistent Disable of Initializer
CRP-02M	Major	✓ Yes	Inexistent Slippage Protection
CRP-03M	Major	Nullified	Inexplicable Deposit Flow
CRP-04M	Major	✓ Yes	Unsupported Withdrawal Mechanism
EAP-01M	Unknown	Nullified	Improper Accuracy of Point Calculations
EAP-02M	Unknown	Nullified	Pure Off-Chain Point Utilization
EAP-03M	Minor	Nullified	Inexistent Prevention of Re-Invocation
EAP-04M	Minor	Nullified	Potentially Redundant Amount Restriction
EAP-05M	Minor	Nullified	Unfair Reset of Deposit Time
EFN-01M	Informational	✓ Yes	Inexistent Disable of Initializer
EFN-02M	Minor	✓ Yes	Incorrect Balance Assumption
EFN-03M	Minor	✓ Yes	Inexistent Sanitization of Exit Timestamp
EFN-04M	Major	✓ Yes	Inexistent Caller Validation
EFN-05M	Major	! Acknowledged	Weak Validation of Node State

ID	Severity	Addressed	Title
EFM-01M	Informational	✓ Yes	Inexistent Disable of Initializer
EFM-02M	Minor	Nullified	Inexistent Prevention of Duplicate Exit
EFM-03M	Minor	✓ Yes	Inexistent Sanitization of Non-Exit Penalty Rate
EFM-04M	Major	✓ Yes	Inexistent Validation of Node State
EFM-05M	Major	! Acknowledged	Weak Validation of Node State
NOM-01M	Unknown	✓ Yes	Inexplicable Capability of Re-Invocation
NOM-02M	Major	Nullified	Incorrect Verification of Whitelist
PRM-01M	Unknown	✓ Yes	Inexplicable Capability of Re-Invocation
PRM-02M	Informational	✓ Yes	Inexistent Disable of Initializer
PRM-03M	Minor	✓ Yes	Inexistent Sanitization of Fee Proportion
SMR-01M	Informational	✓ Yes	Inexistent Disable of Initializer
SMR-02M	Informational	✓ Yes	Inexplicable Data Types
SMR-03M	Minor	✓ Yes	Inexistent Sanitization of Valid Type
SMR-04M	Medium	✓ Yes	Improper Score Maintenance Mechanisms
SME-01M	Unknown	✓ Yes	Inexplicable Capability of Re-Invocation
SME-02M	Informational	⊘ Yes	Inexistent Disable of Initializer

ID	Severity	Addressed	Title
SME-03M	Major	Nullified	Incorrect Data Entry
SME-04M	Major	× No	ETH2.0 Validator Front-Run Withdrawal Credential Attack
TNF-01M	Informational	✓ Yes	Inexistent Disable of Initializer

Code Style

During the manual portion of the audit, we identified **47 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
AMR-01C	Informational	• Partial	Inefficient Optimization of Iterator Increment
AMR-02C	Informational	• Partial	Inefficient mapping Lookups
AMR-03C	Informational	! Acknowledged	Loop Iterator Optimization
AMR-04C	Informational	✓ Yes	Non-Standard Gap Size
AMR-05C	Informational	✓ Yes	Redundant Duplicate Application of Access Control
BNF-01C	Informational	Yes	Non-Standard Gap Size
CRP-01C	Informational	✓ Yes	Duplicate Invocation of Getter
CRP-02C	Informational	© Partial	Inexistent Gap Declaration
EAP-01C	Informational	Nullified	Code Readability Enhancement
EAP-02C	Informational	Nullified	Generic Typographic Mistakes
EAP-03C	Informational	Nullified	Inefficient Contract TVL Calculation

ID	Severity	Addressed	Title
EAP-04C	Informational	Nullified	Inefficient mapping Lookups
EAP-05C	Informational	Nullified	Insufficient Documentation of Literal
EAP-06C	Informational	Nullified	Redundant Data Point
EAP-07C	Informational	Nullified	Redundant Duplicate Data Points
EAP-08C	Informational	Nullified	Redundant Parenthesis Statements
EAP-09C	Informational	Nullified	Variable Mutability Specifiers (Immutable)
EFN-01C	Informational	Acknowledged	Generic Typographic Mistakes
EFN-02C	Informational	Yes	Ineffectual Conditional Check
EFN-03C	Informational	! Acknowledged	Ineffectual Usage of Safe Arithmetics
EFN-04C	Informational	© Partial	Inefficient Calculation of Rewards
EFN-05C	Informational	Yes	Inefficient Case Handling
EFN-06C	Informational	Nullified	Inefficient Loop Iterator Data Type
EFN-07C	Informational	Nullified	Loop Iterator Optimizations
EFN-08C	Informational	✓ Yes	Optimization of Penalty Calculation
EFN-09C	Informational	⊗ No	Potentially Incorrect Constants
EFN-10C	Informational	© Partial	Redundant Parenthesis Statements

ID	Severity	Addressed	Title
EFN-11C	Informational	! Acknowledged	Repetitive Value Literals
EFM-01C	Informational	Yes	Inexistent Error Messages
EFM-02C	Informational	! Acknowledged	Loop Iterator Optimizations
EFM-03C	Informational	Yes	Non-Standard Gap Size
EFM-04C	Informational	Yes	Redundant Parenthesis Statements
EFM-05C	Informational	1 Acknowledged	Repetitive Value Literal
NOM-01C	Informational	Yes	Inefficient mapping Lookups
PRM-01C	Informational	Yes	Non-Standard Gap Size
PRM-02C	Informational	© Partial	Optimization of Code Block
PRM-03C	Informational	Yes	Repetitive Invocation of Getter Function
PRM-04C	Informational	! Acknowledged	Repetitive Value Literal
SMR-01C	Informational	! Acknowledged	Generic Typographic Mistake
SMR-02C	Informational	! Acknowledged	Ineffectual Usage of Safe Arithmetics
SMR-03C	Informational	Yes	Non-Standard Gap Size
SMR-04C	Informational	! Acknowledged	Redundant Storage Reads
SME-01C	Informational	Yes	Inexistent Error Message

ID	Severity	Addressed	Title
SME-02C	Informational	! Acknowledged	Loop Iterator Optimizations
SME-03C	Informational	✓ Yes	Non-Standard Gap Size
TNF-01C	Informational	✓ Yes	Non-Standard Gap Size
TYR-01C	Informational	Acknowledged	Redundant Evaluation of Balance

AuctionManager Static Analysis Findings

AMR-01S: Inexistent Event Emissions

Туре	Severity	Location
Language Specific	Informational	AuctionManager.sol:L221-L223, L227-L229

Description:

The linked functions adjust sensitive contract variables yet do not emit an event for it.

```
src/AuctionManager.sol

SOL

221 function disableWhitelist() public onlyOwner {
222  whitelistEnabled = false;
223 }
```

We advise an event to be declared and correspondingly emitted for each function to ensure off-chain processes can properly react to this system adjustment.

Alleviation:

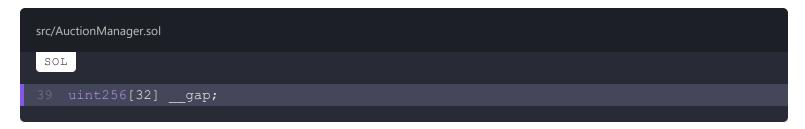
Two events have been introduced to the codebase each signalling the whitelist's enabled and disabled state respectively, alleviating this exhibit.

AMR-02S: Inexistent Visibility Specifier

Туре	Severity	Location
Code Style	Informational	AuctionManager.sol:L39

Description:

The linked variable has no visibility specifier explicitly set.



We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between pragma versions.

Alleviation:

A public visibility specifier has been introduced for the referenced member of the contract, addressing this exhibit. Given that the member represents a variable that serves no purpose outside of the contract's context we advise the variable to be set as internal instead of public as a matter of optimization.

AMR-03S: Literal Equality of **bool** Variables

Туре	Severity	Location
Gas Optimization	Informational	AuctionManager.sol:L89, L100, L168, L202

Description:

The linked bool comparisons are performed between variables and bool literals.



We advise each bool variable to be utilized directly either in its negated (!) or original form.

Alleviation:

All referenced equality comparisons of bool variables have been optimized to utilize each bool variable's value directly as advised.

AMR-04S: Inexistent Sanitization of Input Addresses

Туре	Severity	Location
Input Sanitization	Minor	AuctionManager.sol:L60-L77, L285-L291, L295-L299

Description:

The linked function(s) accept address arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in **constructor** implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

We advise some basic sanitization to be put in place by ensuring that each address specified is non-zero.

Alleviation:

All referenced instances of address arguments are properly sanitized via require checks ensuring they are non-zero, fully alleviating this exhibit.

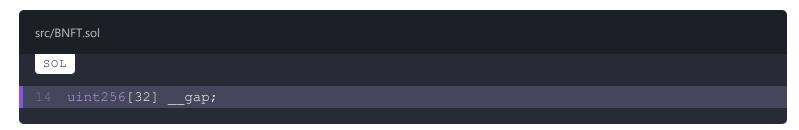
BNFT Static Analysis Findings

BNF-01S: Inexistent Visibility Specifier

Туре	Severity	Location
Code Style	Informational	BNFT.sol:L14

Description:

The linked variable has no visibility specifier explicitly set.



We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between pragma versions.

Alleviation:

A public visibility specifier has been introduced for the referenced member of the contract, addressing this exhibit. Given that the member represents a variable that serves no purpose outside of the contract's context we advise the variable to be set as internal instead of public as a matter of optimization.

BNF-02S: Inexistent Sanitization of Input Address

Туре	Severity	Location
Input Sanitization	Minor	BNFT.sol:L20-L26

Description:

The linked function accepts an address argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in **constructor** implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

```
src/BNFT.sol

SOL

20 function initialize(address _stakingManagerAddress) initializer external {
21     __ERC721_init("Bond NFT", "BNFT");
22     __Ownable_init();
23     __UUPSUpgradeable_init();
24

25     stakingManagerAddress = _stakingManagerAddress;
26 }
```

We advise some basic sanitization to be put in place by ensuring that the address specified is non-zero.

Alleviation:

The referenced instance of an address argument is properly sanitized via a require check ensuring that it is non-zero, fully alleviating this exhibit.

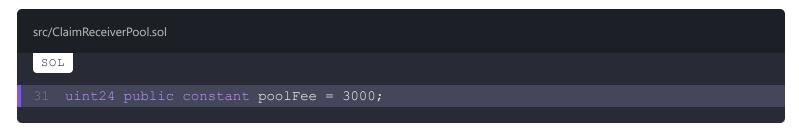
ClaimReceiverPool Static Analysis Findings

CRP-01S: Illegible Numeric Value Representation

Туре	Severity	Location
Code Style	Informational	ClaimReceiverPool.sol:L31

Description:

The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.



Alleviation:

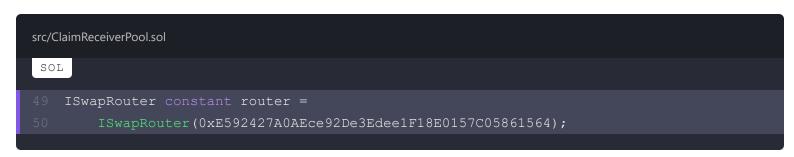
The underscore separator has been properly introduced to the referenced value, optimizing its legibility.

CRP-02S: Inexistent Visibility Specifiers

Туре	Severity	Location
Code Style	Informational	ClaimReceiverPool.sol:L49-L50, L53-L54

Description:

The linked variables have no visibility specifier explicitly set.



We advise them to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between pragma versions.

Alleviation:

All referenced variables have had a public visibility specifier set, alleviating this exhibit in full.

CRP-03S: Inexistent Sanitization of Input Addresses

Туре	Severity	Location
Input Sanitization	Minor	ClaimReceiverPool.sol:L72-L90

Description:

The linked function(s) accept address arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in **constructor** implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

```
src/ClaimReceiverPoolsol

SoL

72  function initialize(
73   address _rEth,
74   address _wstEth,
75   address _cbEth,
76   address _cbEth,
77   address _scoreManager
78  ) external initializer {
79    rETH = _rEth;
80    wstETH = _wstEth;
81    sfrxETH = _sfrxEth;
82    cbETH = _cbEth;
83

84    scoreManager = IScoreManager(_scoreManager);
85

86    __Pausable_init();
87    __Ownable_init();
88    __UUPSUpgradeable_init();
89    __ReentrancyGuard_init();
90  }
```

We advise some basic sanitization to be put in place by ensuring that each address specified is non-zero.

Alleviation:

All referenced instances of address arguments are properly sanitized via require checks ensuring they are non-zero, fully alleviating this exhibit.

EarlyAdopterPool Static Analysis Findings

EAP-01S: Illegible Numeric Value Representations

Туре	Severity	Location
Code Style	Informational	EarlyAdopterPool.sol:L183, L215-L216

Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

```
src/EarlyAdopterPool.sol

SOL

183 claimDeadline = block.timestamp + (_claimDeadline * 86400);
```

To properly illustrate each value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of 1e18, we advise fractions to be utilized directly (i.e. 1e17 becomes 0.1e18) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (_) separator to discern the percentage decimal (i.e. 10000 becomes 100_00, 300 becomes 3_00 and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. 1000000 becomes 1_000_000).

Alleviation:

The EtherFi team has opted not to remediate any finding in the EarlyAdopterPool implementation as they have deemed its on-chain data points unusable. As such, we consider this exhibit nullified given that it pertains a system component that will not be utilized in the EtherFi system.

EAP-02S: Inexistent Sanitization of Input Addresses

Туре	Severity	Location
Input Sanitization	Minor	EarlyAdopterPool.sol:L85-L100

Description:

The linked function(s) accept address arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in **constructor** implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

```
src/EarlyAdopterPool.sol

sol

85   constructor(
86    address _reTH,
87    address _wsteTH,
88    address _sfrxeTH,
89    address _cbETH
90 ) {
91    reTH = _reTH;
92    wsteTH = _wsteTH;
93    sfrxeTH = _sfrxeTH;
94    cbETH = _cbETH;
95

96    reTHInstance = IERC20(_reTH);
97    wsteTHInstance = IERC20(_sfrxeTH);
98    sfrxeTHInstance = IERC20(_sfrxeTH);
99    cbETHInstance = IERC20(_cbETH);
100 }
```

We advise some basic sanitization to be put in place by ensuring that each address specified is non-zero.

Alleviation:

The EtherFi team has opted not to remediate any finding in the EarlyAdopterPool implementation as they have deemed its on-chain data points unusable. As such, we consider this exhibit nullified given that it pertains a system component that will not be utilized in the EtherFi system.

EAP-03S: Potential Lock of Native Assets

Туре	Severity	Location
Language Specific	Minor	EarlyAdopterPool.sol:L74

Description:

The linked receive / fallback function performs no sanitization as to its caller and no function within the contract expects funds to have been received directly by the contract.

Impact:

Any native funds accidentally sent to the contract may be forever locked.

```
src/EarlyAdopterPool.sol

SOL

74 receive() external payable {}
```

We advise the code to properly prohibit accidental native assets from being permanently locked in the contract by introducing a require check restricting the msg.sender to the contract(s) expected to transfer assets to the system (i.e. in case of a wrapped native version of an asset, only the wxxx contract address should be allowed). Alternatively, if the contract is not expected to receive native assets directly the function should be removed in its entirety.

Alleviation:

The EtherFi team has opted not to remediate any finding in the EarlyAdopterPool implementation as they have deemed its on-chain data points unusable. As such, we consider this exhibit nullified given that it pertains a system component that will not be utilized in the EtherFi system.

EAP-04S: Improper Invocations of EIP-20 transfer / transferFrom

Туре	Severity	Location
Standard Conformity	Medium	EarlyAdopterPool.sol:L127, L268-L271

Description:

The linked statements do not properly validate the returned bool values of the **EIP-20** standard transfer & transferFrom functions. As the **standard dictates**, callers **must not** assume that false is never returned.

Impact:

If the code mandates that the returned <code>bool</code> is <code>true</code>, this will cause incompatibility with tokens such as USDT / Tether as no such <code>bool</code> is returned to be evaluated causing the check to fail at all times. On the other hand, if the token utilized can return a <code>false</code> value under certain conditions but the code does not validate it, the contract itself can be compromised as having received / sent funds that it never did.

```
src/EarlyAdopterPool.sol

SOL

127 require(IERC20(_erc20Contract).transferFrom(msg.sender, address(this), _amount), "TransferFrom(msg.sender))
```

Since not all standardized tokens are **EIP-20** compliant (such as Tether / USDT), we advise a safe wrapper library to be utilized instead such as SafeERC20 by OpenZeppelin to opportunistically validate the returned only if it exists in each instance.

Alleviation:

The EtherFi team has opted not to remediate any finding in the EarlyAdopterPool implementation as they have deemed its on-chain data points unusable. As such, we consider this exhibit nullified given that it pertains a system component that will not be utilized in the EtherFi system.

EtherFiNode Static Analysis Findings

EFN-01S: Illegible Numeric Value Representation

Туре	Severity	Location
Code Style	Informational	EtherFiNode.sol:L436

Description:

The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.

```
src/EtherFiNode.sol

SOL

436 return uint256(timeElapsed / (24 * 3600));
```

Alleviation:

The underscore separator has been properly introduced to the referenced value, optimizing its legibility.

EFN-02S: Inexistent Sanitization of Input Address

Туре	Severity	Location
Input Sanitization	Minor	EtherFiNode.sol:L25-L29

Description:

The linked function accepts an address argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in **constructor** implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

```
src/EtherFiNode.sol

SOL

25  function initialize(address _etherFiNodesManager) public {
26    require(stakingStartTimestamp == 0, "already initialised");
27    stakingStartTimestamp = uint32(block.timestamp);
28    etherFiNodesManager = _etherFiNodesManager;
29 }
```

We advise some basic sanitization to be put in place by ensuring that the address specified is non-zero.

Alleviation:

The referenced instance of an address argument is properly sanitized via a require check ensuring that it is non-zero, fully alleviating this exhibit.

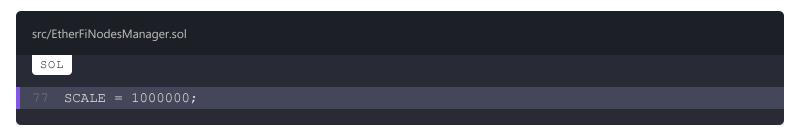
EtherFiNodesManager Static Analysis Findings

EFM-01S: Illegible Numeric Value Representations

Туре	Severity	Location
Code Style	Informational	EtherFiNodesManager.sol:L77, L93-L96, L107-L110

Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.



To properly illustrate each value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of 1e18, we advise fractions to be utilized directly (i.e. 1e17 becomes 0.1e18) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (_) separator to discern the percentage decimal (i.e. 10000 becomes 100_00, 300 becomes 3_00 and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. 1000000 becomes 1_000_000).

Alleviation:

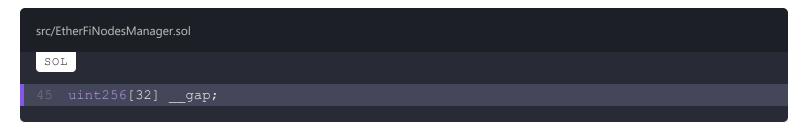
While the underscore character has been introduced to all referenced variables, it has been done so using conventional numbers rather than percentage-based values. We advise literals such as 815625, meant to represent 81.5625%, to be written as 815625 better illustrating their purpose.

EFM-02S: Inexistent Visibility Specifier

Туре	Severity	Location
Code Style	Informational	EtherFiNodesManager.sol:L45

Description:

The linked variable has no visibility specifier explicitly set.



We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between pragma versions.

Alleviation:

A public visibility specifier has been introduced for the referenced member of the contract, addressing this exhibit. Given that the member represents a variable that serves no purpose outside of the contract's context we advise the variable to be set as internal instead of public as a matter of optimization.

EFM-03S: Inexistent Sanitization of Input Addresses

Туре	Severity	Location
Input Sanitization	Minor	EtherFiNodesManager.sol:L63-L119

Description:

The linked function(s) accept address arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in **constructor** implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

```
src/EtherFiNodesManager.sol

SOL

63  function initialize(
64   address _treasuryContract,
65   address _auctionContract,
66   address _stakingManagerContract,
67   address _tnftContract,
68   address _bnftContract,
69   address _protocolRevenueManagerContract
70 ) external initializer {
```

We advise some basic sanitization to be put in place by ensuring that each address specified is non-zero.

Alleviation:

All referenced instances of address arguments are properly sanitized via require checks ensuring they are non-zero, fully alleviating this exhibit.

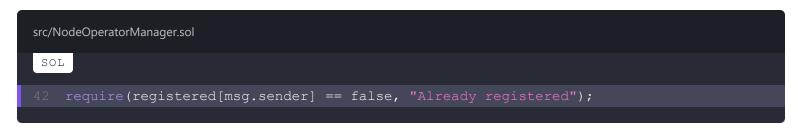
NodeOperatorManager Static Analysis Findings

NOM-01S: Literal Equality of **bool** Variable

Туре	Severity	Location
Gas Optimization	Informational	NodeOperatorManager.sol:L42

Description:

The linked bool comparison is performed between a variable and a bool literal.



We advise the bool variable to be utilized directly either in its negated (!) or original form.

Alleviation:

The referenced equality comparison of a bool variable has been optimized to utilize the bool variable's value directly as advised.

NOM-02S: Inexistent Sanitization of Input Address

Туре	Severity	Location
Input Sanitization	Minor	NodeOperatorManager.sol:L127-L131

Description:

The linked function accepts an address argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in **constructor** implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

We advise some basic sanitization to be put in place by ensuring that the address specified is non-zero.

Alleviation:

The referenced instance of an address argument is properly sanitized via a require check ensuring that it is non-zero, fully alleviating this exhibit.

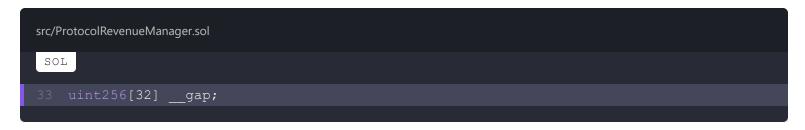
ProtocolRevenueManager Static Analysis Findings

PRM-01S: Inexistent Visibility Specifier

Туре	Severity	Location
Code Style	Informational	ProtocolRevenueManager.sol:L33

Description:

The linked variable has no visibility specifier explicitly set.



We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between pragma versions.

Alleviation:

A public visibility specifier has been introduced for the referenced member of the contract, addressing this exhibit. Given that the member represents a variable that serves no purpose outside of the contract's context we advise the variable to be set as internal instead of public as a matter of optimization.

PRM-02S: Inexistent Sanitization of Input Addresses

Туре	Severity	Location
Input Sanitization	Minor	ProtocolRevenueManager.sol:L121-L125, L130-L134

Description:

The linked function(s) accept address arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in **constructor** implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

```
src/ProtocolRevenueManager.sol

SOL

118 /// @notice Instantiates the interface of the node manager for integration
119 /// @dev Set manually due to cirular dependencies
120 /// @param _etherFiNodesManager etherfi node manager address to set
121 function setEtherFiNodesManagerAddress(
122    address _etherFiNodesManager
123 ) external onlyOwner {
124    etherFiNodesManager = IEtherFiNodesManager(_etherFiNodesManager);
125 }
```

We advise some basic sanitization to be put in place by ensuring that each address specified is non-zero.

Alleviation:

All referenced instances of address arguments are properly sanitized via require checks ensuring they are non-zero, fully alleviating this exhibit.

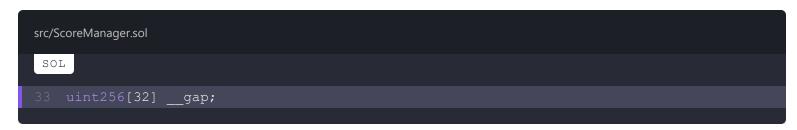
ScoreManager Static Analysis Findings

SMR-01S: Inexistent Visibility Specifier

Туре	Severity	Location
Code Style	Informational	ScoreManager.sol:L33

Description:

The linked variable has no visibility specifier explicitly set.



We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between pragma versions.

Alleviation:

A public visibility specifier has been introduced for the referenced member of the contract, addressing this exhibit. Given that the member represents a variable that serves no purpose outside of the contract's context we advise the variable to be set as internal instead of public as a matter of optimization.

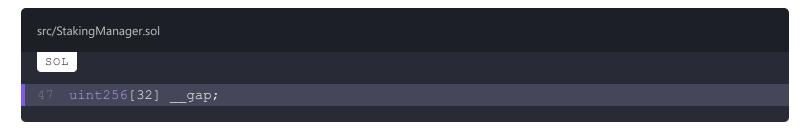
StakingManager Static Analysis Findings

SME-01S: Inexistent Visibility Specifier

Туре	Severity	Location
Code Style	Informational	Staking Manager. sol: L47

Description:

The linked variable has no visibility specifier explicitly set.



We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between pragma versions.

Alleviation:

A public visibility specifier has been introduced for the referenced member of the contract, addressing this exhibit. Given that the member represents a variable that serves no purpose outside of the contract's context we advise the variable to be set as internal instead of public as a matter of optimization.

SME-02S: Inexistent Sanitization of Input Addresses

Туре	Severity	Location
Input Sanitization	Minor	StakingManager.sol:L76-L90, L223-L229, L239-L244, L246-L248, L250-L252

Description:

The linked function(s) accept address arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in **constructor** implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

We advise some basic sanitization to be put in place by ensuring that each address specified is non-zero.

Alleviation:

All referenced instances of address arguments are properly sanitized via require checks ensuring they are non-zero, fully alleviating this exhibit.

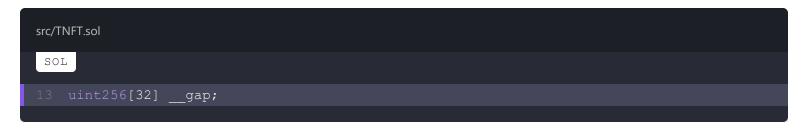
TNFT Static Analysis Findings

TNF-01S: Inexistent Visibility Specifier

Туре	Severity	Location
Code Style	Informational	TNFT.sol:L13

Description:

The linked variable has no visibility specifier explicitly set.



We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between pragma versions.

Alleviation:

A public visibility specifier has been introduced for the referenced member of the contract, addressing this exhibit. Given that the member represents a variable that serves no purpose outside of the contract's context we advise the variable to be set as internal instead of public as a matter of optimization.

TNF-02S: Inexistent Sanitization of Input Address

Туре	Severity	Location	
Input Sanitization	Minor	TNFT.sol:L19-L25	

Description:

The linked function accepts an address argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in **constructor** implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

```
src/TNFT.sol

sol

19 function initialize(address _stakingManagerAddress) initializer external {
20     __ERC721_init("Transferrable NFT", "TNFT");
21     __Ownable_init();
22     __UUPSUpgradeable_init();
23
24     stakingManagerAddress = _stakingManagerAddress;
25 }
```

We advise some basic sanitization to be put in place by ensuring that the address specified is non-zero.

Alleviation:

The referenced instance of an address argument is properly sanitized via a require check ensuring that it is non-zero, fully alleviating this exhibit.

AuctionManager Manual Review Findings

AMR-01M: Inexplicable Capability of Re-Invocation

Туре	Severity	Location
Centralization Concern	Unknown	AuctionManager.sol:L285-L291, L295-L299

Description:

```
The AuctionManager::setProtocolRevenueManager & AuctionManager::setStakingManagerContractAddress permit the protocolRevenueManager & stakingManagerContractAddress variables respectively to be set after the contract's initialization due to circular dependencies, however, each function can be invoked an arbitrary number of times.
```

```
src/AuctionManager.sol

SOL

281 /// @notice Sets an instance of the protocol revenue manager
282 /// @dev Needed to process an auction fee
283 /// @param _protocolRevenueManager the addres of the protocol manager
284 /// @notice Performed this way due to circular dependencies
285 function setProtocolRevenueManager(
286    address _protocolRevenueManager

287 ) external onlyOwner {
288    protocolRevenueManager = IProtocolRevenueManager(
299    _protocolRevenueManager
290 );
291 }
292
293 /// @notice Sets the stakingManagerContractAddress address in the current contract
294 /// @param _stakingManagerContractAddress new stakingManagerContract address
295 function setStakingManagerContractAddress(
296    address _stakingManagerContractAddress
297 ) external onlyOwner {
298    stakingManagerContractAddress = _stakingManagerContractAddress;
299 }
```

As both the ProtocolRevenueManager & StakingManager contracts represent an upgradeable module, we advise the referenced functions to be invoke-able only once.

Alleviation:

All referenced functions have had require checks introduced that ensure they cannot be re-invoked beyond their initialization, alleviating this exhibit's concerns fully.

AMR-02M: Inexistent Disable of Initializer

Туре	Severity	Location
Standard Conformity	Informational	AuctionManager.sol:L60-L77

Description:

The AuctionManager contract is meant to be an upgradeable contract that is initialized via the AuctionManager::initialize function, however, the base implementation of AuctionManager is not disabling the initializer during its construction.

Impact:

While not an active threat in this particular instance, base implementations that may perform a delegatecall to an administrator-defined party can be compromised even if proxied. As such, it is best practice to always initialize base implementations of proxies automatically on deployment.

We advise a constructor to be introduced to AuctionManager that executes

Initializable::_disableInitializers, ensuring that the base implementation of AuctionManager cannot be initialized maliciously.

Alleviation:

A constructor was introduced that properly disables the contract's initializers via the Initializable::_disableInitializers function, disallowing the contract from being initialized at its logic contract location.

AMR-03M: Insufficient Validation of Bid Size

Туре	Severity	Location
Input Sanitization	Informational	AuctionManager.sol:L121

Description:

The _bidSize the user specifies for an AuctionManager::createBid call is insufficiently sanitized as it is permitted to be 0, causing the function to "succeed" as a no-op.

Impact:

While the _bidsize is not sanitized adequately, no vulnerability arises from this behaviour and as such the finding has been classified as "informational".

```
src/AuctionManager.sol

sol

118 uint64 keysRemaining = nodeOperatorManagerInterface.getNumKeysRemaining(
119    msg.sender
120 );

121 require(_bidSize <= keysRemaining, "Insufficient public keys");</pre>
```

We advise the referenced require check to ensure that _bidsize is a non-zero number, guaranteeing that an AuctionManager::createBid execution will be accompanied by at least one bid creation.

Alleviation:

A require check was adequately introduced to the AuctionManager::createBid function ensuring that the bidsize is non-zero and thus preventing no-op AuctionManager::createBid transactions from successfully executing.

AMR-04M: Improper Entry Clean-Up

Туре	Severity	Location
Logical Fault	Minor	AuctionManager.sol:L166-L183

Description:

The AuctionManager::cancelBid function will incorrectly clean up the data entries associated with a particular bid ID as it will only set its isActive status to false.

Impact:

Apart from not properly deleting the bid entries, the code also decrements the numberOfActiveBids after the external distribution of funds to the msg.sender has been performed. As a result, if the StakingManager::batchDepositWithBidIds function is invoked during this time the code will insufficiently sanitize the number of bids that are attempted to be made.

Example:

src/AuctionManager.sol

```
163 /// @notice Cancels a specified bid by de-activating it
164 /// @dev Require the bid to exist and be active
165 /// @param _bidId the ID of the bid to cancel
166 function cancelBid(uint256 _bidId) public whenNotPaused {
167     require(bids[_bidId].bidderAddress == msg.sender, "Invalid bid");
168     require(bids[_bidId].isActive == true, "Bid already cancelled");
169
170     // Cancel the bid by de-activating it
171     bids[_bidId].isActive = false;
172
173     // Get the value of the cancelled bid to refund
174     uint256 bidValue = bids[_bidId].amount;
175
176     // Refund the user with their bid amount
177     (bool sent, ) = msg.sender.call(value: bidValue)("");
178     require(sent, "Failed to send Ether");
179
180     numberOfActiveBids--;
181
182     emit BidCancelled(_bidId);
183 }
```

We advise the code to delete the bid entirely (i.e. delete bids[_bidId]) after the bidValue has been extracted and to also decrement the numberOfActiveBids before the funds are distributed to the msg.sender, ensuring that the code conforms to the Checks-Effects-Interactions pattern and that the bid is properly removed from the system.

Alleviation:

The numberOfActiveBids value is properly decremented prior to the disbursement of funds to the msg.sender, preventing the contract from having an interim corrupt state and thus conforming to the CEI pattern.

AMR-05M: Insufficient Validation of Minimum Bid Amount

Туре	Severity	Location
Input Sanitization	Minor	AuctionManager.sol:L304

Description:

The AuctionManager::setMinBidPrice function will permit a new minBidAmount to be set, however, the validation it performs does not include the whitelistBidAmount which must be less-than the newly set minBidAmount.

Impact:

It is possible to misconfigure the contract and have a whitelist bid amount that is greater-than the current minimum bid amount, eliminating the benefits of whitelisted bids.

Example:

src/AuctionManager.sol

```
301 /// @notice Updates the minimum bid price
302 /// @param _newMinBidAmount the new amount to set the minimum bid price as
303 function setMinBidPrice(uint64 _newMinBidAmount) external onlyOwner {
304     require(_newMinBidAmount < maxBidAmount, "Min bid exceeds max bid");
305     minBidAmount = _newMinBidAmount;
306 }
307
308 /// @notice Updates the maximum bid price
309 /// @param _newMaxBidAmount the new amount to set the maximum bid price as
310 function setMaxBidPrice(uint64 _newMaxBidAmount) external onlyOwner {
311     require(_newMaxBidAmount > minBidAmount, "Min bid exceeds max bid");
312     maxBidAmount = _newMaxBidAmount;
313 }
314
315 /// @notice Updates the minimum bid price for a whitelisted address
316 /// @param _newAmount the new amount to set the minimum bid price as
317 function updateWhitelistMinBidAmount(
318     uint128 _newAmount
319 ) external onlyOwner {
320     require(_newAmount < minBidAmount;
321     whitelistBidAmount = _newAmount;
322 }
```

We advise the require check referenced to be updated, ensuring that _newMinBidAmount is greater-than the current whitelistBidAmount.

Alleviation:

The AuctionManager::setMinBidPrice function was properly updated to ensure that the newMinBidAmount is greater-than the whitelistBidAmount, upholding the contract's guarantee that a whitelist bid amount is less than the minimum permitted for a regular bid.

BNFT Manual Review Findings

BNF-01M: Inexistent Disable of Initializer

Туре	Severity	Location
Standard Conformity	Informational	BNFT.sol:L20-L26

Description:

The BNFT contract is meant to be an upgradeable contract that is initialized via the BNFT::initialize function, however, the base implementation of BNFT is not disabling the initializer during its construction.

Impact:

While not an active threat in this particular instance, base implementations that may perform a delegatecall to an administrator-defined party can be compromised even if proxied. As such, it is best practice to always initialize base implementations of proxies automatically on deployment.

Example:

src/BNFT.sol

We advise a constructor to be introduced to BNFT that executes

Initializable::_disableInitializers, ensuring that the base implementation of BNFT cannot be initialized maliciously.

Alleviation:

A constructor was introduced that properly disables the contract's initializers via the Initializable::_disableInitializers function, disallowing the contract from being initialized at its logic contract location.

BNF-02M: Incorrect Override of Functionality

Туре	Severity	Location
Logical Fault	Major	BNFT.sol:L36-L44

Description:

The <code>BNFT::transferFrom</code> function override is meant to disallow any transfers to occur unless they are part of <code>BNFT::mint</code> operations, however, the methodology applied solely overrides the <code>ERC721Upgradeable::transferFrom</code> function and does not affect other functions, such as <code>ERC721Upgradeable::safeTransferFrom</code>.

Impact:

The BNFT asset is presently transferrable via the ERC721Upgradeable::safeTransferFrom function as the contract incorrectly overrides only the BNFT::transferFrom function.

```
src/BNFT.sol

SOL

28  /// @notice Mints NFT to required user

29  /// @dev Only through the staking contrate and not by an EOA

30  /// @param _reciever receiver of the NFT

31  /// @param _validatorId the ID of the NFT

32  function mint(address _reciever, uint256 _validatorId) external onlyStakingManager {

33     _safeMint(_reciever, _validatorId);

34  }

35  

36  //ERC721 transfer function being overidden to make it soulbound

37  function transferFrom(

38    address from,

39    address to,

40    uint256 tokenId

41  ) public virtual override(ERC721Upgradeable) {

42    require(from == address(0), "Err: token is SOUL BOUND");

43    super.transferFrom(from, to, tokenId);

44  }
```

We advise the ERC721Upgradeable::_beforeTokenTransfer hook to be overridden instead, allowing it to be invoked solely when from == address(0) and thus capturing all types of "transfer" cases that the
ERC721Upgradeable may implement.

Alleviation:

The contract now properly overrides the <code>ERC721Upgradeable::_beforeTokenTransfer</code> function, ensuring its transfer restrictions are applied in all types of transfers performed by the <code>EIP-20</code> asset.

ClaimReceiverPool Manual Review Findings

CRP-01M: Inexistent Disable of Initializer

Туре	Severity	Location
Standard Conformity	Informational	ClaimReceiverPool.sol:L72-L90

Description:

The ClaimReceiverPool contract is meant to be an upgradeable contract that is initialized via the ClaimReceiverPool::initialize function, however, the base implementation of ClaimReceiverPool is not disabling the initializer during its construction.

Impact:

While not an active threat in this particular instance, base implementations that may perform a delegatecall to an administrator-defined party can be compromised even if proxied. As such, it is best practice to always initialize base implementations of proxies automatically on deployment.

Example:

src/ClaimReceiverPool.sol

```
71 /// @notice initialize to set variables on deployment
72 function initialize(
73 address _rEth,
74 address _wstEth,
75 address _sfrxEth,
76 address _cbEth,
77 address _scoreManager

78 ) external initializer {
79 rETH = _rEth;
80 wstETH = _wstEth;
81 sfrxETH = _sfrxEth;
82 cbETH = _cbEth;
83 cbETH = _cbEth;
84 scoreManager = IScoreManager(_scoreManager);
85
86 __Pausable_init();
87 __Ownable_init();
88 __UUPSUpgradeable_init();
89 __ReentrancyGuard_init();
90 }

__ReentrancyGuard_init();
```

We advise a constructor to be introduced to ClaimReceiverPool that executes

Initializable::_disableInitializers, ensuring that the base implementation of ClaimReceiverPool cannot be initialized maliciously.

Alleviation:

A constructor was introduced that properly disables the contract's initializers via the Initializable::_disableInitializers function, disallowing the contract from being initialized at its logic contract location.

CRP-02M: Inexistent Slippage Protection

Туре	Severity	Location
Logical Fault	Major	ClaimReceiverPool.sol:L247-L248

Description:

The <code>ClaimReceiverPool::_swapExactInputSingle</code> function that is extensively in use by the <code>ClaimReceiverPool::deposit</code> function performs an on-chain Uniswap V3 swap without specifying any form of slippage protection, rendering each user's deposits fully susceptible to slippage attacks.

Impact:

Whenever a user wishes to deposit the funds they had staked in the early adopter pool, all their non-native assets will be fully susceptible to on-chain sandwich attacks that would greatly impact the end-result of their deposit to the protocol.

Example:

src/ClaimReceiverPool.sol

```
SOL
233 function swapExactInputSingle(
       uint256 amountIn,
       address tokenIn
236 ) internal returns (uint256 amountOut) {
       IERC20 ( tokenIn) .approve (address (router) , _amountIn);
      ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
            .ExactInputSingleParams({
               tokenIn: tokenIn,
               tokenOut: wEth,
               fee: poolFee,
               recipient: address(this),
               deadline: block.timestamp,
               amountIn: amountIn,
               amountOutMinimum: 0,
               sqrtPriceLimitX96: 0
           });
       amountOut = router.exactInputSingle(params);
```

As the user's assets are swapped directly and are the only assets affected by the slippage, we advise the ClaimReceiverPool::deposit function to accept an array of arguments that indicate what slippage should be applied on each asset, permitting the users to control the amount they will ultimately deposit to the system.

To note, this feature would need to be accompanied by a fully-fledged front-end that enables the user to specify these slippage levels properly akin to other DeFi protocols.

Alleviation:

The ClaimReceiverPool::deposit flow was adjusted per our recommendation, accepting slippage arguments in the form of basis points for each of the swaps to be performed and thus alleviating this exhibit.

CRP-03M: Inexplicable Deposit Flow

Туре	Severity	Location
Logical Fault	Major	ClaimReceiverPool.sol:L105-L155

Description:

The deposit flow of the ClaimReceiverPool appears to contradict the EarlyAdopterPool implementation as the EarlyAdopterPool implementation was meant to point to the contract and transfer all the user's funds directly to it.

Impact:

As the <code>EarlyAdopterPool</code> and <code>ClaimReceiverPool</code> implementations are incompatible, the <code>EarlyAdopterPool</code> would transfer user funds to the <code>ClaimReceiverPool</code> and the users would not be able to access / claim them as part of their deposit.

```
src/ClaimReceiverPool.sol

sol

223 function _swapERC20ForETH(address _token, uint256 _amount) internal returns (uint256)
224    if (_amount == 0) {
225        return 0;
226    }

227    IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);
228    uint256 amountOut = _swapExactInputSingle(_amount, _token);
229    wethContract.withdraw(amountOut);
230    return amountOut;
231 }
```

We advise the ClaimReceiverPool to be revised, relying on the ClaimReceiverPool::deposit arguments that are verified by the Merkle Proof and not transferring any assets of the user via _swaperc20Foreth / the call's msg.value as these assets would have already been automatically deposited by the EarlyAdopterPool.

Alleviation:

The EtherFi team evaluated this exhibit and has decided to proceed with a different deposit flow rendering the <code>EarlyAdopterPool</code> implementation's data points unusable. As such, we consider this exhibit nullified as it is no longer relevant.

CRP-04M: Unsupported Withdrawal Mechanism

Туре	Severity	Location
Logical Fault	Major	ClaimReceiverPool.sol:L229

Description:

The ClaimReceiverPool::_swapERC20ForETH function will attempt to unwrap the WETH asset the contract has received via the WETH::withdraw function, however, such an operation will fail as the ClaimReceiverPool does not have any receive function declared.

Impact:

The ClaimReceiverPool is presently incapable of adequately unwrapping the assets it receives from an **EIP- 20** to WETH swap, rendering the contract's conversion code inoperable.

```
src/ClaimReceiverPool.sol

sol

223 function _swapERC20ForETH(address _token, uint256 _amount) internal returns (uint256)
224    if (_amount == 0) {
225        return 0;
226    }
227    IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);
228    uint256 amountOut = _swapExactInputSingle(_amount, _token);
229    wethContract.withdraw(amountOut);
230    return amountOut;
231 }
```

We advise a receive function to be declared that ensures its msg.sender is the wethContract address, permitting the weth asset to be properly unwrapped post-swap.

Alleviation:

A receive function was properly introduced to the contract ensuring that it can successfully receive native funds as part of its WETH::withdraw operation.

EarlyAdopterPool Manual Review Findings

EAP-01M: Improper Accuracy of Point Calculations

Туре	Severity	Location
Mathematical Operations	Unknown	EarlyAdopterPool.sol:L224

Description:

The EarlyAdopterPool::calculateUserPoints function will yield a value whose accuracy is inflated as the decimal normalization performed at the end is incorrect.

Impact:

The severity of this exhibit will be adjusted depending on the desirable accuracy of points by the EtherFi team.

Example:

src/EarlyAdopterPool.sol

```
SOL
204 function calculateUserPoints(address user) public view returns (uint256) {
       uint256 lengthOfDeposit;
      if (claimingOpen == 0) {
           lengthOfDeposit = block.timestamp - depositInfo[ user].depositTime;
            lengthOfDeposit = endTime - depositInfo[ user].depositTime;
       uint256 userMultiplier = Math.min(
           2000,
           1000 + ((lengthOfDeposit * 10) / 2592) / 10
       );
       uint256 totalUserBalance = depositInfo[ user].etherBalance +
            depositInfo[ user].totalERC20Balance;
            ((Math.sqrt(totalUserBalance) * lengthOfDeposit) *
               userMultiplier) / 1e14;
```

In detail, if the "points" of a user are desired to be in "per-second" accuracy this is not presently achieved by the contract. The contract should perform a division by 1e18 (to normalize the totalUserBalance) and another division by 1e3 or 100_0 to normalize the userMultiplier.

Presently, the contract performs a division by 1e14 that causes the final point result to have an accuracy of
which is arbitrary. We advise the code to be corrected and the accuracy of
EarlyAdopterPool::calculateUserPoints to be clearly documented.

Alleviation:

The EtherFi team has opted not to remediate any finding in the <code>EarlyAdopterPool</code> implementation as they have deemed its on-chain data points unusable. As such, we consider this exhibit nullified given that it pertains a system component that will not be utilized in the EtherFi system.

EAP-02M: Pure Off-Chain Point Utilization

Туре	Severity	Location
Centralization Concern	Unknown	EarlyAdopterPool.sol:L174, L177

Description:

The "points" a user collects as their deposit remains in the EarlyAdopterPool contract are utilized solely in an off-chain manner as they are emitted in the Fundsclaimed event and are not utilized anywhere else.

We advise the utilization of points to be revisited, potentially enforcing an integration between <code>EarlyAdopterPool</code> and <code>ScoreManager</code> for this particular pool only as presently it is difficult to ascertain whether the points gathered in the <code>EarlyAdopterPool</code> have been properly replayed in the <code>ScoreManager</code> contract.

Alleviation:

The EtherFi team has opted not to remediate any finding in the EarlyAdopterPool implementation as they have deemed its on-chain data points unusable. As such, we consider this exhibit nullified given that it pertains a system component that will not be utilized in the EtherFi system.

EAP-03M: Inexistent Prevention of Re-Invocation

Туре	Severity	Location
Logical Fault	Minor	EarlyAdopterPool.sol:L180-L188

Description:

The <code>EarlyAdopterPool::setClaimingOpen</code> function can be invoked multiple times at will, resetting the <code>claimDeadline</code> as well as the <code>endTime</code> incorrectly on each invocation and significantly affecting the point calculations.

Impact:

A re-invocation of EarlyAdopterPool::setClaimingOpen will cause points reported by

EarlyAdopterPool::claim to become inflated and causing the "total" points of the pool to also be miscalculated.

We advise the function to be invoke-able only once, ensuring that a non-zero _claimDeadline has also been specified during the call.

Alleviation:

The EtherFi team has opted not to remediate any finding in the EarlyAdopterPool implementation as they have deemed its on-chain data points unusable. As such, we consider this exhibit nullified given that it pertains a system component that will not be utilized in the EtherFi system.

EAP-04M: Potentially Redundant Amount Restriction

Туре	Severity	Location
Logical Fault	Minor	EarlyAdopterPool.sol:L112, L144

Description:

The EarlyAdopterPool::OnlyCorrectAmount modifier is meant to prevent deposits that do not fall within the 0.1 ether and 100 ether bounds, however, the check is applied to the per-deposit amount rather than the total amount of a user.

Impact:

The limitation of an EarlyAdopterPool::deposit call can be bypassed by performing multiple deposits thus defeating its purpose.

We advise the modifier's purpose to be revisited and it to potentially factor in the existing deposit of a user as it is possible to exceed the 100 ether mark by depositing the same asset multiple times with 100 ether per deposit.

Alleviation:

The EtherFi team has opted not to remediate any finding in the EarlyAdopterPool implementation as they have deemed its on-chain data points unusable. As such, we consider this exhibit nullified given that it pertains a system component that will not be utilized in the EtherFi system.

EAP-05M: Unfair Reset of Deposit Time

Туре	Severity	Location
Logical Fault	Minor	EarlyAdopterPool.sol:L124, L148

Description:

The EarlyAdopterPool::deposit and EarlyAdopterPool::depositEther functions will unfairly reset the global depositTime of the user that applies to all assets deposited to the pool.

Impact:

The system presently favours single-asset deposits over multi-asset deposits as the latter will suffer "loss-of-time" between deposits.

Example:

src/EarlyAdopterPool.sol

```
SOL
110 function deposit (address erc20Contract, uint256 amount)
       DepositingOpen
       whenNotPaused
       require(
            ( erc20Contract == rETH ||
               erc20Contract == sfrxETH ||
               erc20Contract == wstETH ||
               erc20Contract == cbETH),
       );
       depositInfo[msg.sender].depositTime = block.timestamp;
       depositInfo[msg.sender].totalERC20Balance += amount;
       userToErc20Balance[msg.sender][ erc20Contract] += amount;
       require(IERC20( erc20Contract).transferFrom(msg.sender, address(this), amount),
       emit DepositERC20(msg.sender, amount);
           rETHInstance.balanceOf(address(this)),
           wstETHInstance.balanceOf(address(this)),
           sfrxETHInstance.balanceOf(address(this)),
           cbETHInstance.balanceOf(address(this)),
           address (this) .balance,
           getContractTVL()
       );
141 function depositEther()
       OnlyCorrectAmount (msg.value)
       DepositingOpen
       whenNotPaused
       depositInfo[msg.sender].depositTime = block.timestamp;
       depositInfo[msg.sender].etherBalance += msg.value;
```

amit DanasitEth/mag candon mag value).

```
emit Depositeth (msg.sender, msg.value);

152 emit EthTVLUpdated (address (this).balance, getContractTVL());

153 }
```

We advise the code to either retain a depositTime per asset, or to expose a single function via which all relevant assets can be deposited in a single call. In the present implementation, if a user wishes to deposit multiple assets the time elapsed between each deposit will be "lost" as the depositTime will only be reset to the latest deposit's timestamp.

Alleviation:

The EtherFi team has opted not to remediate any finding in the EarlyAdopterPool implementation as they have deemed its on-chain data points unusable. As such, we consider this exhibit nullified given that it pertains a system component that will not be utilized in the EtherFi system.

EtherFiNode Manual Review Findings

EFN-01M: Inexistent Disable of Initializer

Туре	Severity	Location
Standard Conformity	Informational	EtherFiNode.sol:L25-L29

Description:

The **EtherFiNode::initialize** function is meant to be invoked once during the contract's lifetime, however, the base implementation of **EtherFiNode** does not initialize itself.

Impact:

While not an active threat in this particular instance, base implementations that may perform a delegatecall to an administrator-defined party can be compromised even if proxied. As such, it is best practice to always initialize base implementations of proxies automatically on deployment.

```
src/EtherFiNode.sol

SOL

25  function initialize(address _etherFiNodesManager) public {
26    require(stakingStartTimestamp == 0, "already initialised");
27    stakingStartTimestamp = uint32(block.timestamp);
28    etherFiNodesManager = _etherFiNodesManager;
29 }
```

We advise a constructor to be introduced to EtherFiNode that sets the stakingStartTimestamp to the maximum of uint256 (type (uint256) .max), disabling the base implementation of the EtherFiNode contract.

Alleviation:

The contract has had a **constructor** introduced that initializes the contract in the custom way we described in our recommendation, alleviating this exhibit in full.

EFN-02M: Incorrect Balance Assumption

Туре	Severity	Location
Logical Fault	Minor	EtherFiNode.sol:L333, L340-L341, L379

Description:

The **EtherFiNode::getFullWithdrawalPayouts** calculations assume that the balance in the function is at minimum equal to 16 ether due to the require check validating that the balance of the contract is greater-than-or-equal (>=) to 16 ether, however, the actual balance in use by the code is less than that as the vested auction rewards are subtracted if they cannot be claimed.

Impact:

As the vested auction fee is not accounted for in the calculations, the principal distribution may be performed with an incorrect assumption of at least 16 ether in the contract.

Example:

src/EtherFiNode.sol

```
SOL
       address(this).balance >= 16 ether,
335);
       phase == VALIDATOR PHASE.EXITED,
339);
       (vestedAuctionRewards - getClaimableVestedRewards());
344 uint256[] memory payouts = new uint256[](4);
348 if (balance > 32 ether) {
           payouts[0],
           payouts[1],
           payouts[2],
           payouts[3]
       ) = getRewardsPayouts(
           true,
           false,
           true,
           splits,
           scale,
           splits,
           scale
       );
       balance = 32 ether;
367 uint256 toBnftPrincipal;
368 uint256 toTnftPrincipal;
369 if (balance > 31.5 ether) {
       toBnftPrincipal = balance - 30 ether;
372 } else if (balance > 26 ether) {
       toBnftPrincipal = 1.5 ether;
375 } else if (balance > 25.5 ether) {
```

We advise the code to evaluate the actual balance as being greater-than-or-equal-to 16 ether. Alternatively, we advise the other balance-related findings of this audit report to be assimilated to the code rendering balance evaluations no longer necessary.

Alleviation:

The actual post-claimable reward balance is now utilized in the require check ensuring that at least 16 ether are present in the contract, alleviating this exhibit as a result.

EFN-03M: Inexistent Sanitization of Exit Timestamp

Туре	Severity	Location
Input Sanitization	Minor	EtherFiNode.sol:L62-L67

Description:

The **EtherFiNode::markExited** function does not sanitize the exitTimestamp the node allegedly exited at, permitting it to be misconfigured and cause the contract to underflow in certain operations and potentially lock funds.

Impact:

A misconfigured exit timestamp coupled with an exit request can cause the contract to underflow in EtherFiNode::getNonExitPenalty which is utilized when computing the rewards of a full withdrawal, causing the node's funds to be permanently locked within it.

```
src/EtherFiNode.sol

SOL

62  function markExited(
63    uint32 _exitTimestamp
64 ) external onlyEtherFiNodeManagerContract {
65    phase = VALIDATOR_PHASE.EXITED;
66    exitTimestamp = _exitTimestamp;
67 }
```

We advise the _exitTimestamp to be validated as greater-than the exitRequestTimestamp as well as less-than the current block.timestamp, indicating that the exit has already been performed and that it was properly performed after it was requested (if requested at all).

Alleviation:

The exit timestamp is properly sanitized as a time in the past, preventing the contract's exit from being misconfigured.

EFN-04M: Inexistent Caller Validation

Туре	Severity	Location
Logical Fault	Major	EtherFiNode.sol:L81-L85

Description:

The EtherFiNode::processVestedAuctionFeeWithdrawal function is meant to be invoked by the EtherFiNodesManager, however, the system does not validate its caller permitting the vestedAuctionRewards to be set to 0 even when they have not been distributed.

Impact:

It is presently possible to eliminate any vested auction rewards by invoking the

EtherFiNode::processVestedAuctionFeeWithdrawal function before the fee has been distributed via EtherFiNodesManager.

```
src/EtherFiNode.sol

SOL

81 function processVestedAuctionFeeWithdrawal() external {
82   if (_getClaimableVestedRewards() > 0) {
83      vestedAuctionRewards = 0;
84   }
85 }
```

We advise proper access control to be imposed on this function, ensuring that the vested auction rewards cannot be permanently locked in the EtherFinode instance.

Alleviation:

The **EtherFiNode::onlyEtherFiNodeManagerContract** modifier has been properly introduced to the referenced function, ensuring that it is solely called as part of the withdrawal processes in EtherFiNodesManager.

EFN-05M: Weak Validation of Node State

Туре	Severity	Location
Language Specific	Major	EtherFiNode.sol:L207, L333, L348

Description:

The 32 ether and 8 ether values are utilized throughout the EtherFi codebase to represent the base stake value of an ETH2.0 node and a number up to which ETH2.0 staking rewards can safely accumulate to prior to being withdrawn and distributed to the various users of an EtherFi node respectively.

As the system evaluates whether a node has "exited", has been "slashed", or has accrued normal staking rewards using a balance-based measurement, it is possible to influence a node's state via direct transfers. As an example, you can force a node to exit by directly transferring etherFiNode.balance - 8 ether to it, a significantly undesirable trait. Additionally, there is no inherent limitation to the staking rewards a node may acquire and as such, a node that has been inactive for a significant period of time can exceed this number.

Impact:

It is currently possible to "lock up" rewards of any node until it has been exited at a cost of etherFiNode.balance = 8 ether per node. This opens up an easy-to-access denial-of-service attack that renders all nodes of the EtherFi ecosystem susceptible to outside influence.

We advise the overall flow of EtherFi to be revised to instead rely on a consistent node state. To achieve this, an off-chain mechanism to inform the EtherFi ecosystem of operator slashes needs to be introduced, rendering the need for balance-based state deduction redundant.

Furthermore, calculations within the EtherFiNode implementation need to rely on both the measured balance of the node as well as the node's state. In order to ensure that they cannot be manipulated between the time window of a node being slashed and its slash being reflected on-chain, a distribution request should be throttled via the EtherFiNodeManager using a time threshold in which the EtherFi team is expected to report the node's slash state on-chain.

Alleviation:

The EtherFi team has evaluated this exhibit and has stated that a node operator would be willing to exit to acquire the "donated" ETH. The vulnerability describes that this can be used to reduce the EtherFi network's nodes and this has been accepted by the EtherFi team as an intended function. Due to this, we consider the exhibit as acknowledged.

EtherFiNodesManager Manual Review Findings

EFM-01M: Inexistent Disable of Initializer

Туре	Severity	Location
Standard Conformity	Informational	EtherFiNodesManager.sol:L63-L119

Description:

The EtherFiNodesManager contract is meant to be an upgradeable contract that is initialized via the EtherFiNodesManager::initialize function, however, the base implementation of EtherFiNodesManager is not disabling the initializer during its construction.

Impact:

While not an active threat in this particular instance, base implementations that may perform a delegatecall to an administrator-defined party can be compromised even if proxied. As such, it is best practice to always initialize base implementations of proxies automatically on deployment.

```
src/EtherFiNodesManager.sol

SOL

63  function initialize(
64   address _treasuryContract,
65   address _auctionContract,
66   address _stakingManagerContract,
67   address _tnftContract,
68   address _bnftContract,
69   address _protocolRevenueManagerContract
70 ) external initializer {
```

We advise a constructor to be introduced to EtherFiNodesManager that executes Initializable: _disableInitializers, ensuring that the base implementation of EtherFiNodesManager cannot be initialized maliciously.

Alleviation:

A constructor was introduced that properly disables the contract's initializers via the Initializable::_disableInitializers function, disallowing the contract from being initialized at its logic contract location.

EFM-02M: Inexistent Prevention of Duplicate Exit

Туре	Severity	Location
Input Sanitization	Minor	EtherFiNodesManager.sol:L507, L510-L511

Description:

The **EtherFiNodesManager::processNodeExit** function will not evaluate that the exit being processed is valid, permitting the same node to be exited twice. In such a case, the node's exit timestamp can be overridden affecting its penalty calculations.

Impact:

As the _exitTimestamp will be arbitrarily resettable in the etherFiNode, the penalty it actually applies in its EtherFiNode::getNonExitPenalty can be influenced by resetting it as "exited".

Example:

src/Ether Fi Nodes Manager.sol

```
SOL
       uint256 validatorId,
      uint32 exitTimestamp
       address etherfiNode = etherfiNodeAddress[ validatorId];
       IEtherFiNode(etherfiNode).markExited(exitTimestamp);
       uint256 amount = protocolRevenueManagerInstance
           .distributeAuctionRevenue( validatorId);
       IEtherFiNode(etherfiNode).setLocalRevenueIndex(0);
          uint256 toOperator,
          uint256 toTnft,
           uint256 toBnft,
           uint256 toTreasury
       ) = IEtherFiNode(etherfiNode).calculatePayouts(
               amount,
               protocolRewardsSplit,
               SCALE
           );
       address operator = auctionInterfaceInstance.getBidOwner( validatorId);
       address tnftHolder = tnftInstance.ownerOf( validatorId);
       address bnftHolder = bnftInstance.ownerOf( validatorId);
       numberOfValidators -= 1;
       IEtherFiNode(etherfiNode).withdrawFunds(
           treasuryContract,
           toTreasury,
           operator,
           toOperator,
           tnftHolder,
           toTnft,
          bnftHolder,
           toBnft
       );
```

545 emit NodeExitProcessed (_validatorid);

546

We advise the code to ensure that the etherFiNode is not in an exitted phase already, preventing the exit timestamp of a node from being re-set.

Alleviation:

The EtherFi team has stated that they wish to retain the capability of overwriting the exit timestamp of a node to ensure mistakes can be corrected. As such, we consider this exhibit nullified **based on the fact that it represents desirable behaviour by the EtherFi team**.

EFM-03M: Inexistent Sanitization of Non-Exit Penalty Rate

Туре	Severity	Location
Input Sanitization	Minor	EtherFiNodesManager.sol:L439-L445

Description:

The EtherFiNodesManager::setNonExitPenaltyDailyRate function is meant to allow the nonExitPenaltyDailyRate value to be updated, however, no sanitization is performed on the new nonExitPenaltyDailyRate value.

Impact:

A misconfiguration of this variable will cause arithmetic underflows in each EtherFinode instance thus rendering the system's non-exit penalty inoperable.

```
src/EtherFiNodesManager.sol

SOL

439 /// @notice Sets the Non Exit Penalty Daily Rate amount
440 /// @param _nonExitPenaltyDailyRate the new non exit daily rate
441 function setNonExitPenaltyDailyRate(
442    uint64 _nonExitPenaltyDailyRate
443 ) public onlyOwner {
444    nonExitPenaltyDailyRate = _nonExitPenaltyDailyRate;
445 }
```

We advise it to be mandated as less-than-or-equal-to 100 as otherwise calculations within EtherFiNode will fail to execute properly.

Alleviation:

A require check was properly introduced ensuring that the non-exit penalty daily rate is at most equal to 100.

EFM-04M: Inexistent Validation of Node State

Туре	Severity	Location
Logical Fault	Major	EtherFiNodesManager.sol:L191-L196, L260-L266

Description:

The EtherFiNodesManager::partialWithdraw and

EtherFiNodesManager::partialWithdrawBatchGroupByOperator functions will perm

EtherFiNodesManager::partialWithdrawBatchGroupByOperator functions will permit a node to be partially withdrawn even after it has been marked as EXITED. In such a case, a significant vulnerability arises whereby a user performs a partial withdrawal of an exited node and sets the protocolRewards flag to true.

The code of both functions will invoke the **ProtocolRevenueManager::distributeAuctionRevenue** function which will distribute a value of 0 while setting the node's **localRevenueIndex** to the latest globalRevenueIndex. As such, the node will begin accruing auction rewards when it is not part of the network.

Impact:

Auction fee funds can be siphoned out from the system improperly by inactive validators as the partial withdrawal mechanisms inadequately validate the node's current phase.

Example:

src/EtherFiNodesManager.sol

```
SOL
191 function partialWithdraw(
       uint256 validatorId,
      bool stakingRewards,
       bool protocolRewards,
     bool vestedAuctionFee
196 ) public nonReentrant {
       address etherfiNode = etherfiNodeAddress[ validatorId];
       uint256 balance = address(etherfiNode).balance;
           balance < 8 ether,</pre>
       );
           uint256 toOperator,
           uint256 toTnft,
           uint256 toBnft,
           uint256 toTreasury
       ) = getRewardsPayouts(
               validatorId,
               stakingRewards,
               protocolRewards,
               vestedAuctionFee
       if ( protocolRewards) {
           protocolRevenueManagerInstance.distributeAuctionRevenue(
               validatorId
       if ( vestedAuctionFee) {
           IEtherFiNode(etherfiNode).processVestedAuctionFeeWithdrawal();
       address operator = auctionInterfaceInstance.getBidOwner( validatorId);
       address tnftHolder = tnftInstance.ownerOf( validatorId);
       address bnftHolder = bnftInstance.ownerOf( validatorId);
       IEtherFiNode(etherfiNode).withdrawFunds(
           treasuryContract,
           toTreasury,
           operator,
           toOperator,
```

```
235 toTnft,
236 bnftHolder,
237 toBnft
238 );
239 }
```

We advise the partial withdrawal code to validate the phase of an oracle that is being attempted to be withdrawn from. If the phase is <code>EXITED</code>, the <code>_stakingRewards</code> and <code>_protocolRewards</code> flags should be set to <code>false</code> thus ensuring that only the <code>_vestedAuctionFee</code> flag can be <code>true</code> as the auction fee may vest after a protocol has been exited and its full withdrawal has been performed.

As an additional point, these mechanisms should also ensure that a node is in either a LIVE or EXITED state as otherwise withdrawals should not be possible.

Alleviation:

The code of ProtocolRevenueManager was updated to ensure that if a particular EtherFi node has been marked as exited it is not to be distributed auction revenue rewards. As such, the described vulnerability is not possible. As such, we consider this exhibit alleviated.

EFM-05M: Weak Validation of Node State

Туре	Severity	Location
Language Specific	Major	EtherFiNodesManager.sol:L200, L282, L342

Description:

The 16 ether and 8 ether values are utilized throughout the EtherFi codebase to represent the base stake value of an ETH2.0 node and a number up to which ETH2.0 staking rewards can safely accumulate to prior to being withdrawn and distributed to the various users of an EtherFi node respectively.

As the system evaluates whether a node has "exited", has been "slashed", or has accrued normal staking rewards using a balance-based measurement, it is possible to influence a node's state via direct transfers. As an example, you can force a node to exit by directly transferring etherFiNode.balance - 8 ether to it, a significantly undesirable trait. Additionally, there is no inherent limitation to the staking rewards a node may acquire and as such, a node that has been inactive for a significant period of time can exceed this number.

Impact:

It is currently possible to "lock up" rewards of any node until it has been exited at a cost of etherFiNode.balance = 8 ether per node. This opens up an easy-to-access denial-of-service attack that renders all nodes of the EtherFi ecosystem susceptible to outside influence.

```
src/EtherFiNodesManager.sol

SOL

199 require(
200  balance < 8 ether,
201  "etherfi node contract's balance is above 8 ETH. You should exit the node."
202 );</pre>
```

We advise the overall flow of EtherFi to be revised to instead rely on a consistent node state. To achieve this, an off-chain mechanism to inform the EtherFi ecosystem of operator slashes needs to be introduced, rendering the need for balance-based state deduction redundant.

Furthermore, calculations within the EtherFiNode implementation need to rely on both the measured balance of the node as well as the node's state. In order to ensure that they cannot be manipulated between the time window of a node being slashed and its slash being reflected on-chain, a distribution request should be throttled via the EtherFiNodeManager using a time threshold in which the EtherFi team is expected to report the node's slash state on-chain.

Alleviation:

The EtherFi team has evaluated this exhibit and has stated that a node operator would be willing to exit to acquire the "donated" ETH. The vulnerability describes that this can be used to reduce the EtherFi network's nodes and this has been accepted by the EtherFi team as an intended function. Due to this, we consider the exhibit as acknowledged.

NodeOperatorManager Manual Review Findings

NOM-01M: Inexplicable Capability of Re-Invocation

Туре	Severity	Location
Centralization Concern	Unknown	NodeOperatorManager.sol:L124-L131

Description:

The NodeOperatorManager::setAuctionContractAddress permits the auctionManagerContractAddress entry to be configured due to circular dependencies, however, it can be invoked an arbitrary number of times.

```
src/NodeOperatorManager.sol

SOL

124 /// @notice Sets the auction contract address for verification purposes
125 /// @dev Set manually due to circular dependencies
126 /// @param _auctionContractAddress address of the deployed auction contract address
127 function setAuctionContractAddress(
128    address _auctionContractAddress
129 ) public onlyOwner {
130    auctionManagerContractAddress = _auctionContractAddress;
131 }
```

Given that the AuctionManager implementation represents an upgradeable contract, we advise the code to allow setting the auctionManagerContractAddress only once thus ensuring that the contract's operation cannot be compromised via privilege misuse.

Alleviation:

The referenced function has had a require check introduced to ensure it cannot be re-invoked beyond its initialization, alleviating this exhibit's concerns fully.

NOM-02M: Incorrect Verification of Whitelist

Туре	Severity	Location
Logical Fault	Major	NodeOperatorManager.sol:L50, L137-L149

Description:

The NodeOperatorManager::_verifyWhitelistedAddress function invoked during a NodeOperatorManager::registerNodeOperator invocation is unrestrictive, permitting the transaction to succeed and a user to register as a node operator even if they are not part of the whitelist.

Impact:

In the current implementation, any user can register as a node operator with valid KeyData regardless of whether they have been explicitly authorized.

Even if the system's design is to allow a user to register as a node operator without being present in the whitelist, the current code is incorrect as the user would have no way to re-enter the whitelist after they have registered due to the require check at the top of the function. As such, the current behaviour is incorrect regardless of the system's intended design.

We advise the <code>NodeOperatorManager::_verifyWhitelistedAddress</code> code to be updated, evaluating the whitelisted status in a <code>require</code> check instead.

Alleviation:

The EtherFi team has stated that this is intended behaviour and that they do not intend to allow users to whitelist after they have been registered. As such, we consider this exhibit nullified **as it outlines desirable behaviour by the EtherFi team**.

ProtocolRevenueManager Manual Review Findings

PRM-01M: Inexplicable Capability of Re-Invocation

Туре	Severity	Location
Centralization Concern	Unknown	ProtocolRevenueManager.sol:L121-L125, L130-L134

Description:

The ProtocolRevenueManager::setEtherFiNodesManagerAddress & ProtocolRevenueManager::setAuctionManagerAddress permit the etherFiNodesManager & auctionManager variables respectively to be set after the contract's initialization due to circular dependencies, however, each function can be invoked an arbitrary number of times.

```
src/ProtocolRevenueManager.sol

sol

118 /// @notice Instantiates the interface of the node manager for integration
119 /// @dev Set manually due to cirular dependencies
120 /// @param _etherFiNodesManager etherfi node manager address to set
121 function setEtherFiNodesManagerAddress(
122    address _etherFiNodesManager
123 ) external onlyOwner {
124    etherFiNodesManager = IEtherFiNodesManager(_etherFiNodesManager);
125 }
126
127 /// @notice Instantiates the interface of the auction manager for integration
128 /// @dev Set manually due to cirular dependencies
129 /// @param _auctionManager auction manager address to set
130 function setAuctionManagerAddress(
131    address _auctionManager
132 ) external onlyOwner {
133    auctionManager = IAuctionManager(_auctionManager);
134 }
```

As both the EtherFiNodesManager & AuctionManager contracts represent an upgradeable module, we advise the referenced functions to be invoke-able only once.

Alleviation:

All referenced functions have had require checks introduced that ensure they cannot be re-invoked beyond their initialization, alleviating this exhibit's concerns fully.

PRM-02M: Inexistent Disable of Initializer

Туре	Severity	Location
Standard Conformity	Informational	ProtocolRevenueManager.sol:L39-L48

Description:

The ProtocolRevenueManager contract is meant to be an upgradeable contract that is initialized via the ProtocolRevenueManager::initialize function, however, the base implementation of ProtocolRevenueManager is not disabling the initializer during its construction.

Impact:

While not an active threat in this particular instance, base implementations that may perform a delegatecall to an administrator-defined party can be compromised even if proxied. As such, it is best practice to always initialize base implementations of proxies automatically on deployment.

Example:

src/ProtocolRevenueManager.sol

```
SOL
  contract ProtocolRevenueManager is
      IProtocolRevenueManager,
       PausableUpgradeable,
     OwnableUpgradeable,
      ReentrancyGuardUpgradeable,
      UUPSUpgradeable
       IEtherFiNodesManager public etherFiNodesManager;
       IAuctionManager public auctionManager;
      uint256 public globalRevenueIndex;
       uint128 public vestedAuctionFeeSplitForStakers;
       uint128 public auctionFeeVestingPeriodForStakersInDays;
       uint256[32] gap;
```

We advise a constructor to be introduced to ProtocolRevenueManager that executes Initializable: _disableInitializers, ensuring that the base implementation of ProtocolRevenueManager cannot be initialized maliciously.

Alleviation:

A constructor was introduced that properly disables the contract's initializers via the Initializable::_disableInitializers function, disallowing the contract from being initialized at its logic contract location.

PRM-03M: Inexistent Sanitization of Fee Proportion

Туре	Severity	Location
Input Sanitization	Minor	ProtocolRevenueManager.sol:L149

Description:

The ProtocolRevenueManager::setAuctionRewardSplitForStakers function does not sanitize its input argument, permitting an un-serviceable fee split to be set.

Impact:

A misconfigured fee split will cause the local and global revenue indexes of the EtherFi protocol to be misconfigured, greatly affecting the system's reward accounting.

```
src/ProtocolRevenueManager.sol

SOL

144 /// @notice set the auction reward split for stakers
145 /// @param _split vesting period in days
146 function setAuctionRewardSplitForStakers(
147    uint128 _split
148 ) external onlyOwner {
149    vestedAuctionFeeSplitForStakers = _split;
150 }
```

We advise the code to ensure that the input split is at most equal to 100, the maximum accuracy supported by ProtocolRevenueManager::addAuctionRevenue.

Alleviation:

The auction reward split is now properly sanitized as being at most 100, alleviating this exhibit in full.

ScoreManager Manual Review Findings

SMR-01M: Inexistent Disable of Initializer

Туре	Severity	Location
Standard Conformity	Informational	ScoreManager.sol:L46-L54

Description:

The ScoreManager contract is meant to be an upgradeable contract that is initialized via the ScoreManager::initialize function, however, the base implementation of ScoreManager is not disabling the initializer during its construction.

Impact:

While not an active threat in this particular instance, base implementations that may perform a delegatecall to an administrator-defined party can be compromised even if proxied. As such, it is best practice to always initialize base implementations of proxies automatically on deployment.

Example:

src/ScoreManager.sol

```
SOL
12 contract ScoreManager is
     IScoreManager,
       OwnableUpgradeable,
       PausableUpgradeable,
       ReentrancyGuardUpgradeable,
       UUPSUpgradeable
       uint32 public numberOfTypes;
       mapping(uint256 => mapping(address => bytes32)) public scores;
       mapping(uint256 => bytes32) public totalScores;
       mapping(address => bool) public allowedCallers;
       mapping(uint256 => bytes) public scoreTypes;
       mapping(bytes => uint256) public typeIds;
       uint256[32] gap;
       event ScoreSet(address indexed user, uint256 score typeID, bytes32 data);
       event NewTypeAdded(uint256 Id, bytes ScoreType);
       function initialize() external initializer {
            Pausable init();
           UUPSUpgradeable init();
           ReentrancyGuard init();
```

We advise a constructor to be introduced to ScoreManager that executes

Initializable::_disableInitializers, ensuring that the base implementation of ScoreManager cannot be initialized maliciously.

Alleviation:

A constructor was introduced that properly disables the contract's initializers via the Initializable::_disableInitializers function, disallowing the contract from being initialized at its logic contract location.

SMR-02M: Inexplicable Data Types

Туре	Severity	Location
Language Specific	Informational	ScoreManager.sol:L25, L28

Description:

The ScoreManager contract utilizes a bytes32 variable for maintaining the scores of a particular type ID and the users within it, however, contracts such as ClaimReceiverPool and LiquidityPool all cast the bytes32 values to uint256 values prior to use.

```
src/ScoreManager.sol

SOL

22  // bytes: indicate the type of the score (like the name of the promotion)
23  // address: user wallet address
24  // bytes32: a byte stream of user score + etc

25  mapping(uint256 => mapping(address => bytes32)) public scores;
26

27  // bytes32: a byte stream of aggregated info of users' scores (e.g., total sum)
28  mapping(uint256 => bytes32) public totalScores;
```

We advise the data types of ScoreManager to be converted to uint256, optimizing and simplifying the code of the overall EtherFi project significantly as complex type casts from and to bytes32 values would no longer be necessary.

Alleviation:

The uint256 data types are no utilized for both mapping declarations as well as throughout the contract's codebase, optimizing it significantly.

SMR-03M: Inexistent Sanitization of Valid Type

Туре	Severity	Location
Input Sanitization	Minor	ScoreManager.sol:L62, L74

Description:

The ScoreManager::setScore and ScoreManager::setTotalScore functions do not validate that the supplied typeId is valid.

Impact:

It is possible to alter scores for a type ID that has not yet been included to the ScoreManager.

Example:

src/ScoreManager.sol

```
SOL
  function setScore(
      uint256 typeId,
      address user,
      bytes32 score
   ) external allowedCaller(msg.sender) nonZeroAddress( user) {
       scores[ typeId][ user] = score;
      emit ScoreSet( user, typeId, score);
   function setTotalScore(
      uint256 typeId,
      bytes32 totalScore
   ) external allowedCaller(msg.sender) {
       totalScores[typeId] = totalScore;
  function setCallerStatus(address caller, bool flag) external onlyOwner nonZeroAddres
       allowedCallers[ caller] = flag;
   function addNewScoreType(bytes memory type) external onlyOwner returns (uint256) {
      scoreTypes[numberOfTypes] = type;
       typeIds[ type] = numberOfTypes;
       emit NewTypeAdded(numberOfTypes, type);
       numberOfTypes++;
       return numberOfTypes - 1;
```

We advise a require check to be introduced ensuring that the provided type ID is less-than the value of numberOfTypes.

Alleviation:

The _typeId supplied as input to a **ScoreManager::setScore** call is now properly sanitized as being in existence, alleviating this exhibit in full as the **ScoreManager::setTotalScore** function is no longer present.

SMR-04M: Improper Score Maintenance Mechanisms

Туре	Severity	Location
Language Specific	Medium	ScoreManager.sol:L66, L77

Description:

The ScoreManager contract is meant to maintain a list of user scores as well as their sum for a particular typeId, however, the maintenance of the score list's validity is performed entirely manually.

As multiple transactions are required to maintain each score type's validity, a race-condition manifests whereby users can exploit an incorrect ScoreManager state between adjustment transactions.

Impact:

As the total score and a user's score would be adjusted in separate transactions, a race condition manifests during the time window between those two invocations that a user can exploit while possessing an "unfair" proportion of the total score.

Example:

src/ScoreManager.sol

```
SOL
61 function setScore(
      uint256 typeId,
       address user,
65 ) external allowedCaller(msg.sender) nonZeroAddress( user) {
       scores[ typeId][ user] = score;
      emit ScoreSet( user, typeId, score);
   function setTotalScore(
      uint256 typeId,
       bytes32 totalScore
76 ) external allowedCaller(msg.sender) {
       totalScores[typeId] = totalScore;
```

We advise the code to expose functions that increment or decrement a user's score and in such a case to also increment or decrement the total score of the typeId respectively, ensuring that the score list of ScoreManager is managed automatically.

Alleviation:

The code now properly maintains the total score of a _typeId whenever an individual's score is set with the ScoreManager::setTotalScore function removed, alleviating this exhibit in full.

StakingManager Manual Review Findings

SME-01M: Inexplicable Capability of Re-Invocation

Туре	Severity	Location
Centralization Concern	Unknown	StakingManager.sol:L223-L229, L239-L244, L246-L248, L250- L252

Description:

The referenced functions permit sensitive configurational variables of the contract to be set at will.

```
src/StakingManager.sol

SOL

223 function setEtherFiNodesManagerAddress(
224    address _nodesManagerAddress
225 ) public onlyOwner {
226    nodesManagerIntefaceInstance = IEtherFiNodesManager(
227    _nodesManagerAddress
228   );
229 }
```

Given that these contracts represent either upgradeable implementations or implementations meant to remain the same throughout the StakingManager contract's lifetime, we advise the functions to be invokeable only once by evaluating whether the variable they adjust has already been set to a non-zero entry.

Alleviation:

All referenced functions have had require checks introduced that ensure they cannot be re-invoked beyond their initialization, alleviating this exhibit's concerns fully.

SME-02M: Inexistent Disable of Initializer

Туре	Severity	Location
Standard Conformity	Informational	Staking Manager. sol: L76-L90

Description:

The StakingManager contract is meant to be an upgradeable contract that is initialized via the StakingManager::initialize function, however, the base implementation of StakingManager is not disabling the initializer during its construction.

Impact:

While not an active threat in this particular instance, base implementations that may perform a delegatecall to an administrator-defined party can be compromised even if proxied. As such, it is best practice to always initialize base implementations of proxies automatically on deployment.

We advise a constructor to be introduced to StakingManager that executes

Initializable::_disableInitializers, ensuring that the base implementation of StakingManager cannot be initialized maliciously.

Alleviation:

A constructor was introduced that properly disables the contract's initializers via the Initializable::_disableInitializers function, disallowing the contract from being initialized at its logic contract location.

SME-03M: Incorrect Data Entry

Туре	Severity	Location
Logical Fault	Major	StakingManager.sol:L291, L339

Description:

Based on the execution flow of AuctionManager and StakingManager, the node operator that will "register" a validator must be the initial bid creator in the AuctionManager that a "financer" has submitted the 32 ether required to run the node. As such, the StakingManager::_processDeposit function must assign the AuctionManager::getBidOwner of the processed bid rather than the msg.sender.

Impact:

The "auction" system is presently not operating as the same user who submits the 32 ether for a node is intended to run it, simply acquiring the fee of an auction arbitrarily at no benefit of the bid's creator.

We advise the referenced assignment to be updated accordingly, ensuring a correct execution and "auction" style flow in the EtherFi codebase.

To note, the way the NFTs of the node's creation are distributed will also need to be governed in trustless manner by the contract's code rather than being specified by the node operator to ensure the original bidder also obtains fund-related rights over the node that is created.

Alleviation:

The EtherFi team has stated the current flow of execution in the contract is correct and our assumption in relation to the matching mechanism is incorrect. As such, we consider this exhibit nullified as **the code satisfies EtherFi's business requirements** in its current state.

SME-04M: ETH2.0 Validator Front-Run Withdrawal Credential Attack

Туре	Severity	Location
Logical Fault	Major	Staking Manager.sol:L299-L304

Description:

The ETH2.0 node deposit mechanism of StakingManager is insecure as it suffers from an inherent flaw in Ethereum's DepositContract. In detail, multiple deposits for the same publickey can be performed with the validator being activated solely when all deposits sum to 32 ether.

A caveat of this system is that the <code>ETH2.0</code> system will honour the withdrawal credentials that were specified in the first <code>DepositContract::deposit</code> transaction, not necessarily the ones specified in <code>StakingManager::_registerValidator</code>. As such, it is possible for all <code>ETH2.0</code> fund related operations (exits, rewards, etc.) to be redirected to a different address unrelated to the EtherFi protocol. For more information, consult <code>RocketPool's Withdrawal Credential Exploit Analysis</code>.

Impact:

It is presently possible to activate an EtherFi node without necessarily setting it as the intended recipient of an ETH2.0 node's withdrawal, undermining the EtherFi system as a whole.

Example:

src/StakingManager.sol

```
SOL
```

```
280 function registerValidator(
       uint256 validatorId,
      address bNftRecipient,
       address tNftRecipient,
       DepositData calldata depositData
       require(
           nodesManagerIntefaceInstance.phase( validatorId) ==
               IEtherFiNode.VALIDATOR PHASE.STAKE DEPOSITED,
       );
       require(bidIdToStaker[ validatorId] == msg.sender, "Not deposit owner");
       address staker = bidIdToStaker[ validatorId];
       bytes memory withdrawalCredentials = nodesManagerIntefaceInstance
            .getWithdrawalCredentials( validatorId);
       depositContractEth2.deposit{value: stakeAmount}(
           depositData.publicKey,
           withdrawalCredentials,
           depositData.signature,
           depositData.depositDataRoot
       );
       nodesManagerIntefaceInstance.incrementNumberOfValidators(1);
       nodesManagerIntefaceInstance.setEtherFiNodePhase(
           validatorId,
           IEtherFiNode.VALIDATOR PHASE.LIVE
       );
       nodesManagerIntefaceInstance
           .setEtherFiNodeIpfsHashForEncryptedValidatorKey(
               validatorId,
               depositData.ipfsHashForEncryptedValidatorKey
           );
```

```
// Mint (T, B)-NFTs to the Staker
uint256 nftTokenId = _validatorId;
TNFTInterfaceInstance.mint(_tNftRecipient, nftTokenId);
BNFTInterfaceInstance.mint(_bNftRecipient, nftTokenId);

auctionInterfaceInstance.processAuctionFeeTransfer(_validatorId);

emit ValidatorRegistered(
    auctionInterfaceInstance.getBidOwner(_validatorId),
    _bNftRecipient,
    _tNftRecipient,
    _validatorId,
    _depositData.publicKey,
    _depositData.ipfsHashForEncryptedValidatorKey

);

334 }
```

We advise the validator registration mechanism to be revised, performing the deposit to the Beacon chain but not affecting the EtherFi system (i.e. not incrementing the number of validators, not affecting the node phase etc.). Afterwards, an entity (such as a DAO or the EtherFi team) that can process off-chain knowledge will need to validate that the beacon chain registration has been performed with the correct withdrawal credentials and submit a transaction to a new function in StakingManager that will "activate" the EtherFi node by setting it to LIVE, adjusting the number of validators, minting the relevant NFTs, transferring the auction fee, and setting the IPFS hash of the encrypted validator key.

Alleviation:

The EtherFi team has stated that they identified this flaw during the audit process, however, the code appears to not apply a solution for it. Additionally, no issue was present in the GitHub repository that outlines it. As such, we consider this exhibit not alleviated.

TNFT Manual Review Findings

TNF-01M: Inexistent Disable of Initializer

Туре	Severity	Location
Standard Conformity	Informational	TNFT.sol:L19-L25

Description:

The TNFT contract is meant to be an upgradeable contract that is initialized via the TNFT::initialize function, however, the base implementation of TNFT is not disabling the initializer during its construction.

Impact:

While not an active threat in this particular instance, base implementations that may perform a delegatecall to an administrator-defined party can be compromised even if proxied. As such, it is best practice to always initialize base implementations of proxies automatically on deployment.

Example:

src/TNFT.sol

We advise a constructor to be introduced to TNFT that executes

Initializable::_disableInitializers, ensuring that the base implementation of TNFT cannot be initialized maliciously.

Alleviation:

A constructor was introduced that properly disables the contract's initializers via the Initializable::_disableInitializers function, disallowing the contract from being initialized at its logic contract location.

AuctionManager Code Style Findings

AMR-01C: Inefficient Optimization of Iterator Increment

Туре	Severity	Location
Gas Optimization	Informational	AuctionManager.sol:L126, L245-L249

Description:

The referenced optimization of the iterator's increment statement is ineffective as a private function is invoked that contains significant overhead.

```
src/AuctionManager.sol

SOL

126 for (uint256 i = 0; i < _bidSize; i = uncheckedInc(i)) {</pre>
```

We advise the code to instead optimize the iterator's increment by omitting it from the for declaration and relocating it at the end of the for loop's body, wrapping the increment statement (++i) in an unchecked code block.

Alleviation:

While the inefficient uncheckedInc invocation was omitted, the code still inefficiently increments the iterator by performing a simple i++ operation. We advise the operation to be relocated to the end of the for loop in an unchecked code block and to additionally perform a pre-fix increment operation (++i) as its more optimal than a post-fix increment operation (i++).

AMR-02C: Inefficient mapping Lookups

Туре	Severity	Location
Gas Optimization	Informational	AuctionManager.sol:L167, L168, L171, L174, L191, L193, L202, L204

Description:

The linked statements perform key-based lookup operations on mapping declarations from storage multiple times for the same key redundantly.

```
src/AuctionManager.sol

sol

166 function cancelBid(uint256 _bidId) public whenNotPaused {
167    require(bids[_bidId].bidderAddress == msg.sender, "Invalid bid");
168    require(bids[_bidId].isActive == true, "Bid already cancelled");
169
170    // Cancel the bid by de-activating it
171    bids[_bidId].isActive = false;
172
173    // Get the value of the cancelled bid to refund
174    uint256 bidValue = bids[_bidId].amount;
175
176    // Refund the user with their bid amount
177    (bool sent, ) = msg.sender.call{value: bidValue}("");
178    require(sent, "Failed to send Ether");
179
180    numberOfActiveBids--;
181
182    emit BidCancelled(_bidId);
183 }
```

As the lookups internally perform an expensive keccak256 operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the mapping in case of primitive types or holds a storage pointer to the struct contained.

Alleviation:

While the bid cancellation mechanism optimized its mapping lookups, the

AuctionManager::updateSelectedBidInformation and AuctionManager::reEnterAuction Code segments were not updated.

AMR-03C: Loop Iterator Optimization

Туре	Severity	Location
Gas Optimization	Informational	AuctionManager.sol:L158

Description:

The linked for loop increments / decrements the iterator "safely" due to Solidity's built-in safe arithmetics (post-0.8.x).

```
src/AuctionManager.sol

SOL

158 for (uint256 i = 0; i < _bidIds.length; i++) {</pre>
```

We advise the increment / decrement operation to be performed in an unchecked code block as the last statement within the for loop to optimize its execution cost.

Alleviation:

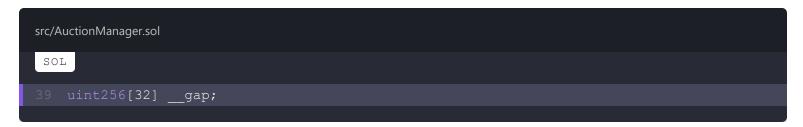
The EtherFi team examined this exhibit and opted not to apply a remediation in the current iteration of the codebase, instead acknowledging it.

AMR-04C: Non-Standard Gap Size

Туре	Severity	Location
Standard Conformity	Informational	AuctionManager.sol:L39

Description:

The referenced ___gap variable is meant to replicate OpenZeppelin's upgradeability standard by declaring an offset of variables that can be declared at a later point on the same contract without affecting the order of variables in storage in the overall contract.



The methodology employed for calculating the appropriate length for the variable in OpenZeppelin is to utilize the value of 50 as a base and subtract the number of 32-byte slots that are already occupied by the contract (i.e. 1 in the case of TNFT). We advise the size of this variable to be corrected by applying the same methodology and ensuring the length of the gap is directly correlated to the storage layout of the contract it resides in.

Alleviation:

The gap array's length has been adjusted to a standardized value as advised.

AMR-05C: Redundant Duplicate Application of Access Control

Туре	Severity	Location
Gas Optimization	Informational	AuctionManager.sol:L157, L166

Description:

The top-level AuctionManager::cancelBidBatch function will apply the

PausableUpgradeable::whenNotPaused modifier and will invoke the AuctionManager::cancelBid function that also applies the same modifier.

```
src/AuctionManager.sol

SOL

157 function cancelBidBatch(uint256[] calldata _bidIds) external whenNotPaused {
158    for (uint256 i = 0; i < _bidIds.length; i++) {
159         cancelBid(_bidIds[i]);
160    }
161 }
162
163 /// @notice Cancels a specified bid by de-activating it
164 /// @dev Require the bid to exist and be active
165 /// @param _bidId the ID of the bid to cancel
166 function cancelBid(uint256 _bidId) public whenNotPaused {</pre>
```

We advise the PausableUpgradeable::whenNotPaused modifier to be omitted from the top-level

AuctionManager::cancelBidBatch function, optimizing the code's gas cost.

As an alternative optimization, the code of AuctionManager::cancelBid can be relocated to an internal underscore-prefixed (_) function that is invoked by both AuctionManager::cancelBid and AuctionManager::cancelBidBatch while retaining the PausableUpgradeable::whenNotPaused modifier in AuctionManager::cancelBidBatch, ensuring that the batch cancellation operation applies the PausableUpgradeable::whenNotPaused modifier only once during its execution.

Alleviation:

The code of AuctionManager::cancelBid was relocated to an AuctionManager::_cancelBid internal function that both the AuctionManager::cancelBid and AuctionManager::cancelBid functions invoke, optimizing the codebase as advised.

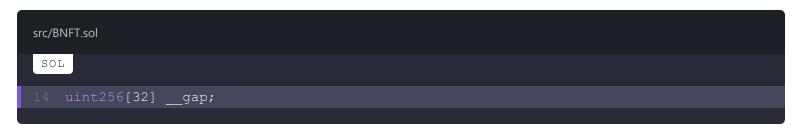
BNFT Code Style Findings

BNF-01C: Non-Standard Gap Size

Туре	Severity	Location
Standard Conformity	Informational	BNFT.sol:L14

Description:

The referenced ___gap variable is meant to replicate OpenZeppelin's upgradeability standard by declaring an offset of variables that can be declared at a later point on the same contract without affecting the order of variables in storage in the overall contract.



The methodology employed for calculating the appropriate length for the variable in OpenZeppelin is to utilize the value of 50 as a base and subtract the number of 32-byte slots that are already occupied by the contract (i.e. 1 in the case of TNFT). We advise the size of this variable to be corrected by applying the same methodology and ensuring the length of the gap is directly correlated to the storage layout of the contract it resides in.

Alleviation:

The gap array's length has been adjusted to a standardized value as advised.

ClaimReceiverPool Code Style Findings

CRP-01C: Duplicate Invocation of Getter

Туре	Severity	Location
Gas Optimization	Informational	ClaimReceiverPool.sol:L126, L139

Description:

The referenced declarations are assigned to the same evaluation in two separate variables.

We advise the same variable to be utilized and the second declaration to be omitted entirely, optimizing the code.

Alleviation:

The score system of the deposit flow in the contract has been refactored rendering this exhibit no longer applicable.

CRP-02C: Inexistent Gap Declaration

Туре	Severity	Location
Standard Conformity	Informational	ClaimReceiverPool.sol:L18

Description:

The ClaimReceiverPool contract does not have any gap variable declared.

```
src/ClaimReceiverPool.sol

SOL

18 contract ClaimReceiverPool is

19 Initializable,
20 PausableUpgradeable,
21 OwnableUpgradeable,
22 ReentrancyGuardUpgradeable,
23 UUPSUpgradeable
24 {
```

We advise one to be introduced akin to the rest of the codebase.

Alleviation:

While a gap has been introduced to the codebase, it has been introduced in between variable declarations rather than at the end. We strongly advise its declaration to be relocated to the end of the contract, permitting upgrade-able variable extensibility in a standardized way.

EarlyAdopterPool Code Style Findings

EAP-01C: Code Readability Enhancement

Туре	Severity	Location
Code Style	Informational	EarlyAdopterPool.sol:L159, L175, L243

Description:

The <code>EarlyAdopterPool::transferFunds</code> function is meant to be utilized by the <code>EarlyAdopterPool::withdraw</code> functions with an input argument signifying whether the funds should be sent to the depositor or the <code>claimReceiverContract</code>, however, this argument is utilized as a <code>uint256</code> with two literal values (0 or 1, with the latter case applying to all values different than 0).

Example:

src/EarlyAdopterPool.sol

```
SOL
243 function transferFunds(uint256 identifier) internal {
       uint256 rETHbal = userToErc20Balance[msg.sender][rETH];
       uint256 wstETHbal = userToErc20Balance[msg.sender][wstETH];
       uint256 sfrxEthbal = userToErc20Balance[msg.sender][sfrxETH];
       uint256 cbEthBal = userToErc20Balance[msg.sender][cbETH];
       uint256 ethBalance = depositInfo[msg.sender].etherBalance;
       depositInfo[msg.sender].depositTime = 0;
       depositInfo[msg.sender].totalERC20Balance = 0;
       depositInfo[msg.sender].etherBalance = 0;
       userToErc20Balance[msg.sender][rETH] = 0;
       userToErc20Balance[msg.sender][wstETH] = 0;
       userToErc20Balance[msg.sender][sfrxETH] = 0;
       userToErc20Balance[msg.sender][cbETH] = 0;
       address receiver;
       if ( identifier == 0) {
           receiver = msg.sender;
           receiver = claimReceiverContract;
       require(rETHInstance.transfer(receiver, rETHbal), "Transfer failed");
       require (wstETHInstance.transfer (receiver, wstETHbal), "Transfer failed");
```

require(sfrxETHInstance.transfer(receiver, sfrxEthbal), "Transfer failed");

require(cbETHInstance.transfer(receiver, cbEthBal), "Transfer failed");

(bool sent,) = receiver.call{value: ethBalance}("");

require(sent, "Failed to send Ether");

We advise an enum to be utilized instead, achieving the same result albeit with much greater code legibility as well as stricter function behaviour as the EarlyAdopterPool::transferFunds function accepts input arguments greater than 1 when it should not.

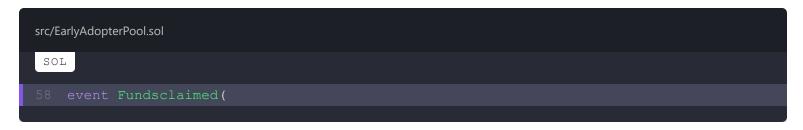
Alleviation:

EAP-02C: Generic Typographic Mistakes

Туре	Severity	Location
Code Style	Informational	EarlyAdopterPool.sol:L58, L71, L243, L314, L322

Description:

The referenced lines contain typographical mistakes (i.e. private variable without an underscore prefix) or generic documentational errors (i.e. copy-paste) that should be corrected.



We advise them to be corrected enhancing the legibility of the codebase.

Alleviation:

EAP-03C: Inefficient Contract TVL Calculation

Туре	Severity	Location
Gas Optimization	Informational	EarlyAdopterPool.sol:L131-L135, L136

Description:

The Total-Value-Locked (TVL) calculation the contract performs in <code>EarlyAdopterPool::deposit</code> is inefficient as it will fetch all the balances held by the contract during the emission of the <code>ERC2OTVLUpdated</code> event and then re-fetch them during the execution of <code>EarlyAdopterPool::getContractTVL</code>.

```
src/EarlyAdopterPool.sol

sol

130 emit ERC20TVLUpdated(

131    rETHInstance.balanceOf(address(this)),

132    wstETHInstance.balanceOf(address(this)),

133    sfrxETHInstance.balanceOf(address(this)),

134    cbETHInstance.balanceOf(address(this)),

135    address(this).balance,

136    getContractTVL()
137 );
```

We advise the calculations of <code>EarlyAdopterPool::getContractTVL</code> to be replicated in the <code>EarlyAdopterPool::deposit</code> function by using the same balances that have already been fetched for the <code>ERC2OTVLUpdated</code> event.

Alleviation:

EAP-04C: Inefficient mapping Lookups

Туре	Severity	Location
Gas Optimization	Informational	EarlyAdopterPool.sol:L124-L125, L148-L149, L208, L210, L218-L219, L244-L247, L249, L251-L253, L255-L258, L302-L306

Description:

The linked statements perform key-based lookup operations on mapping declarations from storage multiple times for the same key redundantly.

Example:

src/EarlyAdopterPool.sol

```
SOL
110 function deposit (address erc20Contract, uint256 amount)
       OnlyCorrectAmount( amount)
       DepositingOpen
       whenNotPaused
            ( erc20Contract == rETH | |
                erc20Contract == wstETH ||
                erc20Contract == cbETH),
       );
       depositInfo[msg.sender].depositTime = block.timestamp;
       depositInfo[msg.sender].totalERC20Balance += amount;
       userToErc20Balance[msg.sender][ erc20Contract] += _amount;
       require(IERC20( erc20Contract).transferFrom(msg.sender, address(this), amount),
       emit DepositERC20 (msg.sender, amount);
           rETHInstance.balanceOf(address(this)),
           wstETHInstance.balanceOf(address(this)),
           sfrxETHInstance.balanceOf(address(this)),
           cbETHInstance.balanceOf(address(this)),
           address (this) .balance,
           getContractTVL()
       );
```

As the lookups internally perform an expensive keccak256 operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the mapping in case of primitive types or holds a storage pointer to the struct contained.

Alleviation:

EAP-05C: Insufficient Documentation of Literal

Туре	Severity	Location
Code Style	Informational	EarlyAdopterPool.sol:L216

Description:

The 2592 variable utilized in the userMultiplier calculation within

EarlyAdopterPool::calculateUserPoints should be relocated to a constant variable declaration with adequate documentation.

Impact:

As an additional point, the maximum multiplier of 200% may not be achievable during the lifetime of the <code>EarlyAdopterPool</code> as it represents a length of 10 months. To achieve a better multiplier factor, the actual duration of the contract's deposit lifetime can be utilized as a divisor of the actual <code>lengthOfDeposit</code> of the user, ensuring a multiplier result guaranteed to be at most 200_0 and at minimum 100_0 via a <code>Math::max</code> operation.

We advise it to be relocated as such, surrounded by text that clearly denotes it is meant to depict the duration that elapses to achieve a 10% increase per month (whose duration is simplified to 30 days).

Alleviation:

EAP-06C: Redundant Data Point

Туре	Severity	Location
Gas Optimization	Informational	EarlyAdopterPool.sol:L166, L184, L207, L323

Description:

The claimingOpen data point is meant to indicate whether EarlyAdopterPool::claim transactions should be possible, however, the same "state" can be validated by evaluating whether the value of claimDeadline is non-zero, a case only satisfied after EarlyAdopterPool::setClaimingOpen has been invoked.

We advise this adjustment to be performed, optimizing the code's storage space and gas cost throughout its functions.

Alleviation:

EAP-07C: Redundant Duplicate Data Points

Туре	Severity	Location
Gas Optimization	Informational	EarlyAdopterPool.sol:L91-L94, L96-L99

Description:

The EarlyAdopterPool contract contains its supported deposit tokens in their address format as immutable variables as well as in their IERC20 format as simple, no-visibility variables.

```
src/EarlyAdopterPoolsol

Sol.

29 address private immutable rETH; // 0xae78736Cd615f374D3085123A210448E74Fc6393;
30 address private immutable wstETH; // 0x7f39C581F595B53c5cb19bD0b3f8dA6c935E2Ca0;
31 address private immutable sfrxETH; // 0xac3e018457b222d93114458476f3e3416abbe38f;
32 address private immutable cbETH; // 0xBe9895146f7AF43049calc1AE358B0541Ea49704;

33

34 //Future contract which funds will be sent to on claim (Most likely LP)
35 address public claimReceiverContract;

36

37 //Status of claims, 1 means claiming is open
38 uint8 public claimingOpen;

39

40 //user address => token address = balance
41 mapping(address => mapping(address => uint256)) public userToErc20Balance;
42 mapping(address => UserDepositInfo) public depositInfo;

43

44 IERC20 rETHInstance;
45 IERC20 sfrxETHInstance;
46 IERC20 sfrxETHInstance;
47 IERC20 cbETHInstance;
```

We advise the contract to solely retain either the <code>IERC20</code> or <code>address</code> counterparts of the tokens, casting the variables to the desirable type (<code>address</code> or <code>IERC20</code> respectively) as needed. We should note that the <code>address</code> and <code>IERC20</code> types are identical at the storage level and can both be set as <code>immutable</code>, they simply serve as syntactic sugar for the Solidity compiler to expose the relevant methods in the case of an <code>interface</code>-type.

Alleviation:

EAP-08C: Redundant Parenthesis Statements

Туре	Severity	Location
Code Style	Informational	EarlyAdopterPool.sol:L117-L120, L283-L287, L307

Description:

The referenced statements are redundantly wrapped in parenthesis' (()).

```
src/EarlyAdopterPool.sol

SOL

117 (_erc20Contract == rETH ||
118    _erc20Contract == sfrxETH ||
119    _erc20Contract == wstETH ||
120    _erc20Contract == cbETH),
```

We advise them to be safely omitted, increasing the legibility of the codebase.

Alleviation:

EAP-09C: Variable Mutability Specifiers (Immutable)

Туре	Severity	Location
Gas Optimization	Informational	EarlyAdopterPool.sol:L96-L99

Description:

The linked variables are assigned to only once during the contract's constructor.

```
src/EarlyAdopterPool.sol

Sol

85    constructor(
86      address _rETH,
87      address _wstETH,
88      address _sfrxETH,
89      address _cbETH
90    ) {
91      rETH = _rETH;
92      wstETH = _wstETH;
93      sfrxETH = _sfrxETH;
94      cbETH = _cbETH;
95

96      rETHInstance = IERC20(_rETH);
97      wstETHInstance = IERC20(_sfrxETH);
98      sfrxETHInstance = IERC20(_cbETH);
99      cbETHInstance = IERC20(_cbETH);
100 }
```

We advise them to be set as immutable greatly optimizing their read-access gas cost.

Alleviation:

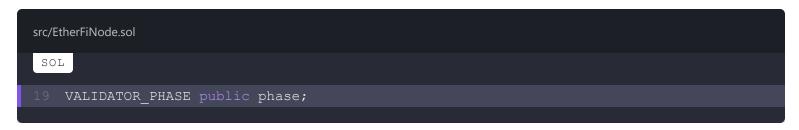
EtherFiNode Code Style Findings

EFN-01C: Generic Typographic Mistakes

Туре	Severity	Location
Code Style	Informational	EtherFiNode.sol:L19, L38, L115

Description:

The referenced lines contain typographical mistakes (i.e. private variable without an underscore prefix) or generic documentational errors (i.e. copy-paste) that should be corrected.



We advise them to be corrected enhancing the legibility of the codebase.

Alleviation:

The EtherFi team examined this exhibit and opted not to apply a remediation in the current iteration of the codebase, instead acknowledging it.

EFN-02C: Ineffectual Conditional Check

Туре	Severity	Location
Gas Optimization	Informational	EtherFiNode.sol:L301

Description:

The referenced require check is ineffectual as it evaluates that an unsigned integer (penaltyAmount) is greater-than-or-equal-to the value of 0 which represents a tautology.

```
src/EtherFiNode.sol

SOL

300 uint256 penaltyAmount = _principal - remaining;

301 require(penaltyAmount >= 0, "Incorrect penalty amount");
```

We advise the require check to either be omitted or instead validate that the penaltyAmount is non-zero, either of which we consider an adequate resolution to this exhibit.

Alleviation:

The ineffectual conditional check has been safely removed from the codebase, optimizing it as a result.

EFN-03C: Ineffectual Usage of Safe Arithmetics

Туре	Severity	Location
Language Specific	Informational	EtherFiNode.sol:L173, L203

Description:

The linked mathematical operations are guaranteed to be performed safely by surrounding conditionals evaluated in either require checks or if-else constructs.

```
src/EtherFiNode.sol

SOL

169 if (_vestedAuctionFee) {
170    uint256 rewards = _getClaimableVestedRewards();

171    uint256 toTnft = (rewards * 29) / 32;

172    tnft += toTnft; // 29 / 32

173    bnft += rewards - toTnft; // 3 / 32

174 }
```

Given that safe arithmetics are toggled on by default in pragma versions of 0.8.x, we advise the linked statements to be wrapped in unchecked code blocks thereby optimizing their execution cost.

Alleviation:

The EtherFi team examined this exhibit and opted not to apply a remediation in the current iteration of the codebase, instead acknowledging it.

EFN-04C: Inefficient Calculation of Rewards

Туре	Severity	Location
Gas Optimization	Informational	EtherFiNode.sol:L211, L261

Description:

The EtherFiNode::getStakingRewardsPayouts and EtherFiNode::getProtocolRewardsPayouts functions will continue execution even if the rewards to be split are 0, inefficiently performing multiple calculations.

Example:

src/EtherFiNode.sol

```
SOL
187 function getStakingRewardsPayouts(
       IEtherFiNodesManager.RewardsSplit memory splits,
       uint256 scale
       onlyEtherFiNodeManagerContract
           uint256 toNodeOperator,
           uint256 toTnft,
          uint256 toBnft,
           uint256 toTreasury
       uint256 balance = address(this).balance;
       uint256 rewards = (balance > vestedAuctionRewards)
           ? balance - vestedAuctionRewards
       if (rewards >= 32 ether) {
           rewards -= 32 ether;
       } else if (rewards >= 8 ether) {
           rewards = 0;
           uint256 operator,
           uint256 tnft,
           uint256 bnft,
           uint256 treasury
       ) = calculatePayouts(rewards, splits, scale);
       if (exitRequestTimestamp > 0) {
           uint256 daysPassedSinceExitRequest = getDaysPassedSince(
               exitRequestTimestamp,
               uint32(block.timestamp)
```

```
treasury += operator;
operator = 0;

227     }

228     }

229

230     return (operator, tnft, bnft, treasury);

231 }
```

We advise the functions to return early if the rewards to be split are 0, optimizing their execution cost.

Alleviation:

While the function returns early in the case of rewards >= 8 ether, the code will still execute if balance > vestedAuctionRewards when it should return early. As such, we consider this exhibit partially alleviated.

EFN-05C: Inefficient Case Handling

Туре	Severity	Location
Gas Optimization	Informational	EtherFiNode.sol:L286

Description:

The case whereby a full year has elapsed since the exit request and exit timestamp of a node is inefficiently handled as the remaining value is set to 0 instead of directly returning the principal as the penalty amount.

Example:

src/EtherFiNode.sol

```
SOL
269 function getNonExitPenalty(
     uint128 principal,
      uint64 dailyPenalty,
      uint32 exitTimestamp
      if (exitRequestTimestamp == 0) {
           return 0;
       uint256 daysElapsed = getDaysPassedSince(
           exitRequestTimestamp,
           exitTimestamp
```

```
273 ) public view onlyEtherFiNodeManagerContract returns (uint256) {
       );
       uint256 daysPerWeek = 7;
       uint256 weeksElapsed = daysElapsed / daysPerWeek;
       uint256 remaining = principal;
       if (daysElapsed > 365) {
           remaining = 0;
            for (uint64 i = 0; i < weeksElapsed; i++) {</pre>
               remaining =
                    (remaining * (100 - dailyPenalty) ** daysPerWeek) /
                    (100 ** daysPerWeek);
           daysElapsed -= weeksElapsed * daysPerWeek;
           for (uint64 i = 0; i < daysElapsed; i++) {
                remaining = (remaining * (100 - dailyPenalty)) / 100;
       uint256 penaltyAmount = principal - remaining;
       require(penaltyAmount >= 0, "Incorrect penalty amount");
       return penaltyAmount;
```

We advise a direct return statement of the principal amount to be performed, optimizing this case's gas cost.

Alleviation:

The <u>principal</u> value is yielded directly in place of the zero-value assignment per our recommendation, optimizing the codebase.

EFN-06C: Inefficient Loop Iterator Data Type

Туре	Severity	Location
Gas Optimization	Informational	EtherFiNode.sol:L288, L295

Description:

The EVM is built to operate on 32-byte data types and any operations on types less than that require additional low-level EVM instructions that increase their gas cost.

Given that the referenced variables are simply iterators, we advise them to be upcast to uint256 variables thus reducing their gas cost.

Alleviation:

The referenced loops are no longer present in the codebase as part of a separate exhibit, rendering this exhibit no longer applicable.

EFN-07C: Loop Iterator Optimizations

Туре	Severity	Location
Gas Optimization	Informational	EtherFiNode.sol:L288, L295

Description:

The linked for loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics (post-0.8.x).

```
src/EtherFiNode.sol

SOL

288 for (uint64 i = 0; i < weeksElapsed; i++) {</pre>
```

We advise the increment / decrement operations to be performed in an unchecked code block as the last statement within each for loop to optimize their execution cost.

Alleviation:

The referenced loops are no longer present in the codebase as part of a separate exhibit, rendering this exhibit no longer applicable.

EFN-08C: Optimization of Penalty Calculation

Туре	Severity	Location
Gas Optimization	Informational	EtherFiNode.sol:L288-L297

Description:

The iterative penalty calculation within **EtherFiNode::getNonExitPenalty** is inefficient as it will split the calculations per-week while they can be split per-month safely.

Example:

src/EtherFiNode.sol

```
SOL
269 function getNonExitPenalty(
      uint128 principal,
      uint64 dailyPenalty,
       uint32 exitTimestamp
273 ) public view onlyEtherFiNodeManagerContract returns (uint256) {
       if (exitRequestTimestamp == 0) {
           return 0;
       uint256 daysElapsed = getDaysPassedSince(
           exitRequestTimestamp,
           exitTimestamp
       );
       uint256 daysPerWeek = 7;
       uint256 weeksElapsed = daysElapsed / daysPerWeek;
       uint256 remaining = principal;
       if (daysElapsed > 365) {
           remaining = 0;
            for (uint64 i = 0; i < weeksElapsed; i++) {</pre>
               remaining =
                    (remaining * (100 - dailyPenalty) ** daysPerWeek) /
                    (100 ** daysPerWeek);
           daysElapsed -= weeksElapsed * daysPerWeek;
           for (uint64 i = 0; i < daysElapsed; i++) {</pre>
                remaining = (remaining * (100 - dailyPenalty)) / 100;
```

uint256 penaltyAmount = principal - remaining;

return penaltyAmount;

require(penaltyAmount >= 0, "Incorrect penalty amount");

A value of 10**75 can safely fit in a uint256 variable, meaning that a calculation of remaining * 10**32 would be safe to perform as long as remaining is less-than-or-equal to 10**43, presented otherwise as 10**25 units of a typical 1e18 asset.

As such, a daysElapsed value of up to 30 can be immediately utilized in the power-to calculation safely without requiring any loop. To further optimize the code for durations greater than a month, we advise a while loop introduced that runs as long as daysElapsed is greater-than 30. Within it, the remaining value should be set directly to

```
(remaining * (100 - _dailyPenalty) ** Math.min(30, daysElapsed)) / (100 ** Math.min(30, daysElapsed))
. The daysElapsed iterator should be subtracted by the same value (Math.min(30, daysElapsed)),
optimizing the EtherFiNode::getNonExitPenalty function's execution significantly. As an added note, the
(100 - _dailyPenalty) value can be stored to a local variable outside the while loop further optimizing
```

Alleviation:

the code's gas cost.

The penalty calculation has been optimized per our recommendation, significantly reducing the gas cost of estimating the non-exit penalty. The EtherFi team chose a value of 7 instead of 30 for each loop's calculation, ensuring a greater degree of safety in the calculations.

EFN-09C: Potentially Incorrect Constants

Туре	Severity	Location
Code Style	Informational	EtherFiNode.sol:L205, L207, L224, L285, L333, L397

Description:

The referenced lines indicate numeric constants in use within the EtherFiNode codebase that appear incorrect and should at minimum be adequately documented.

```
src/EtherFiNode.sol

SOL

394 // While the NonExitPenalty keeps growing till 1 ether,
395 // the incentive to the node operator stops growing at 0.5 ether
396 // the rest goes to the treasury

397 if (bnftNonExitPenalty > 0.5 ether) {
398     payouts[0] += 0.5 ether;
399     payouts[3] += (bnftNonExitPenalty - 0.5 ether);
400 } else {
401     payouts[0] += bnftNonExitPenalty;
402 }
```

In sequence, each variable is assumed to:

The second and the first before last entries of this list are incorrect as a slash operation can be of up to the full amount of a validator and the staking rewards an operator accumulates are uncapped and can exceed ether. For more information, consult the "Weak Validation of Node State" findings in the audit report.

Alleviation:

The EtherFi requested additional guidance in relation to this exhibit. We believe that the second and first before last entries of the list in the exhibit (L207:8 ether and L333:16 ether) are incorrect values.

At minimum, we advise them to be adequately documented and relocated to constant declarations.

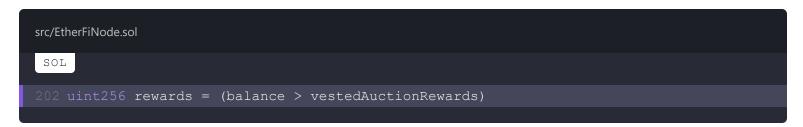
EFN-10C: Redundant Parenthesis Statements

Туре	Severity	Location
Code Style	Informational	EtherFiNode.sol:L202, L399

Description:

The referenced statements are redundantly wrapped in parenthesis' (()).

Example:



Recommendation:

We advise them to be safely omitted, increasing the legibility of the codebase.

Alleviation:

While the redundant parenthesis from the second instance have been removed, they remain in the first instance.

EFN-11C: Repetitive Value Literals

Туре	Severity	Location
Code Style	Informational	EtherFiNode.sol:L205, L206, L290, L291, L296, L348, L363, L397, L398, L399

Description:

The linked value literals are repeated across the codebase multiple times.

Example:

```
src/EtherFiNode.sol

SOL

205 if (rewards >= 32 ether) {
```

Recommendation:

We advise each to be set to its dedicated **constant** variable instead optimizing the legibility of the codebase.

Alleviation:

The EtherFi team examined this exhibit and opted not to apply a remediation in the current iteration of the codebase, instead acknowledging it.

EtherFiNodesManager Code Style Findings

EFM-01C: Inexistent Error Messages

Туре	Severity	Location
Code Style	Informational	EtherFiNodesManager.sol:L98-L104, L112-L118, L277-L280

Description:

The linked require checks have no error messages explicitly defined.

We advise each to be set so to increase the legibility of the codebase and aid in validating the require checks' conditions.

Alleviation:

Proper error messages have been introduced for all referenced require checks.

EFM-02C: Loop Iterator Optimizations

Туре	Severity	Location
Gas Optimization	Informational	EtherFiNodesManager.sol:L163, L184, L249, L274, L378

Description:

The linked for loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics (post-0.8.x).

```
src/EtherFiNodesManager.sol

SOL

163 for (uint256 i = 0; i < _validatorIds.length; i++) {</pre>
```

We advise the increment / decrement operations to be performed in an unchecked code block as the last statement within each for loop to optimize their execution cost.

Alleviation:

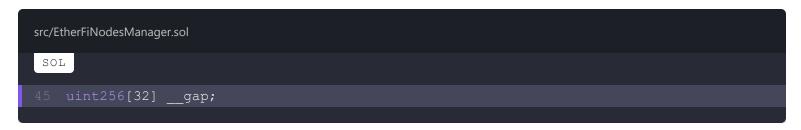
The EtherFi team examined this exhibit and opted not to apply a remediation in the current iteration of the codebase, instead acknowledging it.

EFM-03C: Non-Standard Gap Size

Туре	Severity	Location
Standard Conformity	Informational	EtherFiNodesManager.sol:L45

Description:

The referenced ___gap variable is meant to replicate OpenZeppelin's upgradeability standard by declaring an offset of variables that can be declared at a later point on the same contract without affecting the order of variables in storage in the overall contract.



The methodology employed for calculating the appropriate length for the variable in OpenZeppelin is to utilize the value of 50 as a base and subtract the number of 32-byte slots that are already occupied by the contract (i.e. 1 in the case of TNFT). We advise the size of this variable to be corrected by applying the same methodology and ensuring the length of the gap is directly correlated to the storage layout of the contract it resides in.

Alleviation:

The gap array's length has been adjusted to a standardized value as advised.

EFM-04C: Redundant Parenthesis Statements

Туре	Severity	Location
Code Style	Informational	Ether Fi Nodes Manager. sol: L99-L102, L113-L116

Description:

The referenced statements are redundantly wrapped in parenthesis' (()).

```
src/EtherFiNodesManager.sol

SOL

99 (stakingRewardsSplit.treasury +
100    stakingRewardsSplit.nodeOperator +
101    stakingRewardsSplit.tnft +
102    stakingRewardsSplit.bnft) == SCALE,
```

We advise them to be safely omitted, increasing the legibility of the codebase.

Alleviation:

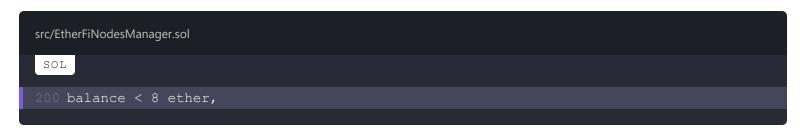
Both redundant parenthesis have been removed from the codebase as advised.

EFM-05C: Repetitive Value Literal

Туре	Severity	Location
Code Style	Informational	EtherFiNodesManager.sol:L200, L282

Description:

The linked value literal is repeated across the codebase multiple times.



We advise it to be set to a constant variable instead optimizing the legibility of the codebase.

Alleviation:

The EtherFi team examined this exhibit and opted not to apply a remediation in the current iteration of the codebase, instead acknowledging it.

NodeOperatorManager Code Style Findings

NOM-01C: Inefficient mapping Lookups

Туре	Severity	Location
Gas Optimization	Informational	NodeOperatorManager.sol:L44, L53-L54, L65, L67, L71, L72, L106, L107

Description:

The linked statements perform key-based lookup operations on mapping declarations from storage multiple times for the same key redundantly.

```
src/NodeOperatorManager.sol

sol

59  /// @notice Fetches the next key they have available to use
60  /// @param _user the user to fetch the key for
61  /// @return the ipfs index available for the validator
62  function fetchNextKeyIndex(
63    address _user
64  ) external onlyAuctionManagerContract returns (uint64) {
65    uint64 totalKeys = addressToOperatorData[_user].totalKeys;
66    require(
67         addressToOperatorData[_user].keysUsed < totalKeys,
68         "Insufficient public keys"
69    );
70

71    uint64 ipfsIndex = addressToOperatorData[_user].keysUsed;
72    addressToOperatorData[_user].keysUsed++;
73    return ipfsIndex;
74  }</pre>
```

As the lookups internally perform an expensive keccak256 operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the mapping in case of primitive types or holds a storage pointer to the struct contained.

Alleviation:

All referenced mapping lookups have been optimized as advised.

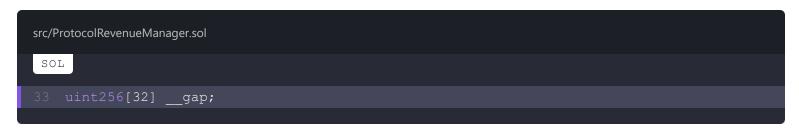
ProtocolRevenueManager Code Style Findings

PRM-01C: Non-Standard Gap Size

Туре	Severity	Location
Standard Conformity	Informational	ProtocolRevenueManager.sol:L33

Description:

The referenced gap variable is meant to replicate OpenZeppelin's upgradeability standard by declaring an offset of variables that can be declared at a later point on the same contract without affecting the order of variables in storage in the overall contract.



The methodology employed for calculating the appropriate length for the variable in OpenZeppelin is to utilize the value of 50 as a base and subtract the number of 32-byte slots that are already occupied by the contract (i.e. 1 in the case of TNFT). We advise the size of this variable to be corrected by applying the same methodology and ensuring the length of the gap is directly correlated to the storage layout of the contract it resides in.

Alleviation:

The gap array's length has been adjusted to a standardized value as advised.

PRM-02C: Optimization of Code Block

Туре	Severity	Location
Gas Optimization	Informational	ProtocolRevenueManager.sol:L174-L178

Description:

The ProtocolRevenueManager::getAccruedAuctionRevenueRewards function's code block is relatively inefficient in its return mechanism.

We advise the code to immediately yield 0 if localRevenueIndex is 0 and to yield the globalRevenueIndex - localRevenueIndex calculation in any other case, rendering the local amount variable redundant and optimizing the code's legibility.

Alleviation:

The code was partially optimized to the version we advised, rendering this exhibit partially alleviated.

PRM-03C: Repetitive Invocation of Getter Function

Туре	Severity	Location
Gas Optimization	Informational	ProtocolRevenueManager.sol:L63, L68

Description:

The referenced getter function is invoked twice in the same function context.

```
src/ProtocolRevenueManager.sol

SOL

60  /// @notice All of the received Ether is shared to all validators! Cool!
61  receive() external payable {
62    require(
63         etherFiNodesManager.numberOfValidators() > 0,
64         "No Active Validator"
65    );
66    globalRevenueIndex +=
67    msg.value /
68    etherFiNodesManager.numberOfValidators();
69 }
```

We advise it to be invoked once, stored to a local variable, and consequently utilized for the two referenced instances thus optimizing the code's gas cost.

Alleviation:

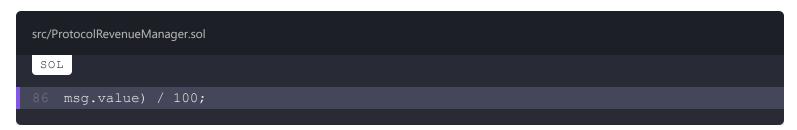
The referenced getter function is now invoked only once and stored to a local variable as advised.

PRM-04C: Repetitive Value Literal

Туре	Severity	Location
Code Style	Informational	ProtocolRevenueManager.sol:L86

Description:

The linked value literal is repeated across the codebase multiple times.



We advise it to be set to a constant variable instead optimizing the legibility of the codebase.

Alleviation:

The EtherFi team examined this exhibit and opted not to apply a remediation in the current iteration of the codebase, instead acknowledging it.

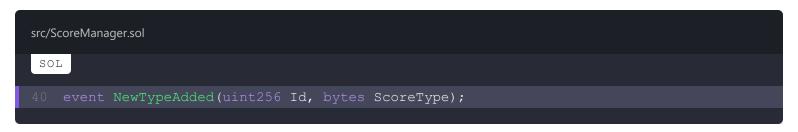
ScoreManager Code Style Findings

SMR-01C: Generic Typographic Mistake

Туре	Severity	Location
Code Style	Informational	ScoreManager.sol:L40

Description:

The referenced line contains a typographical mistake (i.e. private variable without an underscore prefix) or generic documentational error (i.e. copy-paste) that should be corrected.



We advise this to be done so to enhance the legibility of the codebase.

Alleviation:

The EtherFi team examined this exhibit and opted not to apply a remediation in the current iteration of the codebase, instead acknowledging it.

SMR-02C: Ineffectual Usage of Safe Arithmetics

Туре	Severity	Location
Language Specific	Informational	ScoreManager.sol:L96

Description:

The linked mathematical operation is guaranteed to be performed safely by surrounding conditionals evaluated in either require checks or if-else constructs.

```
src/ScoreManager.sol

SOL

95  numberOfTypes++;
96  return numberOfTypes - 1;
```

Given that safe arithmetics are toggled on by default in pragma versions of 0.8.x, we advise the linked statement to be wrapped in an unchecked code block thereby optimizing its execution cost.

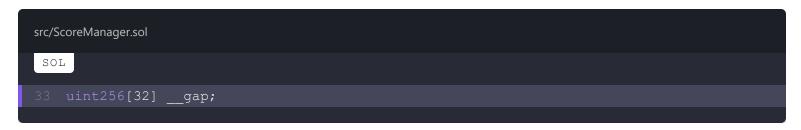
Alleviation:

SMR-03C: Non-Standard Gap Size

Туре	Severity	Location
Standard Conformity	Informational	ScoreManager.sol:L33

Description:

The referenced ___gap variable is meant to replicate OpenZeppelin's upgradeability standard by declaring an offset of variables that can be declared at a later point on the same contract without affecting the order of variables in storage in the overall contract.



The methodology employed for calculating the appropriate length for the variable in OpenZeppelin is to utilize the value of 50 as a base and subtract the number of 32-byte slots that are already occupied by the contract (i.e. 1 in the case of TNFT). We advise the size of this variable to be corrected by applying the same methodology and ensuring the length of the gap is directly correlated to the storage layout of the contract it resides in.

Alleviation:

The gap array's length has been adjusted to a standardized value as advised.

SMR-04C: Redundant Storage Reads

Туре	Severity	Location
Gas Optimization	Informational	ScoreManager.sol:L90, L91, L93, L95, L96

Description:

The referenced instructions all read the numberOfTypes variable from the contract's storage instead of storing it to a local variable for all consequent utilizations.

```
src/ScoreManager.sol

SOL

87  /// @notice creates a new type of score
88  /// @param _type the bytes value type being added
89  function addNewScoreType(bytes memory _type) external onlyOwner returns (uint256) {
90     scoreTypes[numberOfTypes] = _type;
91     typeIds[_type] = numberOfTypes;
92

93     emit NewTypeAdded(numberOfTypes, _type);
94

95     numberOfTypes++;
96     return numberOfTypes - 1;
97 }
```

We advise the numberOfTypes variable to be read once at the beginning of the
ScoreManager::addNewScoreType function and stored to a local numberOfTypes variable that is
consequently utilized in all referenced statements, significantly optimizing the gas cost of the function.

Alleviation:

StakingManager Code Style Findings

SME-01C: Inexistent Error Message

Туре	Severity	Location
Code Style	Informational	Staking Manager. sol: L217

Description:

The linked require check has no error message explicitly defined.

```
src/StakingManager.sol

SOL

217 require(bidIdToStaker[_validatorId] == address(0), "");
```

We advise one to be set so to increase the legibility of the codebase and aid in validating the require check's condition.

Alleviation:

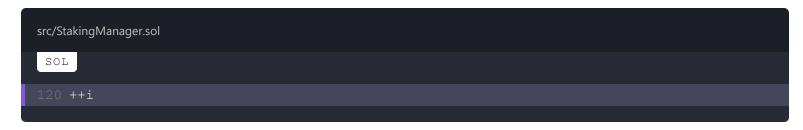
An explicit error message was introduced to the referenced require check as advised.

SME-02C: Loop Iterator Optimizations

Туре	Severity	Location
Gas Optimization	Informational	Staking Manager. sol: L120, L181

Description:

The linked for loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics (post-0.8.x).



We advise the increment / decrement operations to be performed in an unchecked code block as the last statement within each for loop to optimize their execution cost.

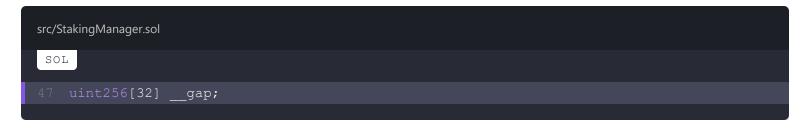
Alleviation:

SME-03C: Non-Standard Gap Size

Туре	Severity	Location
Standard Conformity	Informational	Staking Manager. sol: L47

Description:

The referenced ___gap variable is meant to replicate OpenZeppelin's upgradeability standard by declaring an offset of variables that can be declared at a later point on the same contract without affecting the order of variables in storage in the overall contract.



The methodology employed for calculating the appropriate length for the variable in OpenZeppelin is to utilize the value of 50 as a base and subtract the number of 32-byte slots that are already occupied by the contract (i.e. 1 in the case of TNFT). We advise the size of this variable to be corrected by applying the same methodology and ensuring the length of the gap is directly correlated to the storage layout of the contract it resides in.

Alleviation:

The gap array's length has been adjusted to a standardized value as advised.

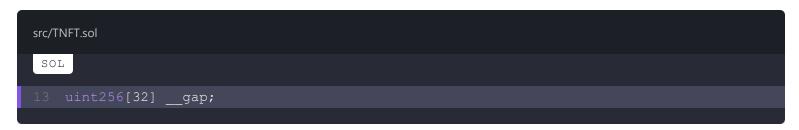
TNFT Code Style Findings

TNF-01C: Non-Standard Gap Size

Туре	Severity	Location
Standard Conformity	Informational	TNFT.sol:L13

Description:

The referenced ___gap variable is meant to replicate OpenZeppelin's upgradeability standard by declaring an offset of variables that can be declared at a later point on the same contract without affecting the order of variables in storage in the overall contract.



The methodology employed for calculating the appropriate length for the variable in OpenZeppelin is to utilize the value of 50 as a base and subtract the number of 32-byte slots that are already occupied by the contract (i.e. 1 in the case of TNFT). We advise the size of this variable to be corrected by applying the same methodology and ensuring the length of the gap is directly correlated to the storage layout of the contract it resides in.

Alleviation:

The gap array's length has been adjusted to a standardized value as advised.

Treasury Code Style Findings

TYR-01C: Redundant Evaluation of Balance

Туре	Severity	Location
Gas Optimization	Informational	Treasury.sol:L16

Description:

```
The Treasury::withdraw function will fail if the _amount specified exceeds the contract's balance (address(this).balance) as the call instruction would fail.
```

We advise the require check to be omitted, optimizing the function's execution cost. Alternatively, if verbose error messages are desirable the check should remain.

Alleviation:

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

External Call Validation

Many contracts that interact with DeFi contain a set of complex external call executions that need to happen in a particular sequence and whose execution is usually taken for granted whereby it is not always the case. External calls should always be validated, either in the form of require checks imposed at the contract-level or via more intricate mechanisms such as invoking an external getter-variable and ensuring that it has been properly updated.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Solidity language boasts that discerns it from other conventional programming languages. For example, the EVM is a 256-bit machine meaning that operations on less-than-256-bit types are more costly for the EVM in terms of gas costs, meaning that loops utilizing a uint8 variable because their limit will never exceed the 8-bit range actually cost more than redundantly using a uint256 variable.

Code Style

An official Solidity style guide exists that is constantly under development and is adjusted on each new Solidity release, designating how the overall look and feel of a codebase should be. In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a

local-level variable contains the same name as a contract-level variable that is present in the inheritance chain of the local execution level's context.

Gas Optimization

Gas optimization findings relate to ways the codebase can be optimized to reduce the gas cost involved with interacting with it to various degrees. These types of findings are completely optional and are pointed out for the benefit of the project's developers.

Standard Conformity

These types of findings relate to incompatibility between a particular standard's implementation and the project's implementation, oftentimes causing significant issues in the usability of the contracts.

Mathematical Operations

In Solidity, math generally behaves differently than other programming languages due to the constraints of the EVM. A prime example of this difference is the truncation of values during a division which in turn leads to loss of precision and can cause systems to behave incorrectly when dealing with percentages and proportion calculations.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Centralization Concern

This category covers all findings that relate to a significant degree of centralization present in the project and as such the potential of a Single-Point-of-Failure (SPoF) for the project that we urge them to re-consider and potentially omit.

Reentrant Call

This category relates to findings that arise from re-entrant external calls (such as EIP-721 minting operations) and revolve around the inapplicacy of the Checks-Effects-Interactions (CEI) pattern, a pattern that dictates checks (require statements etc.) should occur before effects (local storage updates) and interactions (external calls) should be performed last.

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the "client") with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.